INSIDE OS9 LEVEL II

THE INSIDE STORY OF OS9 FOR THE TANDY COLOR COMPUTER 3

by Kevin K. Darling
INSIDE OS9 LEVEL II

THE INSIDE STORY OF OS9 FOR THE TANDY COLOR COMPUTER 3

by Kevin K. Darling
table of contents

1-INTRODUCTION

Section 1-1  Forward and General Info
Section 1-2  Basics of OS9
Section 1-3  The GiME MMU
Section 1-4  DAT Images
Section 1-5  Level Two

2-THE SYSTEM

Section 2-1  Direct Page variables and System memory map.
Section 2-2  System calls (F$calls)
Section 2-3  System calls (I$calls)
Section 2-4  Interrupts

3-DEVICES

Section 3-1  RBF Random Block File Manager
Section 3-2  RBF calls
Section 3-3  SCF Sequential File Manager
Section 3-4  Pipe manager
Section 3-5  General Information

4-WINDOWS

Section 4-1  Window Basics
Section 4-2  Global and CC3IO Memory
Section 4-3  Fonts and things
Section 4-4  Window Descriptor
5-MISCELLANEOUS

Section 5-1  The Shell
Section 5-2  Using Rogue to make OS9 Level II
Section 5-3  Bug Fixes
Section 5-4  Graficom Font Conversion
Section 5-5  User Tips

6-SOURCES

Section 6-1  Alarm Utility
Section 6-2  DMem Utility
Section 6-3  MMap Utility
Section 6-4  PMap Utility
Section 6-5  Proc Utility
Section 6-6  SMap Utility

7-REFERENCE

Section 7-1  GIME Register Map
Section 7-2  Tables and other Stuff
Section 7-3  Video Display Codes
Section 7-4  Keyboard codes
Section 7-5  Error codes
INSIDE OS9 LEVEL II

Introduction
FOREWORD

Around the middle of February, Frank Hogg asked me to do a "little something" on Level Two OS-9 for the CoCo-3. This is the result, a compilation of old and new notes I and others had made for ourselves.

Organizing anything about OS-9 is tough, since each part of it interacts closely with the rest. In the end, I decided to simply present information as a series of essays and tables. Some of these are ones that I had made for L-I, but apply equally well to L-II. Maybe in a half year or so we'll come out with a second edition, but we really wanted to help people out NOW.

To me, at least, it is very like being blind not knowing exactly what occurs during the execution of a program that I have written. For that reason, I have taken a look at OS-9 on the CoCo from the inside out.

The idea is that if you can figure out what's happening on the inside, you have a better chance of knowing what to do from the user level. In essence, this whole collection is a reference work for myself and my friends out there like you.

Level-II wasn't out yet at the beginning of this writing, and I had not seen the Tandy manual until the end, so please bear with me if things have changed somewhat.

In general, I will not duplicate explanations provided by the Tandy manuals, Microware manuals or the Rainbow Guide. Instead, my intention is to enhance them. You should get them, too. Dale Puckett and Peter Dibble are working now on a book about windows for the user. I will be doing more on drivers soon.

This reference work is the result of many hours of studying and probing by myself and others. Hopefully, it will save you at least some of the time and trouble that we have had. Since this is meant as part tutorial, part quick reference, some tables may occur more than once as I felt necessary.

Special thanks are due to Frank Hogg, for publishing this and for being "patient" with delays. I also owe a lot to the many people on CompuServe's OS-9 Forum, who keep asking the right questions.

Thanks also to Pete Lyall for letting me use his excerpts on login, Kent Meyers for much help on internals, and to Chris Babcock for delving into the fonts for us.

And, of course, none of this would have been done without the support and love of my dearest friend and sweetheart, Marsha. Thank you, Sweet Thang!

I hope it helps. Best wishes, and Have Fun.
   Kevin K Darling - 30 March 1987
OVERVIEW OF OS9

The following is all of OS9 in one spot:

UNIVERSAL SYSTEM TABLES:

- Direct page vars - table pointers, interrupt vectors
- Memory bitmaps - maps of free / in-use memory
- Service dispatch tables - vectors for SWI2 system calls
- Module directory - pointers to in-memory modules
- Device table - info on used devices (/D0/P, etc)
- IRQ polling table - vectors interrupts to drivers

PROCESS INFORMATION:

- Process descriptors - process specific information
- Path descriptors - I/O open file information
- Driver static storage - device driver constant memory

PROGRAM MODULES:

- User programs - your program
- Kernal - handles in-memory processing
- Ioman - controls I/O resources
- File Mgrs - file handling and editing
- Drivers - data storage and transfer
- Device descriptors - device characteristics

SIMPLE SYSTEM MEMORY MAP

00000-01FFF System Variables
02000-07DFFF Free memory, bootfile
7E000-7FFFF Kernal
7F000-7FFFF I/O and GIME
# The Main Players:

<table>
<thead>
<tr>
<th>Modules</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL, BOOT</td>
<td>. Reset hardware and Boot</td>
</tr>
<tr>
<td>OS9P1</td>
<td>. Initialization of system</td>
</tr>
<tr>
<td>OS9P2</td>
<td>Handling of most SWI2 service calls (except I/O)</td>
</tr>
<tr>
<td></td>
<td>Memory management and process control</td>
</tr>
<tr>
<td></td>
<td>Module directory upkeep, module searching</td>
</tr>
<tr>
<td></td>
<td>Allocation of process descriptors</td>
</tr>
<tr>
<td>IOMAN</td>
<td>. Handling I/O related SWI2 service calls</td>
</tr>
<tr>
<td></td>
<td>Allocation of path descriptors</td>
</tr>
<tr>
<td></td>
<td>IRQ polling table entries</td>
</tr>
<tr>
<td></td>
<td>Device IRQ polling</td>
</tr>
<tr>
<td></td>
<td>Device table entries for desc, driver, filmgr</td>
</tr>
<tr>
<td></td>
<td>Queuing processes trying to use same path desc</td>
</tr>
<tr>
<td></td>
<td>Allocation of driver static memory</td>
</tr>
<tr>
<td></td>
<td>Copying device desc init table to path desc</td>
</tr>
<tr>
<td></td>
<td>Calling file mgr for I/O calls</td>
</tr>
<tr>
<td>RBFMAN</td>
<td>. Allocation of data buffers</td>
</tr>
<tr>
<td>SCFMAN</td>
<td>File &amp; directory allocation and management</td>
</tr>
<tr>
<td>PIPEMAN</td>
<td>Edit, seek, read, write of file</td>
</tr>
<tr>
<td></td>
<td>Queuing processes trying to use same device</td>
</tr>
<tr>
<td>CC3DISK</td>
<td>. Allocation of verify buffer</td>
</tr>
<tr>
<td>CC3IO</td>
<td>Read / write of data buffers from / to device</td>
</tr>
<tr>
<td>PRINTER</td>
<td>Device interrupt handling</td>
</tr>
<tr>
<td>RS232</td>
<td>Device status / error monitoring</td>
</tr>
<tr>
<td>REL</td>
<td>- Resets hardware, calls OS9p1</td>
</tr>
<tr>
<td>INIT</td>
<td>- Data module containing system constants</td>
</tr>
<tr>
<td>BOOT</td>
<td>- Load OS9Boot if initial dir's, paths fail</td>
</tr>
<tr>
<td>CC3GO</td>
<td>- CHX CMDS, Startup, Autoex, Shell</td>
</tr>
<tr>
<td>CLOCK</td>
<td>- System timekeeping, VIRQ's, Alarm calls</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Descriptors</td>
<td>- info on each process</td>
</tr>
<tr>
<td>Path Descriptors</td>
<td>- info local to each I/O path</td>
</tr>
<tr>
<td>Device Table</td>
<td>- device memory, desc, filmgr, driver</td>
</tr>
<tr>
<td>Polling Table</td>
<td>- device status address, driver IRQ vector</td>
</tr>
<tr>
<td>Module Directory</td>
<td>- address, user cnt of program modules</td>
</tr>
</tbody>
</table>
MULTI-TASKING PRINCIPLES

The power of the 6809's addressing modes enables the m/l programmer to easily write code that will execute at any memory address. Furthermore, if the code is written to access program variables by offsets to the index registers, more than one user can execute that code as long as he has his own data area.

The point of all this is that the 6809 made it easy for Microware to write an operating system that can load a program anywhere there is enough contiguous memory, assign the user a data space, and through SWI2 (trap) calls, access system I/O and memory resources.

Now, since we know that we can be processing code and sharing the 64K memory space with other programs, we can allow more than one program/user at more or less the same time by switching between the processes fast enough to appear to each user that he has his own computer.

How often is fast? In some other multi-tasking systems, each process is responsible for signaling to the operating system kernel that it was ready to give up some of its CPU time. The advantage of this method was that time-critical code wasn't interrupted. (OS9 users can simply shut off interrupts if this is necessary.) But this method depends on the user to write the switching signal into his code so that it was hit often enough to give other processes a chance to run.

In OS9, there is always a system 'clock' that interrupts the 6809 about 10 times a second, and causes the next process to be given a CPU time slice.* Other interrupts from any I/O devices needing service cause the system to execute the interrupt service routine in the driver for that device, and quickly resume the original process.

Switching between processes is the easy part. Each process has a process descriptor, holding information about it. When the 6809 is interrupted, the current address it is at in the program, and the CPU's registers are saved on the system stack in the process's data area. The stack pointer's value is saved in the current program's process descriptor for later retrieval.

The kernal then determines who gets the next time slice according to age and priority. The stack pointer of the new main process is loaded from its process descriptor, and since the stack pointer is now pointing to a 'snapshot' of its process's registers, a RTI instruction will cause the program to continue as if nothing had ever stopped it.

So, in essence, each process thinks that it is alone in the machine with its own program and data area limits defined, although if needed, it can find limited info on the others. Besides device interrupts and normal task-switching, two other events may have an effect on a program's running without its knowing about it: I/O queuing and untrapped signals.

* Actually 60 times/second on the CoCo, but a process time slice is considered to be 6 'ticks', or 1/10th second.
MULTI-TASKING PRINCIPLES
PROCESS QUEUES/STATES

PROCESS QUEUES

These are just what they sound like - an ordered arrangement of programs. They are kept in a linked list, that is, each has a pointer to the next in line. When a process changes queues, the process descriptor itself isn't moved, just the pointers are.

A process is always in one of three major queues (except for the current process):

Active - Normal running; gets its turn in varying amounts of the total processor time according to its age, priority, and state.

Sleeping - A program has put itself to Sleep for a specified tick count, or until it gets a signal. (As in waiting for its I/O turn)

Waiting - Special Sleep state that terminates on a signal or child's death / F$Exit. Entered via F$Wait.

STATES

The P$State byte in a process's descriptor has different bits set depending on what the program is doing, where it is currently executing, and what external occurrences have affected it.

A process has one or more of these state attributes:

SysState %1000 0000 Is using system resources, or is being started/aborted by the kernel.

TimSleep %0100 0000 Asleep: awaiting signal, sleep over.

TimOut %0010 0000 Has used up its time slice. This is a temporary flag used by the kernel.

Suspend %0000 1000 Continues to age in active queue, but is passed over for execution. Used in place of Sleep and Signal calls in someL-II drivers.

Condem %0000 0010 Has received a deadly signal, dies by a forced F$Exit call as soon as it is no longer in a system state.

Dead %0000 0001 Is already unexecutable, as its data and program areas have been relinquished by an F$Exit call. The process descriptor is kept so that the death signal code may be passed to the parent on F$Wait.

The System State is a privileged mode, as the kernel doesn't make the process give up the next time slice, but instead lets it run continuously until it leaves the system state.
The reason for this is that the process is servicing an interrupt, changing the amount of free memory, or doing I/O to a device, and thus should be allowed to run until it is safe to change programs, or it has released the device for other use.

It is because of the System State that interrupts are allowed almost always. Any driver interrupt code acts as an "outside" program that temporarily takes over the CPU, but the current process is not changed and will continue when the driver is finished taking care of the interrupt source.

MULTI-TASKING PRINCIPLES

I/O

If two or more processes want to do input/output/status operations on the same device, all except the first will have to wait in line (queue). Under OS9, IOMAN and the file managers are responsible for this control.

Each open path has a path descriptor associated with it. This is a 64-byte packet of information about the file. Because OS9 allows a path that has been opened to a file or device to be duplicated, and used by another process, several programs may be talking about the same path (and path descriptor). Provision must be made to queue an I/O attempt using the same path. (The most common instance of this is with /TERM.)

Since all I/O calls pass through the system module IOMAN, the I/O manager, it checks a path descriptor variable called PD.CPR to see if it is clear, or not in use. If it is in use, the process in inserted in a queue to await it's turn.

Here the process descriptor plays a part. Two of its pointers are used here: P$IOQP (previous link) and P$IOQN (next link). P$IOQP is set to the ID of the process just ahead of this one, and the P$IOQN of the process ahead in line is set to this one's ID, forming a chain (linked list) of process ID pointers waiting to use this particular device.

When a process has made it through a manager to the point that the manager must do I/O through a device driver, it checks a flag in the driver's static (permanent) storage called V.BUSY. If it is clear, no one is using the device at that instant, and V.BUSY is set to the process's internal ID number.

If V.BUSY is not clear (another process got there first and is waiting for it's call to finish), the manager inserts the process in an I/O queue to wait its turn.

When the process (executing the file manager) is through with the device, it clears V.BUSY, and all the processes waiting in line are woken up to try again. As far as I know, V.Busy only becomes very important if a driver has put it's process to sleep, as otherwise the program would have exclusive access while within a system call anyway.

Thus a process seeking use of a device and its driver must wait FIRST for the path to be clear, and THEN for the device used by that particular path. If two processes are talking to two different files, or have each opened their own paths and the file is considered shareable, they will only have to wait in line for device use.

Again, it should be noted that once one process has started I/O operations, it has near-total use of the CPU time, except of course for interrupt routines or if it goes to sleep in the driver or a queue.
MULTI-TASKING PRINCIPLES

Signals are communication flags, as the name implies. Since processes operate isolated from each other, signals provide an asynchronous method of inter-process flagging and control.

Commonly used signals include the Kill and the Wakeup codes. Wakeup is essential to let the next process in an I/O queue get its turn in line at a path or device.

OS9 has a signal-sending call, F$Send, which sends a one byte signal to the process ID specified, and causes the recipient to be inserted in the active process queue. Any signal other than Kill or Wake is put in the P$Signal byte of its process descriptor.

If it was the Kill signal, the P$State byte in the process descriptor has the Condemned bit set to alert the kernel to kill that process. A Wake signal clears the P$Signal byte, since just making the destination an active process was enough.

Signals are not otherwise acted upon until the destination process returns to the User state. (It'd be unwise to bury a process in the midst of using the floppy drives, for instance.) However, drivers and the kernel may take note of any pending signals and alter their behavior accordingly.

When the kernel brings a process to the active state, the P$Signal byte in the descriptor is checked for a non-zero value (Kill=0, but the Condemned bit was set instead, causing a rerouting to the F$Exit 'good-bye' call as soon as the killed process enters a non-system state). The process is given a chance to use the signal right off.

If the program has done a F$Icpt call to set a signal trap, a fake register stack is set up below the process's real one, holding the signal, data area and trap vector: P$Signal, P$SigDat, P$SigVec. The kernel then does its usual RTI to continue the program where it left off.

Instead, the program picks up at the signal vector where it usually stores the signal in the data area for later checking when convenient (totally up to the programmer, though). The trap routine is itself expected to end with a RTI, thus finally getting back to the normal flow of execution by pulling the real registers that are next on the stack.

If the program has NOT done a F$Icpt call, the kernel drop-kicks it into F$Exit, the same as a Kill signal does.

SIGNALS:

<table>
<thead>
<tr>
<th>Code</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S$Kill Abort process (cannot be trapped)</td>
</tr>
<tr>
<td>1</td>
<td>S$Wake Insert process in Active process queue</td>
</tr>
<tr>
<td>2</td>
<td>S$Abort Keyboard abort (Break Key)</td>
</tr>
<tr>
<td>3</td>
<td>S$Intrpt Keyboard interrupt (Shift-Break)</td>
</tr>
<tr>
<td>4</td>
<td>S$Window Window has changed</td>
</tr>
<tr>
<td>5-255</td>
<td>user defineable so far</td>
</tr>
</tbody>
</table>
## INITIATING A PROCESS

<table>
<thead>
<tr>
<th>#</th>
<th>VAR</th>
<th>MOD</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P$ID</td>
<td>OS9</td>
<td>Allocates a 64-byte process descriptor.</td>
</tr>
<tr>
<td></td>
<td>P$User</td>
<td></td>
<td>Copy parent's user index</td>
</tr>
<tr>
<td></td>
<td>P$Prior</td>
<td></td>
<td>and priority.</td>
</tr>
<tr>
<td></td>
<td>P$Age</td>
<td></td>
<td>Age set to zero.</td>
</tr>
<tr>
<td></td>
<td>P$State</td>
<td></td>
<td>State of process is System State.</td>
</tr>
<tr>
<td></td>
<td>D.Proc</td>
<td></td>
<td>Current process desc is now this one.</td>
</tr>
<tr>
<td></td>
<td>P$DIO</td>
<td></td>
<td>Copies parent's default directory ptrs.</td>
</tr>
<tr>
<td>2</td>
<td>P$PATH</td>
<td>IOMAN</td>
<td>Called three times to I$Dup the first 3 paths of the parent (std in, out, error).</td>
</tr>
<tr>
<td>3</td>
<td>P$SWI</td>
<td>OS9</td>
<td>Make these 3 vectors = D.UsrSvc (0040).</td>
</tr>
<tr>
<td></td>
<td>P$SWI2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P$SWI3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P$Signal</td>
<td></td>
<td>Clear process's signal, signal vector.</td>
</tr>
<tr>
<td></td>
<td>P$SigVec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>P$PModul</td>
<td></td>
<td>F$Link to desired program module.</td>
</tr>
<tr>
<td>4a</td>
<td>P$PModul</td>
<td>IOMAN</td>
<td>F$Load from xdir if not in memory.</td>
</tr>
<tr>
<td>5</td>
<td>OS9</td>
<td></td>
<td>Error end if not Program/System module.</td>
</tr>
<tr>
<td>6</td>
<td>P$ADDR</td>
<td>OS9P2</td>
<td>F$Mem request to &gt;= data area needed.</td>
</tr>
<tr>
<td></td>
<td>P$PagCnt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>P$SP</td>
<td>OS9</td>
<td>Copy parameters to top of new data area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set stack pointer to RTI stack registers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set up RTI stack with register values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PC - module entry point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>U - start of data area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y - top of data area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X - parameters pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DP - start of data area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D - length of parameters passed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SP-&gt; CC - interrupts okay, E flag for RTI</td>
</tr>
<tr>
<td>8</td>
<td>D.Proc</td>
<td></td>
<td>Put back parent as current process.</td>
</tr>
<tr>
<td></td>
<td>P$CID</td>
<td></td>
<td>Get PARENT's other child, and</td>
</tr>
<tr>
<td></td>
<td>P$SID</td>
<td></td>
<td>make it new proc's sibling link.</td>
</tr>
<tr>
<td></td>
<td>P$PID</td>
<td></td>
<td>( PARENT's new P$CID = new P$SID )</td>
</tr>
<tr>
<td></td>
<td>P$PID</td>
<td></td>
<td>Copy parent's ID to new proc desc.</td>
</tr>
<tr>
<td>9</td>
<td>P$State</td>
<td></td>
<td>State of new is no longer System State.</td>
</tr>
<tr>
<td></td>
<td>P$Queue</td>
<td></td>
<td>Return new child's ID to parent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F$AProc - insert process in active queue.</td>
</tr>
</tbody>
</table>
INSIDE OS9 LEVEL II
INTRODUCTION
Section 2

OS9 I/O

OPENING A FILE/DEVICE

<table>
<thead>
<tr>
<th>#</th>
<th>VAR</th>
<th>MOD</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PD.PD</td>
<td>IOMAN</td>
<td>Allocates a 64-byte block path descriptor</td>
</tr>
<tr>
<td></td>
<td>PD.MOD</td>
<td></td>
<td>Sets access mode desired.</td>
</tr>
<tr>
<td></td>
<td>PD.CNT</td>
<td></td>
<td>Sets user cnt=1 for this path desc.</td>
</tr>
<tr>
<td>2</td>
<td>PD.DEV</td>
<td>IOMAN</td>
<td>Attaches the device (drive) used.</td>
</tr>
<tr>
<td></td>
<td>V$STAT</td>
<td></td>
<td>Allocates memory for device driver (CCDisk)</td>
</tr>
<tr>
<td></td>
<td>V.PORT</td>
<td></td>
<td>Sets device address in driver static memory</td>
</tr>
<tr>
<td>3</td>
<td>V.xxxx</td>
<td>DRIVER</td>
<td>The driver's init subroutine is called to</td>
</tr>
<tr>
<td></td>
<td>V.xxxx</td>
<td></td>
<td>initialize the device and static memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If device uses IRQ's, uses F$IRQ call:</td>
</tr>
<tr>
<td>4</td>
<td>Q$FOLL</td>
<td>OS9</td>
<td>Sets up IRQ polling table entry.</td>
</tr>
<tr>
<td></td>
<td>Q$FRTY</td>
<td></td>
<td>( address, flip &amp; mask bytes, service add,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>static storage, priority of IRQ )</td>
</tr>
<tr>
<td>5</td>
<td>V$DRIV</td>
<td>IOMAN</td>
<td>Sets up rest of device table.</td>
</tr>
<tr>
<td></td>
<td>V$DESC</td>
<td></td>
<td>( module addresses of desc, driver, mgr)</td>
</tr>
<tr>
<td></td>
<td>V$FMRG</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>V$USR</td>
<td></td>
<td>Sets user count of device=1</td>
</tr>
<tr>
<td>6</td>
<td>PD.OPT</td>
<td>IOMAN</td>
<td>Copies device desc info to path desc.</td>
</tr>
<tr>
<td></td>
<td>PD.SAS</td>
<td></td>
<td>( default values: drive #, step rate,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sides, baud rate, lines/page, etc. )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Calls file managr Open subroutine:</td>
</tr>
<tr>
<td>7</td>
<td>PD.BUF</td>
<td>FLMGR</td>
<td>Allocates buffer for file use.</td>
</tr>
<tr>
<td></td>
<td>PD.DVT</td>
<td></td>
<td>Copies device table entry for user.</td>
</tr>
<tr>
<td></td>
<td>PD.FST-</td>
<td></td>
<td>Opens file for use, and sets up</td>
</tr>
<tr>
<td></td>
<td>PD.xxx</td>
<td></td>
<td>file mgr pointers and variables.</td>
</tr>
<tr>
<td>8</td>
<td>P$PATH</td>
<td>IOMAN</td>
<td>Puts path desc # in proc desc I/O table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns table pointer to user as path nmbr.</td>
</tr>
</tbody>
</table>

2,3,4,5 only if first time for that device,
else V$USR = V$USR + 1
PD.DEV = Device table entry
4 only if device uses IRQ's
GIME DAT

The memory management abilities of the CoCo-3 are the source of it's ability to run Level-II. To help explain what a DAT is, and it's usefulness, here's a text file I first posted on the OS9 Forum on 5 August 86.

Q: What is the difference between the 512K boards that are sold now and the 512K CoCo-3?

LOGICAL vs PHYSICAL ADDRESSES ---

To understand the difference, you must first keep in mind that the 6809, having 16 address lines, can only DIRECTLY access 64K of RAM. The only way for the CPU to use any extra memory is to externally change the address going to the RAM.

The address coming from the CPU itself is called the Logical Address. The converted address presented to the RAM is called the Physical Address.

For instance, the CPU could read a byte from $E003 in it's 64K Logical Address space, but external hardware could translate the $E003 into, say, a Physical Address of $1B003, by looking up the entry for the 4K block $E in a fast RAM table.

A coarser, but more familiar, example to CC owners is the $FFDF (64K RAM) 'poke'. The SAM chip can address 96K of Physical memory (64K RAM and 32K ROM). When that register was written to, the SAM translated all accesses to memory in the Logical (CPU) range of $8000-$FFE7 to Physically point to the other 32K bank of RAM, instead of the ROM. A similar example is the use of the Page Bit register, to translate Logical accesses to $0000-$7FFF into using the other Physical 32K bank of RAM.

MEMORY MANAGEMENT ---

The hardware that does the actual translation between the Logical --> Physical addresses is called a Memory Management Unit (MMU). In the case above, the SAM was the MMU. One common type of hardware MMU is called a DAT, for Dynamic Address Translation. A DAT consists of a Task Register and some fast look-up RAM. It's called Dynamic partly because the translation table is not fixed, but can be modified. I'll go into more detail on a DAT later.

THE COCO-2 BOARDS ---

The memory expansions sold for the CC2 are an extremely simple form of a DAT. Most only allow the upper or lower 32K of Logical Addresses to access a different upper or lower 32K bank of Physical Memory. Leaving out I/O addresses and ROM for the moment, their 64K modes simplistically look like: (for 256K)
INSIDE OS9 LEVEL II
INTRODUCTION
Section 3

Logical (CPU) XXXX
Address:

$FFFF +--------+--------+
I I I I I I
I U0 I U1 I U2 I U3 I Upper 32K Banks
I I I I I I
I I I I I I example: CPU access of
$0100

$8000 +--------+--------+
using Bank 2 = L2+$0100
I I I I I I is RAM address $20100.
I I I I I I
I L0 I L1 I L2 I L3 I Lower 32K Banks
I I I I I I

$0000 +--------+--------+
$0001 10010 20000 30000 Physical (RAM) Hex Address

The Physical memory that the CPU addressed is chosen from a combination of (L0 or L1 or L2 or
L3) AND (U0 or U1 or U2 or U3). Some boards would mostly only allow the selection of Banks
in number pairs (eg: L1+U1, L2+U2), or keeping L0 constant, and varying the Upper (U0-U3).

The important point here is that you could not 'mix & match' the Banks (Upper appear as Lower,
Lower as Upper, or say, map U2 from $0000-$7FFF and U3 as $8000-$FFFF).

To use data from one bank to another generally required the copying of that data. This is why most
applications of the extra memory were as RamDisks, or extra data storage, NOT as programs.
(Tho you could have four different copies of the Color Basic ROMS for example, or four different
OS9 '64K machines' running one at a time.)

THE COCO-3 DAT ---

To make the most economical use of the available RAM, and make the most use of reentrant (used
by more than one process at a time) and position-independent (runnable at any address, possibly
using a different data area) programs or sections of data, the DAT has to be much more flexible
than the Bank switching schemes above.

For instance, in the example given of four copies of the Basic ROMS, what if you had not
modified the Extended Color ROM? You would have wasted 24K of RAM (3 banks x 8K) on
extra copies. (Actually, you wasted 32K, since it'd be even better just to keep the original ROM 'in
place'.) Or what if you really wanted one ROM copy and seven 32K RAM program spaces? Or
you need to temporarily map in 32K of video RAM? Or keep seven different variations of the Disk
ROM, which would all (at least on a CC2) need to made to appear at $C000 up?

And we haven't even discussed OS9 yet!

What have we figured out? We need both smaller translation 'blocks' and a way of making those
physical blocks appear to the CPU at any logical block size boundary.
INSIDE OS9 LEVEL II
INTRODUCTION
Section 3

What size should a block be? So far, it seems that the smaller the better for a programmer or operating system, because that could leave more 'free blocks' left over for other use. This will become apparent later, in the Level-II discussion. Many Level-II machines use a 4K block. The CoCo-3 uses an 8K block size. In most cases, this may not be restrictive, except perhaps on a base 128K machine.

And so we come to the CoCo DAT. Here's a simple diagram:

```
+--------+                  +--------+
 I       I                  I       I
 I CPU I A13-A15 R0-R2 I----I  I Task# I  VIDEO ADDR
 I  I--------/-----/---->I  DAT I P13-P18 +=-1-----
 I I I (3 addr) I RAM I----/-<--I RAM I 512K
 I I
 I I  +--------+ I ADD I-->RAM
 I  I A0-A12 --------------- I MUX I
 I  I--------/-----------------/-<--I I
 +--------+                  +--------+

/ ............ GIME ............ /
```

As shown, the DAT RAM would be 8 six-bit words x 2 tasks (explained below).

From left to right, the Logical Addresses from the CPU are translated into a extended Physical Address to access the RAM.

The upper 3 CPU lines (A13-A15) are used to tell the DAT which 8K Logical Block is being used (1 of 8 in a 64K map) and act as DAT RAM address (R0-R2) lines. At that Logical Block address in the DAT is a 6-bit data word, which forms the extended Physical Address lines P13-P18. The lower CPU address lines are passed thru as is to point within the 8K RAM block (out of the 512K RAM) selected by P13-P18.

Note that 6 bits can form 64 block select words. Multiply 64 possible blocks by 8K per block, and there's your 512K RAM. You may write any 6-bit value to each of the 8 DAT RAM locations, thus choosing which of the 64 8K-blocks you wish to appear within the 8K address block the CPU wishes to access. You could even write the same value several times, making the same 8K physical RAM show up at different logical CPU addresses.

The Task number acts as the DAT R3 address line, and simply allows selection between 2 sets of eight DAT RAM words. This makes it simpler to change between 64K maps. Normally, you can software select the Task number.

AN ANALOGY ---

Okay, this has been rough on some of you, and my explanation may need some explaining <grin> so a simpler analogy is in order:

Let's say you have a fancy new TV cabinet with 8 sets from bottom to top in it. You can watch all 8 at a time. (This makes you the CPU, and each screen is 8K of your logical 64K address space.)
Ah, but each set also has 64 channels! So you can tune each set to ANY of the channels, or several to the SAME channel. (Each channel is like one 8K block out of the 64 available to you in a 512K machine.) When you tune in a program, you are said to have "mapped it in".

An analogy to the Task Register would be if each set had TWO channel selectors A and B, and you had one switch to select whether ALL the sets used their A or B setting. This is generally called "task switching". If you wanted to switch to a C,D, or E task, you'd have to get up and retune all 8 sets on their A or B selectors (all A or all B), possibly from a list (called a "DAT Image") you had made from TV Guide.

Get it now? The CC2 512K expansions would then be like the same cabinet, only the top or bottom four sets always tune together and only have 8 selector positions; the same eight channels per same position. Which would you buy?

NOW I HAVE IT! --

BUT WHAT USE IS ALL THIS?

So far, we've seen that the 64-8K blocks can be arranged any which way that you'd like to see them, 8 at a time. As a quick example of what could be done, let's see how a text editor might work. We'll assume the upper 32K is RSDOS always, and not to be touched, to keep this simple.

This leaves us with 32K, or four 8K blocks for our program and data (the text). In our example, we'll make the editor code itself just under 24K long, which leaves us only 8K for text. So, here's the map:

```
E000-FFFF logical block 7   hires cmds & I/O
C000-DFFF   6  disk basic
A000-BFFF   5  color basic
8000-9FFF   4  extended basic
6000-7FFF   3  editor
4000-5FFF   2  editor
2000-3FFF   1  editor
0000-1FFF   0  text
```

(Note that this is kind of unrealistic, since you'd probably not want to have the text down in RSDOS variable territory, but this is just an extremely simple example, okay?)

Okay, you type in 8K of text. Normally, that'd be all you could do, but remember that we can make any Physical 8K Block map into any Logical 8K Block. So the editor, when it realizes that it's buffer is almost full, could tell the GIME MMU to make a different RAM block (out of the 64, minus those used by Basic for text, etc) appear to the CPU in our logical block 0 (from $0000-$1FFF).

Even if Basic uses up 8 actual RAM blocks for it's own use, and the editor uses 3, we still could use (64-11) or 53x8K blocks. That's over 400K of text space! By swapping real (physical) RAM into our 64K (logical) map like this, the only limitation on spreadsheets, editors, etc, is that the programmer must respect the 8K block boundaries.

Hmmm... you say. I could even swap in different editor programs, if I had to, couldn't I? You bet. Now you're starting to get an inkling of how Microware did Level-II.
OK, WHAT ABOUT OS9 LEVEL-II?

L2 gives each process up to 64K to work with. It allocates blocks of memory (you got it - up to eight 8K blocks!) for that process to use as program or data areas.

Having 512K of memory does NOT mean you could do a "basic09 #200k" command line. The CPU can still only access 64K at a time, but the space not used by Basic09 (which itself is about 24K long) is usable for data. So about 64K minus 24K is about 40K, which is very big for a Basic09 program.

Notice a gotcha here, though. If Basic09 was 25K long, then you'd have much less data area possible. Why? Remember the 8K blocks! A 25K program would map in using four 8K blocks (three wouldn't be enough), using up 32K of your 64K map. The same goes if you asked for 9K of data space. You'd get two 8K blocks of RAM mapped in, taking up 16K of CPU space. Aha! Now you understand why the smaller the block size the better.

Back to the good parts. Remember that most OS9 programs are reentrant and position-independent. This means that no matter how many processes or terminal-users want to use a certain program, only ONE copy needs to be in memory. (Check the difference: if you had 10 Basic09 programs running, each needing 30K of data space - they'd need only 24K for B09 + 10*30K, versus 10*(24K+30K), a 216K savings!) The Amiga's programs, for example, aren't reentrant. It'd need 540K.

As far as making 200K virtual programs, there ARE ways of doing that. You could start other processes (Forking), or map in different data modules. Even better, you can pre-Load modules, and by Linking and Unlinking them, they will swap in and out of your 64K address space, a technique much faster than using RamDisks. (A Loaded module is off in RAM somewhere, but not in your map until Linked to.) This is what Basic09 does, by the way, so by writing a program that calls lots of small subprograms, each would get swapped in automatically as you needed them. Instant 400K basic!

TOO MUCH TO SAY ---

Well, there's about a zillion other things I wanted to put in here, like how the page at $FE00-$FEFF is across all maps, to make moving data easier (some move code is there); or how each Level-II process or block of programs has a DAT Image associated with it, that can be swapped into the DAT RAM; or that up to 64K is allocated to the System Task, where the Kernal and Drivers and buffers are; or the neat tricks you could do using the DAT; or show you a possible memory map using the DAT; or about how interrupts switch to the System Task.

(Some of this IS covered in this new collection - Kevin)
DAT IMAGES and TASKS

It may seem that we're spending a lot of space on the DAT, but it's very important to the whole of L-II. So...

As you now know, the DAT in the CoCo-3 allows you to specify which of up to eight blocks will appear in the 6809's logical address map when their numbers are stored and enabled in the GIME's MMU or DAT.

Ideally, an MMU would have enough ram to handle the maps for any conceivable number of programs, modules or movement. But ram that fast is expensive and uses lots of power. So a compromise was made -- in the GIME's case, two sets of DAT registers. That is, two complete 64K maps can be stored and switched between at will.

You will surely need one map for the system plus another for a shell at least. So how does OS9 handle the needs of all the other programs you want to run? By swapping sets of block numbers into the DAT as needed.

The set of block numbers is stored in a packet of information called a DAT Image. Because various OS9 machines use different size blocks (2K, 4K, 8K, are most frequent) and have differing amounts of memory blocks available, a DAT Image can vary in size even though a process descriptor has 64 bytes available for one.

On the CoCo-3, it's 16 bytes long, made up of 8 two-byte entries. The first byte of each entry is usually zero, while the second byte is the physical block number. The exception is when an entry contains a special value of $333E$, which is used to indicate that that logical block is unused as memory for that map.

When expanding the amount of blocks allocated to a map, OS9 checks for the special $333E$ flag bytes. That's how it knows where to place new blocks in the DAT Image.

DAT Images are created for several purposes. The one that affects you the most is the image stored in a process descriptor. Whenever a process comes up in the queue for running, it's DAT image is copied to one of the two sets of GIME task map registers. Then that set is enabled by setting the task register select. Instantly the new logical map is the one seen by the CPU. When a process' timeslice is up, it also gives up the use of the task number.

The task register number used for the process DAT image is usually the same number stored in the P$\text{Task}$ byte in other L-II computers. On the CoCo-3 however, P$\text{Task}$ contains the number of a virtual or fake DAT task map. There are 32 of these, which make it appear as though the GIME had 32 sets of map registers.

If the images are already in the process descriptors, why have virtual tasks? Because it's simpler for the system to look them up in a known table versus searching all over.

The first two virtual DAT tasks (0 and 1) are reserved for the system's use. The first is for the usual kernel, drivers, descriptors, buffers. The second is for GrfDrv's screen and buffer access.

So on the CoCo-3, the task number refers to a table entry that points to the DAT Image to be used. Except for special cases, the pointer is to the image within a process descriptor.
Another use for the images is in the module directory. Unlike Level One, where the entry could simply contain the module's address within the 64K you had, Level Two entries point to a DAT Image of the block or blocks containing the module and any others loaded with it.

While a module file is being loaded, OS9 temporarily allocates a process descriptor and a task number for it. The file is then read into blocks of memory that F$Load has requested. Then the descriptor & task are released, leaving the modules in a kind of "no-man's-land", waiting to be mapped into a program's space.

The visible residue of loading a file of modules is that the free memory count goes down, and any new modules found are entered into the system map's module directory. Otherwise, they don't directly affect a process map until linked into it.

Each Module Directory entry is made up of:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00-01</td>
<td>MD$MPDAT</td>
<td>Module DAT Image Pointer</td>
</tr>
<tr>
<td>02-03</td>
<td>MD$MBSz</td>
<td>Block size total</td>
</tr>
<tr>
<td>04-05</td>
<td>MD$MPtr</td>
<td>Module offset within Image</td>
</tr>
<tr>
<td>06-07</td>
<td>MD$Link</td>
<td>Module link count</td>
</tr>
</tbody>
</table>

A program such as Mdir can use these to display what it does about the modules in memory. First, it gets the module directory using F$GModDr. Then by using the DATImage and offset associated with an entry, Mdir F$Move's the header and name from the blocks where the module has been loaded.

The Mdir example illustrates a third common usage of images, moving data into your program's map for inspection.

Anytime you need to "see" memory external to your process (sorry, you can only legally read it; no writes), you can create a DAT image of your own and use it with F$Move. OS9 will take the offset and amount you pass, and copy that amount over to your map from the offset within the image you made.

In the case of Mdir, the image was moved over by F$GModDr along with the module directory entries. So there's no need to build an image in that case. Just use the MD$MPDAT pointer.

You may also in some cases request movement of data between maps using a reference to a Task number instead. OS9 itself will internally index off the tasks' images for you.

Notice that throughout this section, the image is used over and over simply to allow the cpu to read or write to extended memory.

In the next section, we'll see some examples of DAT Images and maps.
LEVEL TWO IN MORE DETAIL

I will be using "L-II" for Level Two, and "One" for Level One, so as to make differentiating the names a little easier as you read. Other word definitions I use here are (loosely):

space - any 6809 logical 64K address area.
mapping, mapped in - causing blocks to appear in a space.
a map - a space containing mapped-in modules/RAM blocks.
system map - the 64K map containing the system code.
task - a particular map with a certain program and data area
task number - number of a particular task map.
DAT map - a task ready to use thru the hardware/software
    enable of the task number's map.
task register - task number stored here to enable a DAT map.
user code - the programs/data you use (applications).
system code - the programs/data the system uses (file mgrs,
    drivers, descriptors, and the kernel FS & FS Calls,
    IRQ handlers, and scheduling codes).

LEVEL TWO vs ONE: General

The core of understanding L-II is in understanding the separation and handling of 8K blocks, and their use in logical 64K spaces. And why.

DAT -

Under One, you only had 64K of contiguous physical RAM in one 64K logical map. L-II uses the DAT to map any physical 8K blocks of RAM containing program and data modules into a 64K logical address map. When a program's turn to run comes up, the block map data (called a DAT Image) for it's 64K space is copied to and/or enabled in the GIME's DAT.

SWI's -

L-II was designed to run most programs written for One, which is possible since system calls are made using a software interrupt call, passing parameters (via cpu registers pushed on a memory stack) that are pointed to by the 6809's SP register. This gives two advantages over Level One:

1: Virtually none of the system code has to reside in the 64K space containing the user's program and data areas. The system map is switched in place of the caller's map.

2: OS/9 needs only to know the caller's SP and task number (both kept in the caller's process descriptor in the system map) to access the parameters passed, or to move data between the two maps.

(Note that a kernal could be written to do simply this on any CoCo that had the Banker or DSL Ram expansion, etc. But you'd lose the advantage of the smaller flexibly-mapped blocks provided by the GIME's DAT.

The corollary advantage, and the "why" of L-II, is that each user program can have almost an entire 64K space to itself and it's data area, as can also the system code.
THE SYSTEM TASK MAP:

Up to 63.75K of kernel, bootfile (drivers, mgsrs, etc).
I/O buffers.
Descriptors.
System vars & tables.

System calls and other interrupts temporarily "flip" the program flow into this task map. User parameters and R/W data copied from/to system ram for drivers and file managers to act upon.

EACH USER TASK MAP:

Up to 63.5K total for each program and it's pgmdata area. Each task map made out of up to 8 module or pgmdata blocks (8K each) that are mapped in from the 64 (minus those used by the system task or other user tasks) blocks available in a 512K machine.

THE SYSTEM MAP

Oddly enough, the system map is close to what you're used to under Level One. Memory is allocated for buffers and descriptors in pages just as before. The main difference is that no user programs (should) share space here, as they did under Level One.

You still have the Direct Page variables from $0000-00FF along with other system global memory just above it up to $1FFF Towards the top ($7777-FFFF) we run into descriptors, buffers, polling tables, and finally the I/O modules and the kernal. A CoCo-III Level Two System Map looks like this:

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-0FFF</td>
<td>Normal L-II System Variables</td>
</tr>
<tr>
<td>1000-1FFF</td>
<td>New CC3 global mem and CC3I0 tables</td>
</tr>
<tr>
<td>2000-xxxx</td>
<td>free ram</td>
</tr>
<tr>
<td>xxxx-DFFF</td>
<td>Buffers, proc desc, bootfile</td>
</tr>
<tr>
<td>E000-FDFF</td>
<td>REL, Boot, OS9</td>
</tr>
<tr>
<td>FE00-FEFF</td>
<td>Vector page (top of OS9pl)</td>
</tr>
<tr>
<td>FF00-FFFF</td>
<td>I/O and GIME registers</td>
</tr>
</tbody>
</table>

Some areas of special interest include the ...

Vector Page RAM:

This page of RAM is mapped across ALL 64K maps. This "map-global" RAM is necessary so that no matter what other blocks are mapped in place of the system code, there is always a place for interrupts (hardware or software) to go and execute the special code in OS9pl that switches over to the system task.
THE SYSTEM TASK MAP:

Up to 63.75K of kernel, bootfile (drivers, mgrs, etc).
I/O buffers.
Descriptors.
System vars & tables.

System calls and other interrupts temporarily "flip" the program flow into this task map. User parameters and R/W data copied from/to system ram for drivers and file managers to act upon.

EACH USER TASK MAP:

Up to 63.5K total for each program and it's pgmdata area. Each task map made out of up to 8 module or pgmdata blocks (8K each) that are mapped in from the 64 (minus those used by the system task or other user tasks) blocks available in a 512K machine.

THE SYSTEM MAP

Oddly enough, the system map is close to what you're used to under Level One. Memory is allocated for buffers and descriptors in pages just as before. The main difference is that no user programs (should) share space here, as they did under Level One.

You still have the Direct Page variables from $0000-00FF along with other system global memory just above it up to $1FFF Towards the top (????-FEFF) we run into descriptors, buffers, polling tables, and finally the I/O modules and the kernal. A CoCo-III Level Two System Map looks like this:

0000-0FFF Normal L-II System Variables
1000-1FFF New CC3 global mem and CC3IO tables
2000-xxxx free ram
xxxx-DFFF Buffers, proc desc, bootfile
E000-FDFF REL, Boot, OS9
FE00-FEFF Vector page (top of OS9pl)
FF00-FFFF I/O and GIME registers

Some areas of special interest include the ...

Vector Page RAM:

This page of RAM is mapped across ALL 64K maps. This "map-global" RAM is necessary so that no matter what other blocks are mapped in place of the system code, there is always a place for interrupts (hardware or software) to go and execute the special code in OS9pl that switches over to the system task.
BlockMap:

In a 512K CoCo OS/9 has 64 RAM blocks of 8K each to choose from (8K x 64 = 512K). Each is known by a number from 00-3F. The blockmap is a table of flags indicating the current status of each of these blocks, which could be ...

FREE RAM = Ram blocks not in use as Module/ PgmData areas.

RAM IN USE = Ram blocks in use for either:

Modules - Blocks that contain program, subroutine, or data modules. MDIR will show these. Before a module is used, it will have been loaded into free ram blocks. On link or run, those blocks are then mapped into (made to appear in) any task's space. A data module mapped into several maps can provide inter-task vars. Subroutine mods (like for RUNB) can be linked/unlinked, in/out of a task map.

Data - Free ram that has been mapped into a task space for use as pgm data areas. Normally these blocks are only mapped into one task space (unlike module blocks). These blocks will be released to the free RAM pool when the program using them exits.

DAT Images:

Since each task map requires knowing which (of up to 8) blocks are to be mapped in for that process (yes- system code execution is also a process), AND since OS/9 must know in which blocks that program modules have been loaded into, OS/9 keeps individual tables or "images" of those block numbers.

Each Image has 8 slots, two bytes each. A special block number, $333E, is used to designate an unused logical block for that task.

Module Directory:

In Level One, the module directory simply had to point to the module's address. Under L-II, it points to the DAT Image table showing the block(s) the module is physically in and it's beginning offset within the DAT Image logical 64K map.

Process Descriptors:

A descriptor contains pretty much the same info as it did under L-One, but adds the DAT Image for that process, which will be set into the DAT when it's turn to run comes up.

There is also a local process stack area, used while in the system state (executing system code after a system call). This is because the process's real stack is of course in another map, and a local stack is needed if the process were interrupted or went to sleep.
SYSTEM MEMORY ALLOCATION

As I said above, the system map is still allocated internally in pages. However, when you first boot up, it usually will only have about 5 blocks mapped in. Something like:

<table>
<thead>
<tr>
<th>Logical Address</th>
<th>Physical Block(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-1FFF</td>
<td>00 - block 00 is always here</td>
</tr>
<tr>
<td>2000-7FFF</td>
<td>02 - no ram needed here yet</td>
</tr>
<tr>
<td>8000-FFFF</td>
<td>01,02,03 - this is your bootfile, first vars</td>
</tr>
<tr>
<td>E000-FEFF</td>
<td>3F - block 3F always contains the kernel</td>
</tr>
</tbody>
</table>

The system process descriptor of course has the DAT Image that corresponds to this block map.

Any RAM left over in blocks allocated for loading the bootfile is taken by page for system use. For instance, the device table normally is just below the bottom of the boot.

Once you begin running several processes and opening files, the system must allocate more RAM for descriptors and buffers. When all the pages that are free in the blocks already mapped in are used up, OS9 maps in another block, which is then also sub-allocated by page.

Page allocation is still used because buffers, descriptors and tables usually are a page or two size, just as under Level One. So it’s still the best use of available memory.

USER MAPS

MODULE and DATA AREAS

Each user process has the use of a map made up of up to eight 8K blocks. However, it is seldom that all eight are in use (certain basicOS and graphics programs excepted).

More likely, each task map will look like:

<table>
<thead>
<tr>
<th>Logical Address</th>
<th>Physical Block(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-1FFF</td>
<td>?? - 8K data area</td>
</tr>
<tr>
<td>2000-FFFF</td>
<td>?? - no ram needed here yet</td>
</tr>
<tr>
<td>E000-FEFF</td>
<td>?? - block containing program</td>
</tr>
</tbody>
</table>

Again, the process descriptor DAT Image has a copy of the block numbers actually used (instead of ??).

Unlike Level One, RAM for a user process is NOT allocated by page. There’s no need to, for two reasons. First, the data area is not shared with any other process.

Second, no memory can be used from any left over in the program block. Many people ask why not? Hey, they say, since you can map a block anywhere, why can’t some other program take advantage of the unused RAM? The answer is basically that it would just take too many resources to keep track of what module should stay because part of the block was being used for data.
END OF OS9 LEVEL II
INTRODUCTION
Section 5

Even more importantly, what if a program requested more memory while it was running? You'd be stuck, as data areas must be contiguous and any modules within that block would be in the way. One more reason: Level Two was designed to take advantage of modules in ROM. So there's no way to assume that RAM is available in that block.

So, the upshot is that data areas are allocated from any free RAM blocks in the machine, and always 8K at a time. Even if your program only needed two pages to run in, it still gets a block. Now you can see that the smaller the block the better, as in this case having 4K blocks would leave more free RAM for other programs to use.

Just like in Level One, programs end up at the highest logical address possible in a map, and data areas at the bottom. For the same reason as in One, this is done to allow the data area to grow as much as possible if needed.

One very important point to make at this time: since all modules that were loaded together are also mapped into spaces together, it pays to keep module files close to an 8K boundary. More details on this are in the MISC TIPS section at the end of the book.

SWITCHING BETWEEN MAPS

Okay, now we come to the nitty-gritty of Level-Two. This is where we tie together all we've talked about so far. But it's not tough, so don't worry.

Let's say that a program is running in it's own map, and wishes to use a system call for I/O. How does the code get over to the system map where the drivers are?

An OS9 system call is simply a software interrupt. What that means is that what the program is doing and where it's at is saved in the process' memory on a stack of variables.

Then, like all interrupts, program flow is redirected (by reading the CoCo's BASIC ROM, specially mapped in just long enough to get the addresses) to the vector page at logical address FE00 which is at the top of all maps.

The code within that page is part of OS9p1 and it knows that it should change the GIME task register select to task 0, which is always the system map. As soon as it does that, all the kernel, file managers, drivers etc are accessible to the CPU, which will come down out of the vector page to complete your system call. If needed, OS9 will go back to code located in the vector page where it can map in your user task long enough to get and put data.

At the end of the call, the system code jumps back up into the vector page, maps your process' DAT Image back into the GIME's task map 1, then enables task register 1 which allows your program space to reappear to the CPU.

Then the saved registers are taken back off the stack in your map, and your program continues.

If you want to, you can think of Level Two as really giving your program 128K of RAM, as the net effect compared to Level One is just that... under One, your program had to share space with the drivers and kernel, and any system calls stayed within the same old 64K map. Under Two, your program jumps between 64K maps when you make a system call.
INSIDE OS9 LEVEL II
INTRODUCTION
Section 5

One side note: because of the manipulation of the GIME's MMU and the necessity of copying much data between maps, L-II is normally slower than Level One. However, the CoCo-3 makes up for this as it runs at twice the speed of our older CoCo's.

EXAMPLE MAPS

Here are some example process, module and memory maps generated by the programs I've included in the back of this book. Study them and you can see the relationship between what is reported by each utility. They should help give you a better feel as to what's going on in your machine.

EXAMPLE ONE:

I had two shells running, and of course the particular utility that was printing out at the time.

<table>
<thead>
<tr>
<th>ID</th>
<th>Prnt</th>
<th>User</th>
<th>Pty</th>
<th>Age</th>
<th>St</th>
<th>Sig</th>
<th>Module</th>
<th>Std in/out</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 00</td>
<td>Shell</td>
<td>&lt;TERM</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 00</td>
<td>Shell</td>
<td>&gt;TERM</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>128</td>
<td>128</td>
<td>80</td>
<td>0 00</td>
<td>Proc</td>
<td>&lt;W7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt;W7</td>
</tr>
</tbody>
</table>

Below's my PMAP output. The numbers across the top (01 23 etc) are short forms of (0000-1FFF, 2000-3FFF) addresses in each task's logical map. Notice that there are indeed eight 8K block places in each map, but only those blocks that are needed are mapped in (and are in the DAT image of that process, which by the way, is where the map information is gotten by PMAP).

<table>
<thead>
<tr>
<th>ID</th>
<th>01 23 45 67 89 AB CD EF</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00 ... 04 01 02 03 3F</td>
<td>SYSTEM</td>
</tr>
<tr>
<td>2</td>
<td>05 ... 06 ...</td>
<td>Shell</td>
</tr>
<tr>
<td>3</td>
<td>07 ... 06 ...</td>
<td>Shell</td>
</tr>
<tr>
<td>4</td>
<td>0A ... 08</td>
<td>PMap</td>
</tr>
</tbody>
</table>

Now, notice that in the SYSTEM map is Block 00 = system global variables, Block 3F = kernal, Blocks 01,02,03 = bootfile, and Block 04 plus probably part of 01, = system data and tables.

In the shell and pmap lines, we see that Blocks 05,07,0A are being used for data. Block 06 must contain the Shell, and Block 08 must contain Pmap. We can confirm all this by looking at the module directory output below and comparing block numbers:
Module Directory at 00:03:51
Blk Offset Size Ty Rd At Uc Name

3F D06 12A C1 1 r 0 0 REL - the kernel
3F E30 1D0 C1 1 r 0 1 Boot
3F 1000 ED9 C0 8 r 0 0 OS9p1
01 300 CAE C0 2 r 0 1 OS9p2 - boot modules
01 FAE 2E C0 1 r 0 1 Init
01 6947 1EE C1 1 r 0 2 Clock
01 6B35 1AE 11 1 . 0 1 CC3Go
06 0 5FC 11 1 r 0 3 Shell - the Shell file
06 5FC 2E7 11 1 r 0 0 Copy
06 1E10 2D 11 1 r 0 0 Unlink
06 0 28E 11 1 l 0 1 Proc - my cmd's file
06 435 1B1 11 1 r 0 0 MMMap
06 5E6 1F8 11 1 r 0 0 PMap
06 7DE 1D5 11 1 r 0 0 SMMap
06 9B3 136 11 1 r 0 0 DMem
06 A67 240 11 1 r 0 0 Dump
09 0 1FFC C1 1 r 0 1 GrvDrv - grfdrv is alone

Using my MMAP command, we can see below how many blocks are left for the OS9 system to use. Take notice of the block 3E being allocated... that's the video display RAM block.

RAM for video is allocated from higher numbered blocks, since there is a better chance of finding contiguous RAM that way. Normally, blocks don't have to be together for OS9 to use them, but the GIME requires that screen memory be that way for display.

0123456789
# = = = == == == == == == == == == == ==
0 U U U U U U M M M U ... ... <--blocks 00-0A
1 ... ... ... ... ... ... ... ... ...
2 ... ... ... ... ... ... ... ... ...
3 ... ... ... ... ... ... ... ... ... U U <--3E, 3F

Number of Free Blocks: 51
Ram Free in KBytes: 408

EXAMPLE TWO

This real example I ran off the other day. I had five shells, all of which had started another process (by me typing it in).

<table>
<thead>
<tr>
<th>ID</th>
<th>Prnt</th>
<th>User</th>
<th>Pty</th>
<th>Age</th>
<th>St Sig</th>
<th>Module</th>
<th>Std in/out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 0 Shell  &lt;TERM &gt;TERM</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>128</td>
<td>130</td>
<td>80</td>
<td>0 0 Shell  &lt;W7  &gt;W7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 0 Shell  &lt;W4  &gt;W4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 0 pix    &lt;W4  &gt;W4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 0 pix    &lt;TERM &gt;TERM</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 0 Shell  &lt;W5  &gt;W5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>128</td>
<td>128</td>
<td>80</td>
<td>0 0 pix    &lt;W5  &gt;W5</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0 0 Shell  &lt;W6  &gt;W6</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>128</td>
<td>128</td>
<td>80</td>
<td>0 0 Proc   &lt;W7  &gt;O1</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>9</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>C0</td>
<td>0 0 Ball   &lt;W6  &gt;W6</td>
</tr>
</tbody>
</table>
Note the high block numbers in most of the programs. Each window was showing an Atari ST picture in it, and process #11 had Steve Bjork's bouncing ball demo running.

True windows that use GrfInt and Grfdrv are NOT mapped into a program's space. But this was special, as I was running many VDGInt screens, which usually ARE mapped in (on purpose) so that the programs could directly access the video display.

Notice also that my System task had fully been allocated by block. The SMAP later shows what part of them was free.

```
<table>
<thead>
<tr>
<th>ID</th>
<th>01 23 45 67 89 AB CD EF</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00 31 11 04 01 02 03 3F</td>
<td>SYSTEM</td>
</tr>
<tr>
<td>2</td>
<td>05 . . . . . . . . . .</td>
<td>Shell</td>
</tr>
<tr>
<td>3</td>
<td>07 . . . . . . . . . .</td>
<td>Shell</td>
</tr>
<tr>
<td>4</td>
<td>09 . . . . . . . . . .</td>
<td>Shell</td>
</tr>
<tr>
<td>5</td>
<td>0E . . . 3A 3B 3C 3D 0D</td>
<td>pix</td>
</tr>
<tr>
<td>6</td>
<td>0F . . . 36 37 38 39 0D</td>
<td>pix</td>
</tr>
<tr>
<td>7</td>
<td>10 . . . . . . . . . .</td>
<td>Shell</td>
</tr>
<tr>
<td>8</td>
<td>12 . . . 32 33 34 35 0D</td>
<td>pix</td>
</tr>
<tr>
<td>9</td>
<td>13 . . . . . . . . . .</td>
<td>Shell</td>
</tr>
<tr>
<td>10</td>
<td>18 . . . . . . . . . 19</td>
<td>PMap</td>
</tr>
<tr>
<td>11</td>
<td>14 16 17 . . . . . . 31 15</td>
<td>Ball</td>
</tr>
</tbody>
</table>
```

The other point to note is that the Tandy-provided shell file (block 06) goes over the block size-512 byte limit, and thus cannot be mapped into the top block slot, because it would fall on top of the vector page and I/O area from FE00-FFFF.

Here's the MMAP output. Lots of video ram allocated, huh?

```
0123456789ABCD
#-----------------
0UUUUUMMMMMMUU
1UUUUUMUUMU
2-----------------
3--------------
```

Number of Free Blocks: 23
Ram Free in KBytes: 184
And just to show how close I was to a real limit, here's the SMAP utility output. It shows in pages how much memory is left in the system task map. The 32x16 old-style VDG text screens and all the process descriptors (two pages each!), plus a page for each window's SCF input buffer made things rather tight.

```
  0 1 2 3 4 5 6 7 8 9 A B C D E F
   --- --- --- --- --- --- ---
  0 U U U U U U U U U U U U U U
  1 U U U U U U U U U U U U U U
  2 U U U U U U U U U U U U U U
  3 U U U U U U U U U U U U U U
  4 --- --- --- --- --- --- ---
  5 U U U U U U U U U U U U U U
  6 U U U U U U U U U U U U U U
  7 U U U U U U U U U U U U U U
  8 U U U U U U U U U U U U U U
  9 U U U U U U U U U U U U U U
  A U U U U U U U U U U U U U U
  B U U U U U U U U U U U U U U
  C U U U U U U U U U U U U U U
  D U U U U U U U U U U U U U U
  E U U U U U U U U U U U U U U
  F U U U U U U U U U U U U U U
```

Number of Free Pages: 19
Ram Free in KBytes: 4
INSIDE OS9 LEVEL II

The System
### L-II PROCESS DESCRIPTOR VARIABLES

<table>
<thead>
<tr>
<th>Offset</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P$ID</td>
<td>Process ID</td>
</tr>
<tr>
<td>1</td>
<td>P$PID</td>
<td>Parent's ID</td>
</tr>
<tr>
<td>2</td>
<td>P$SID</td>
<td>Sibling's ID</td>
</tr>
<tr>
<td>3</td>
<td>P$CID</td>
<td>Child's ID</td>
</tr>
<tr>
<td>4-05</td>
<td>P$SP</td>
<td>Stack Pointer storage</td>
</tr>
<tr>
<td>6</td>
<td>P$Task</td>
<td>Task Number</td>
</tr>
<tr>
<td>7</td>
<td>P$PagCnt</td>
<td>Data Memory Page Count</td>
</tr>
<tr>
<td>8-09</td>
<td>P$User</td>
<td>User Index</td>
</tr>
<tr>
<td>0A</td>
<td>P$Prior</td>
<td>Priority</td>
</tr>
<tr>
<td>0B</td>
<td>P$Age</td>
<td>Age</td>
</tr>
<tr>
<td>0C</td>
<td>P$State</td>
<td>Status</td>
</tr>
<tr>
<td>0D-0E</td>
<td>P$Queue</td>
<td>Queue Link (next process desc ptr)</td>
</tr>
<tr>
<td>0F</td>
<td>P$IOQP</td>
<td>Previous I/O Queue Link (Proc ID)</td>
</tr>
<tr>
<td>10</td>
<td>P$IOQN</td>
<td>Next I/O Queue Link (Proc ID)</td>
</tr>
<tr>
<td>11-12</td>
<td>P$PModul</td>
<td>Primary Module pointer</td>
</tr>
<tr>
<td>13-14</td>
<td>P$SWI</td>
<td>SWI Entry Point</td>
</tr>
<tr>
<td>15-16</td>
<td>P$SWI2</td>
<td>SWI2 Entry Point</td>
</tr>
<tr>
<td>17-18</td>
<td>P$SWI3</td>
<td>SWI3 Entry Point</td>
</tr>
<tr>
<td>19</td>
<td>P$Signal</td>
<td>Signal Code</td>
</tr>
<tr>
<td>1A-1B</td>
<td>P$SigVec</td>
<td>Signal Intercept Vector</td>
</tr>
<tr>
<td>1C-1D</td>
<td>P$SigDat</td>
<td>Signal Intercept Data Address (U)</td>
</tr>
<tr>
<td>1E</td>
<td>P$DeadLK</td>
<td>Dominant proc ID for locked I/O</td>
</tr>
<tr>
<td>20-2F</td>
<td>P$DIO</td>
<td>Default I/O ptrs (chd, chx)</td>
</tr>
<tr>
<td>30-3F</td>
<td>P$Path</td>
<td>I/O Path Table (real path numbers)</td>
</tr>
<tr>
<td>40-7F</td>
<td>P$DATImg</td>
<td>DAT Image (only 16 used in CoCo-3)</td>
</tr>
<tr>
<td>80-9F</td>
<td>P$Links</td>
<td>Block Link counts (for user map) (8 used)</td>
</tr>
<tr>
<td>A0-AB</td>
<td></td>
<td>Network variables?</td>
</tr>
<tr>
<td>AC</td>
<td></td>
<td>Path number (0,1,2) for selected window</td>
</tr>
</tbody>
</table>

---

rmb $200-. Local stack

P$Stack equ 512 Top of Stack

P$Size equ 512 Size of Process Descriptor
INSIDE OS9 LEVEL II
The System
Section 1

There are three main differences between a L-I and Level Two process descriptor. The L-II additions are:

. DAT Image - so OS9 knows what to map in for the process.
. Link Cnts - so an unlinked won't unmap blocks with other
still-linked-into-this-map modules.
. Stack area- used while in the system state.

The link counts apply to that process map only, and are counts of block links, not individual modules. Say you had a merged module file loaded with Runb, Syscall and Inkey all together taking up two blocks. The first logical block number of the whole group will have a link count of one.

Then perhaps your program calls Inkey. Inkey is found in your map already, and the first block number link count is incremented in the process descriptor. The module directory link count is incremented also.

Now Inkey finishes and is unlinked. The link count is decremented in the module directory and could easily now be zero. But you don't want Runb and Syscall to go away, too! And they won't because the process map block link now only goes down to one again, and so both blocks mapped will stay mapped.

The stack area is needed when an interrupt (software or hardware) occurs. The initial register save will be within the process' stack area. Then OS9 flips over to the system map, where, in case this process' time is up and it's whole state must be saved, OS9 begins using the process descriptor stack area instead.

In a way, the process descriptor stack is an extension of the process data area into the system map.

Under L-I, of course, there was no need for this, as everyone's stack was available at all times.

L-II Direct Page Variable Map $00XX

* Names are standard L-II. Defs with no name are new CC3 vars.

<table>
<thead>
<tr>
<th>Addr</th>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-21</td>
<td>D.Tasks</td>
<td>Task Proc User Table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Points to 32 byte task# map.</td>
</tr>
<tr>
<td>22-23</td>
<td>D.TmpDAT</td>
<td>Temporary DAT Image stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used to point to images used in moves.</td>
</tr>
<tr>
<td>24-25</td>
<td>D.Init</td>
<td>INIT Module ptr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Points to the Init module.</td>
</tr>
<tr>
<td>26-27</td>
<td>D.Poll</td>
<td>Interrupt Polling Routine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vector to IOMan sub to find IRQ sources.</td>
</tr>
<tr>
<td>28</td>
<td>D.Time</td>
<td>System Time Variables:</td>
</tr>
<tr>
<td>28</td>
<td>D.Year</td>
<td>Year</td>
</tr>
<tr>
<td>29</td>
<td>D.Month</td>
<td>Month</td>
</tr>
<tr>
<td>2A</td>
<td>D.Day</td>
<td>Day</td>
</tr>
<tr>
<td>2B</td>
<td>D.Hour</td>
<td>Hour</td>
</tr>
<tr>
<td>2C</td>
<td>D.Min</td>
<td>Minute</td>
</tr>
<tr>
<td>2D</td>
<td>D.Sec</td>
<td>Seconds</td>
</tr>
</tbody>
</table>

2-1-2
INSIDE OS9 LEVEL II
The System
Section 1

2E D.Tick  Tick countdown for slice
       60 Hz IRQ count. (60 ticks = 1 second)
2F D.Slice  Current slice remaining
           Ticks left for current process normal run.
30 D.TSlice  Ticks per Slice constant
           Set to 6 = 1/10 second per process slice
32 D.MotOn  Drive Motor time out
36-37 Boot start address
38-39 Boot length
           New variables for use by os9gen & cobbler.

40-41 D.BlkMap  Memory Block Map
           Points to 64 byte physical block flag array.
44-45 D.ModDir  Module Directory
           Points to the 8 byte dir entries start.
48-49 D.PrcDBT  Process Descriptor Block Table
           Points to 256 byte array of msb addresses.
4A-4B D.SysPrC  System Process Descriptor
           Points to proc desc used while in SysState.
4C-4D D.SysDAT  System DAT Image
           Points to the image within D.SysPrC desc.
4E-4F D.SysMem  System Mem Map
           Points to 256 byte page table for systm map.
50-51 D.ProC  Current Process Desc
           Points to the proc desc in use now.
52-53 D.AProcQ  Active Process Queue
           First proc desc link of procs ready to run.
54-55 D.WProcQ  Waiting Process Queue
           First proc desc link of procs that P3Wait'd.
56-57 D.SProcQ  Sleeping Process Queue
           First proc desc link of procs sleeping.
58-59 D.ModEnd  Module Directory end

5A-5B D.ModDAT  Module Directory DAT image end

6B-6C "Boot Failed" REL vector
           Vector to display of this message.
6B-7C CoCo reset code
           55 NOP NOP B7 FF DF 7E 9F00E
80-81 D.DeVTbl  I/O Device Table
           Points to array of 9-byte device entries.
82-83 D.PollTbl  I/O Polling Table
           Points to array of 9-byte IRQ poll entries.
88-89 D.PthDBT  Path Descriptor Block Table ptr
           Points to base 256-byte path desc's table.
8A D.DMAReq  DMA Request flag (MFI slot use)
           Set= MFI slot has been changed. CC3Disk flag.

90 GIME register copies:
91 Init Reg $FF91 shadow for tasks
92 irqEN  $FF92 shadow IRQ enables
93-9F other GIME shadows

A0 Speed flag (1=2Mhz)
A1-A2 Task DAT Image Ptrs Table ptr
           Pointer to 32 image pointers for task #'s.
A3  0-128K, 1-512K temp flag
A4  FF91 Task Reg Bit (which system state task)
A5-A6  Global CC310 memory
       Pointer to $1000: global mem.
A7-A8  Grfdrv SP storage
       Pointer to end of global mem. sysmap 1 stack.
A9-AA  Grfdrv ->kernal return vector
AB-AC  Kernal ->grfdrv second sysmap
AD-AE  Clock SvcIRQ vector for VIRQ
AF  GIME IRQ bits status
     Set bit = unpooled interrupt as yet.
B0-B1  VIRQ table
       Pointer to the Virtual Interrupt table.
B2-B3  CC310 Keybd IRQ vector
       Vector to keyboard scan sub... used by Clock.

C0-C1  D.SysSvc  System Service Routine entry
C2-C3  D.SysDis  System Service Dispatch Table
C4-C5  D.SysIRQ  Sys State IRQ Routine entry
C6-C7  D.UsrSvc  User Service Routine entry
C8-C9  D.UsrDis  User Service Dispatch Table
CA-CB  D.UsrIRQ  User State IRQ Routine entry
CC-CF  D.SysStk  System stack
CE-CF  D.SvcIRQ  In-System IRQ service
D0  D.SysTsk  System Task number

E0-E1  D.Clock  Secondary Vectors:
E2-E3  D.XSWI3
E4-E5  D.XSWI2
E6-E7  D.XFIRQ
E8-E9  D.XIRQ
EA-EB  D.XSWI
EC-ED  D.XNMI

F2-F3  D.SWI3  Primary Interrupt Vectors:
F4-F5  D.SWI2  (most point to their D.X form above)
F6-F7  D.PIRQ
F8-F9  D.IRQ
FA-FB  D.SWI
FC-FD  D.NMI

OTHER SYSTEM RAM USAGE
(from above pointers- for info only)
0100-011F  D.Tasks  Task table
0120-015F  00A1-A2  Virtual dat tasks ptr
0200-023F  D.BlkMap  Block usage map
             ($80=notram, $01=in use, +$02=module)
0300-03FF  D.SysDis  Sys call dispatch table
0400-04FF  D.UsrDis User call dispatch table
0500-05FF  D.PrcDBT Proc Desc ptrs table
0600-07FF  D.SysPrc  System proc desc
0800-08FF  D.SysStk  (0900) system stack space
0900-09FF  D.SysMem  System page ram map ($01=in use)
0A00-0FFF  D.ModDir  Module DATImages

1000-1FFF  Global cc310 mem, alarm & system use
INSIDE OS9 LEVEL II
The System
Section 1

SAMPLE SYSTEM LOW MEMORY DUMP (000000-00FF)

0 1 2 3 4 5 6 7 8 9 A B C D E F

System Direct Page Variables

0000 A010000000000000 0000000000000000
0010 0000000000000000 00000000000000FF
0020 010000008FAE967E 0000000006233904
0030 0601000000083000 69E3000000000000
0040 02000240A0100000 0500060006400900
0050 0D00760000078000 OB80E86000000000
0060 0000000000000000 00000077FF917EED
0070 55550074127FFDFE 7EED5F0000000000
0080 810825F00000000 8000000000000000
0090 6C00080009000000 0315000000F80000
00A0 0101201000100020 00FE69FE7DE9D500
00B0 82E6E98400000000 0000000000000000
00C0 F3160300FE12F27E 0400FD370900E9D5
00D0 0000000000000000 0000000000000000
00E0 FCDF2F274F316F000 FE12F287F0000000
00FO 0000F271F271F271 E971F271A9B0000

0 1 2 3 4 5 6 7 8 9 A B C D E F

Task Numbers Use Table

0100 0101010000000000 0000000000000000
0110 0000000000000000 0000000000000000
0120 064011876D406D40 0000000000000000
0130 0000000000000000 0000000000000000
0140 0000000000000000 0000000000000000
0150 0000000000000000 0000000000000000
0160 0000000000000000 0000000000000000
0170 0000000000000000 0000000000000000
0180 0000000000000000 0000000000000000
0190 0000000000000000 0000000000000000
01A0 0000000000000000 0000000000000000
01B0 0000000000000000 0000000000000000
01C0 0000000000000000 0000000000000000
01D0 0000000000000000 0000000000000000
01E0 0000000000000000 0000000000000000
01F0 0000000000000000 0000000000000000

0 1 2 3 4 5 6 7 8 9 A B C D E F

Block Map (64 bytes)

0200 01010101010103D0 0303010000000000
0210 0000000000000000 0000000000000000
0220 0000000000000000 0000000000000000
0230 0000000000000000 0000000000000000
0240 0000000000000000 0000000000000000
0250 0000000000000000 0000000000000000
0260 0000000000000000 0000000000000000
0270 0000000000000000 0000000000000000
0280 0000000000000000 0000000000000000
0290 0000000000000000 0000000000000000
02A0 0000000000000000 0000000000000000
02B0 0000000000000000 0000000000000000
02C0 0000000000000000 0000000000000000
02D0 0000000000000000 0000000000000000
02E0 0000000000000000 0000000000000000
02FO 0000000000000000 0000000000000000

"Mfree" would check this map using F$GB1kMp call.

2-1-5
INSIDE OS9 LEVEL II

The System

Section 1

0 1 2 3 4 5 6 7 8 9 A B C D E F

System Dispatch Table

(WI2)

0300 F39397278A39852F 863486AE88484DF
0310 8948986F8A040000 8AC18AA58ACD98FD
0320 F72EF7C38BF8B24 8B98EADB8AE8F636
0330 8CA4A8C638C7E8CA8 8CA08D03EAA40000
0340 000096BF96A0000 0000000000000EA0
0350 F820F89795FC945 FD0FD86F4548D50
0360 8D738DF4F3689062 F386F84F8208E24
0370 FB23F967F9BA8E46 FA86FA3F8A25FC56
0380 FC66FC77FCA1FCC1 FA60000FB4BD0000
0390 FAF6FB12FB1C85DE 9530F38BF679D74
03A0 8BEE8EEB8F13F99C 0000000000000000
03B0 0000000000000000 0000000000000000
03C0 0000000000000000 0000000000000000
03D0 0000000000000000 0000000000000000
03E0 0000000000000000 0000000000000000
03F0 0000000000000000 0000000000000090DE

(I$call vector)

User Dispatch Table

(WI2)

0400 F39396FA8439852F 863486AE88484DF
0410 8940896F8A040000 8AC18AA58ACD98FD
0420 F72EF7B88EE28B17 8B8BEAD88AE8F636
0430 8CA4A8C638C7E8CA8 8CA08D03EAA40000
0440 000096BF96A0000 0000000000000EA0
0450 0000000000000000 0000000000000000
0460 000000000F368000 0000000000000000
0470 0000F5967000000 0000000000000000
0480 0000000000000000 0000000000000000
0490 0000000000000000 000000000000008E74
04A0 8EAE8EEB8000000 0000000000000000
04B0 0000000000000000 0000000000000000
04C0 0000000000000000 0000000000000000
04D0 0000000000000000 0000000000000000
04E0 0000000000000000 0000000000000000
04F0 0000000000000000 0000000000000090D9

(I$call vector)

Process Descriptors

Base Table (Pr0DBT)

Here: 0600 - n/a
0600 - id 1
7800 - id 2
7600 - id 3
6D00 - id 4
...
0 1 2 3 4 5 6 7 8 9 A B C D E F

0600 0100000200000000 0000FFFA00000000 The System (id 1)
0610 0000000000000000 0000000000000000 Process Descriptor
0620 8100000000000000 0000007500000000
0630 0101010000000000 0000000000000000
0640 0000333E333E0004 00100020003003F DAT Images
0650 0000000000000000 0000000000000000
0660 0000000000000000 0000000000000000
0670 0000000000000000 0000000000000000
0680 0000000000000000 005200000000001
0690 0000000000000000 0000000000000000
06A0 0000000000000000 0000000000000000
06B0 0000000000000000 0000000000000000
06C0 0000000000000000 0000000000000000
06D0 0000000000000000 0000000000000000
06E0 0000000000000000 0000000000000000
06F0 0000000000000000 0000000000000000

0 1 2 3 4 5 6 7 8 9 A B C D E F

0700 0000000000000000 0000000000000000 - and it's stack area
07F0 003F004000410042 0043004400450046

0 1 2 3 4 5 6 7 8 9 A B C D E F

0800 0000000000000000 0000000000000000 System Stack Page
08F0 10FEBEF400026D00 FD026D0012E7FE52

0 1 2 3 4 5 6 7 8 9 A B C D E F

0900 0101010101010101 0101010101010101 System 64K Page Map
0910 0101010101010101 0101010101010101 Each byte = one page
0920 0000000000000000 0000000000000000 01 = in use
0930 0000000000000000 0000000000000000 00 = free
0940 0000000000000000 0000000000000000 80 = not ram
0950 0000000000000000 0000000000000000
0960 0000000000000000 0000000000000000
0970 0101010101010101 0101010101010101
0980 0101010101010101 0101010101010101
0990 0101010101010101 0101010101010101
09A0 0101010101010101 0101010101010101
09B0 0101010101010101 0101010101010101
09C0 0101010101010101 0101010101010101
09D0 0101010101010101 0101010101010101
09E0 0101010101010101 0101010101010101
09F0 0101010101010101 0101010101010101 (top page is I/O)

2-1-7
0 1 2 3 4 5 6 7 8 9 A B C D E F
0A00 0FF41ED90D060000 0FF41ED90E300001 Module Directory
0A10 0FF41ED910000000 0EF6CE303000001
0A20 0EF6CE30FA0001 0EF6CE30FDC0001 Each entry is 8 bytes
0A30 0EF6CE319CF0014 0EF6CE32BFD0014 and contains:
0A40 0EF6CE330510008 0EF6CE33081000C DAT Image Ptr - 2
0A50 0EF6CE330B10000 0EF6CE330E10004 Block Size - 2
0A60 0EF6CE336C40004 0EF6CE342FA0001 Offset to Mod - 2
0A70 0EF6CE34FDF0001 0EF6CE35D1C0002 Link Count - 2
0A80 0EF6CE35D610000 0EF6CE35DaA30000 "Mdir" gets this table
0A90 0EF6CE35DB60000 0EF6CE35E290000 using F$GModDr call.
0AA0 0EF6CE35E6C0000 0EF6CE35EAF0000 Towards the end
0AB0 0EF6CE35EF20000 0EF6CE35F350002 and towards the end
0E80 0000000000000009 0000000800000000 is the temporary
0E90 0000000000000009 0000000000000000 DATImage stack.
0EA0 0000000000000006 0000000000000000
0EB0 0000000000000000 0000000000000000
0EC0 0000000000000000 0000000000000000
0ED0 0000000000000000 0000000000000000
0EE0 0000000000000000 0000000000000000
0EF0 000000000000001 00020003003F0000
0FF0 0000000000003F0000 0000000000000000 - end system vars.
1000 ----------------- ----------------- Begin CC3 global mem

2-1-8
OS9 SYSTEM CALLS

The OS9 system service calls, a SWI2 opcode followed by the call number, are the only recommended means to utilize memory, I/O and program control. A process inherits the SWI vectors from its parent, but may change them by the F$SSWI call.

Most of the calls are handled by the OS9 or OS9P2 modules. Any I/O call is vectored to IOMAN, which does its own internal table look-up. Another exception is the get-time call, which is dealt with by the Clock module.

There are two tables that contain the call vectors. The first table is from $00300-003FF, and is the table for calls made while in the system state. The user call table is at $00400-004FF.

To be in the system state, a program must currently be executing code within a system, manager, or driver module. This mainly occurs because of a system call. In other words, once a SWI call is made, all calls made within that call are vectored by the system table.

There are three main reasons for having a system mode. First, if a program is aborted while doing I/O (system mode), the program must be allowed to release I/O resources for other programs to use. Second, path numbers used while in the system mode are the actual path desc block number, and so must be distinguished from a process's path table pointer. And third, since new SWI and IRQ vectors are set on entry to the system mode, time is saved by bypassing this set-system-mode sub.

When a SWI2 call is made, the registers are placed on the current process's stack, and the stack pointer is saved in the process descriptor for easy access by the system modules. This way, the modules can use all the registers (except the SP) with impunity, and they all know where to get parameters passed and where to return values. Each module may do a fair amount of SWI2 calls itself. Under Level One, that meant that you needed to keep a large stack area for your program. That's not so important under Level Two, as the system or process descriptor stack is used mostly instead.

The calls from $28-$33 are regarded as privileged calls, since they have resource allocation powers that would be dangerous if used by a passing (non-system) program module. They may only be used while in the system state.

<table>
<thead>
<tr>
<th>SWI2 SERVICE REQUEST</th>
<th>OS9</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER SWI2</td>
<td>SYS SWI2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>State=sys</td>
<td>DP = 0</td>
</tr>
<tr>
<td>DP = 0</td>
<td>U = SP</td>
</tr>
<tr>
<td>U=SP, store P$SP</td>
<td>Table=sys (D.SysDis)</td>
</tr>
<tr>
<td>Table=user (D.UsrDis)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BSR Docrall</td>
<td>BSR Docrall</td>
</tr>
<tr>
<td>State=user</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>END</td>
<td>END</td>
</tr>
</tbody>
</table>
INSIDE OS9 LEVEL II
The System
Section 2

Docall Subroutine OS9

Get PC off IRQ stack
Get next byte (call)
Inc stack PC past call byte
(I/O call >=$80 ?) n------------->
ly 1
Vector at table-2 (I/O) (Call >= $37?) y------>
ln 1
Get call vector 1
(vector=0?) y------>1
<---------------------1
1
JSR the call vector 'Illegal SVC'

<-n (C set for err?)
ly 1
Return Reg.B=err code in B
<-------->1
1
Return lower 4 bits of CC

END SUB

I/O Vector I/O SERVICE CALL IOMAN

----------------------------------------
USER SYS
1 1
Table=CBC8 Table=CBEA
<---------------------1
1
(call>$90?) y---------------->'Illegal SVC'
ln
Get call vector
JMP to vector (Hidden RTS to OS9 Docall above)
SWI 01  F$LOAD  IOMAN

Alloc temp proc desc
Totram=0, Totmod=0
Set proc prty=caller's
Open EXEC. path to file
FSAllTsk, D.Proc=temp

Call ReadMod header
1
1 (M$ID 87CD okay?) n------------->
1  ly 1
1 Call ReadMod rest
1 1
1 FSReadMod into moddir err----------1
1
1 (known module?) n-->update
1  ly TotMod
1
1<------------------1
1
1 Set FoundMod flag
1<--------1
1

D.Proc=caller proc
Close EXEC. path
Check TotRam-TotMod
Release blocks unused
Dealoc temp proc desc
(FoundMod flag set?) n-->return err
1
1 Return ptr to first module
1 END

Subroutine ReadMod  IOMAN

sub
1
1 ModSiz=ModSiz+request
1<--n (ModSiz >TotRam?)
1  ly
1 Calc # of blocks needed
1 Find free blocks and set=$01
1 Set into temp proc desc dating
1 TotRam=TotRam+new blocks
1 F$SetTsk: update dating
1 1
1<--->1
1
1 Read in header/module
1
1 RTS

2-2-3
INSIDE OS9 LEVEL II
The System
Section 2

Verify Module

Call CRC check
F$FModul in ModDir
"n (find same name?)
\l
\l (revision higher on new?) n---> E$KwMdl
\l \ly
\l--->1

Set ModImg
MPDAT,MPtr,MDLink=0
MBSiz=up to and including module
\l ."n (module in another block?)
\l \ly
\l Free other entry
\l--->1

Mark BlkMap with "ModBlock"
\l
END

SWI 02          F$UNLINK
OS9P2

Calc proc desc dating block #
(does BlkMap show module?) n--->okay end
\ly
Decrement P$Link cnt
Search ModDir
"--------1
\l
\l ."---------
\l Next ModDir entry
\l \l \l
\l--->1 \l \l
\l (same MD$MPtr?) n--->1
\ly \l
\l (same block #?) n--->1
\ly
MD$Link cnt-1
"n (link cnt=0?)
\l \ly
\l Do IODEL if needed
\l Call ClearDir sub
\l--->1

Decrement P$Link cnt
"n (link cnt=0?)
\l \ly
\l Mark P$Datimg blocks as free
\l--->1
END
INSIDE OS9 LEVEL II
The System
Section 2

Subroutine ClearDir OS9P2

sub
1
Get dir entry block #
Check BlkMap flag "end if already clear
Pt to ModDir
  1<---------------------,
.<=n (blk=this entry?)
  1
  ly
  1
End if MD$Link<>0
  1
------->1
  1
Next ModDir entry 1
(last entry?) n------>1
  ly
Free BlkMap flags
Clear DatImg
Clear ModDir entry
  1
RTS

SWI 03 FSFORK OS9P2

FSAllProc desc
Copy parent's FSUser,Prior,DIO
ISDup std 0,1,2 paths
Call MakeProc
FSAllTsk for child
FSMove parameters to child map
FSMove register stack from proc desc to map
FSDelTsk of child
Return child id to caller
Set P$CID of parent, P$PID, P$SID of child
Clear SysState of child
FS$AP roc: activate child
  1
END

Subroutine MakeProc OS9P2

sub
1
FS$Link to module -ok--->.
  1
lok
  1
FS$Load module
  1
  1<-------------------------------1
  1
(Prgrm/Systm+Objct?) n------> err
  ly
Set P$PModul
FS$Mem for new D.Proc
Set new register stack in proc desc
  1
rts

2-2-5
Check D.PrcDBT table for free entry
F$SrgMem 512 byte proc descr
Set D.PrcDBT entry
  1
Set P$ID in proc descr
Clear P$DATImg
State = SysState
  1
END

Quick End if has P$Task
Call RestTk
Call SetTk
  1
END

Point to D.Tasks table
Skip first two (reserved for systm)
Find free entry, mark it used
Return entry number as task
  1
END

Point to D.Tasks table
Clear task entry
  unless is SysTk
  1
END

Clear ImgChg flag in P$State
Get P$Task
Copy P$DATImg's to task map
  1
END

subroutine Check Task
P$State has ImgChg flag set? n-->rts
  1y
Call SetTk
  1
  rts
(children?) n----------> 'No Children' error
    1
        (any dead yet?) y----------------->.
            1
                Return Regs.A=0
                Stop IRQ's
                Place proc at front of W.Queue
                Make a fake RTI stack
                F$Nproc:start next process
            1
                    ...
    1
        <F$Exit of child wakes parent>
        <Regs.D has child ID/code>
            1
                Get real SP
                1
                END

(dest ID=0?) y------>Send signal to all!
    1
        Send to ID only
            1
        END

    1
    <-->
    1
        Stop IRQ's
            1
            1
                (code=abort?)
            1
                Make proc condemned state
                    1
                        (has signal?) y-------.
                            1
                            (signal=wake?) n------>error
                                1
                Store signal
                Wake up proc
                Signal=0 if signal=1
                Insert proc in A.Queue
            1
        END of SUB
INSIDE OS9 LEVEL II
The System
Section 2

=====================================================================================================================

SWI 06    FS$EXIT    OS9P2
=====================================================================================================================

FS$Signal = Regs.B
Close all I/O paths
Return data memory
Unlink primary module
1
Point to our last child
1
1
1
1
1
1 Return proc desc's of all
1 1 live proc desc 1
dead (FS$Exit'd) children.
1 1 Dealloc proc desc 1
1 1----->1 1
1 Zero parent ID 1 Live kids are now orphans.
1 Point to sibling 1
1 1 1
1 1
1 1
1 1
1 1
1 (any children?) y-->1
1 in
1
1
1
1
1 (we have parent?) y----->. If we are orphan ourselves
1 in 1 we exit quickly.
1 Dealloc our proc desc 1
1
1
1
1
1
1
1
1
1
1
1
1
1
1 If parent hasn't FS$Waited,
1 1 we are marked as Dead for
1 1 parent's Wait or Exit.
1 1
1 1
1 1
1
1
1
1
1
1
1
1 .<----------n (parent waiting?)
1 1 ly
1 Mark us as Take parent out of W.Queue
1 Dead FS$Activate parent
1 1 Put ID/code in parent's Regs.D
1 1 Fix sibling links
1 1 Dealloc child proc desc
1 1 1
1 1
1 1
1 1
1 1
1
1 .<-----------------n
1
1
1
1
1
1
1
1
1
1
1 D.Proc = 0000
1 END

2-2-8
SWI 00  F$LINK  OS9P1

Type=Reg.A
Name ptr=Reg.X
Find module dir entry -err------>$DD error
  l
  .<--y (reentrant?)
  l   ln
  l (link cnt=0?) n-------->$DL error
  l   ly
l------->l
  l
Inc link cnt
Return type/lang/hdr/entry
  l
END

SWI 0C  F$ID  OS9P2

Get ID from Proc Desc
Get User from Proc Desc
  l
Return ID in Reg.A
Return User in Reg.Y
  l
END

SWI 0D  F$SPRIOR  OS9P2

ID# = Reg.A
Find Proc Desc for ID -err------>'Not Found'
  l
(same index?) n------------->'Not Yours'
  ly
New proc priority=Reg.B
  l
END

SWI 0E  F$SWI  OS9P2

Point to Proc Desc's SWI table
Type= Reg.A
(type>3?) y-------------->'Illegal SWI Code'
  ln
New vector=Reg.X
  l
END

SWI 0F  F$SPERR  IOMAN

Get Error Path (#2) from Proc Desc table
Convert Reg.B code to ASCII number
Print 'ERROR #'
Print err number
  l
END

2-2-9
SWI 15 F$TIME CLOCK

Destination=Reg.X
F$Move D.Time to dest
1
END

SWI 16 F$SETIME OS9P1

Source=Reg.X
Move source to D.Time
F$Link to 'Clock'
1
(error?) y----------->'Unknown Module'
ln
Jmp to Clock init (after this, Clock usually sets it's
1 own F$Setime call - see below)
(END)

System Module Init CLOCK

Set constants vars
Insert Clock vector at D.IRQ
F$SSVC new Time call
1
END
SWI 2A F$IRQ IOMAN
---------------------------------------------------------------------

Get packet values
Get max # IRQ entries from INIT
Point to poll table (<$62)

.<--y (Reg.X=O?)
  ln
  ln
  (mask=0?) y------->error
  ln
  Search for empty
    1
    (no empties?) y------>'Poll Table Full'
    1
  Sort by priority
  Insert new entry
  1
  END

1------>. * KILL ENTRY *

  Find entry by data address At INIT Module=50C
  Delete it is max # entries.
  Move rest up in table
  1
  END

---------------------------------------------------------------------

POLLING TABLE ENTRY FORM:

00-01 Address of status port
02 Flip byte
03 Mask byte
04-05 IRQ service address
06-07 Storage memory address
08 Priority (0-low,255-high)

---------------------------------------------------------------------

System Module IRQ Polling Routine IOMAN
---------------------------------------------------------------------

Point to polling table
Get max # entries from INIT

1---------------->.  1

.<-->.  1
  Point to next entry  1
  (end of table?) y------>  1---------------->.
    ln  1
    ln  1
    1------------------- 1

    1 Get status byte 'Table Full Err'
    1 Flip and Mask Return error
    1<--n (found it?)
      1
    1 Do service routine
    1<--y (error?)
      ln
    END

---------------------------------------------------------------------

2-3-1
FSlink--device desc
  (get header, device address) err-------------->
FSlink--device driver
  (get driver entry address) err------------------->
FSlink--file manager
  (get mgr entry address) err---------------------->
    1
    Get max # of entries (INIT)
Get device table add (<$60) 1

  --> (entry empty?) y---------------------->
  1 (same desc?) n----------------------> 1
  1 (mem alloc'd?) y----------------------> 1 1
  1
  1
  k<n (any user?)  y 1 1
  1 1
  1 Insert in I/O queue 1 1
  1
  k<----wakeup 1 1 1
  1
  1
  1
  1
  1
  1
  1 Save entry ptr 1 1
  1
  1
  1
  1
  1
    (same port add?) n---------------------->
    1
    (same driver?) n---------------------->
    1
    (user cnt=0?) y---------------------->
    1
    1
    1 Save user cnt 1
    1
    k<--------------------->
    1
    Point to next entry
    k<n (last?)
    1
    (entry found?) y---------------------->
    1
Find empty spot
  (error?) y-->'Table Full' 1
    1
  <--> (mem alloc'd?) 1
    1
  1
  1 Allocate drvr mem & clear
  1
  1 Set V.Port add in mem
  1
  1 Do driver init sub
  1
  1
  1 Insert device tbl data
  1
  k<--------------------->
  1
  (Check desc/drvr modes) err--------> 'Illegal Mode' 1
  1
  <--> (user cnt=0?) 1
  1 (device shareable?) n----------------------> 'Device Busy' 1
  1
  1 Increment user cnt
  1
Return table entry in Regs.U

END

---

2-3-2
INSIDE OS9 LEVEL II
The System
Section 3

SWI 83
IS$DUP
IOMAN
-----------------------------------------------
Get free path # from Proc Desc  err-->'Path Table Full'
Find path desc of old path      err-->'Unknown Path'
Increment path desc image cnt
Return new Proc path ptr in Regs.B

SWI 83/84
IS$CREATE/OPEN
IOMAN
-----------------------------------------------
Get free path # from Proc Desc
1
Get requested mode
Allocate path desc
Do File Manager Create/Open
1
Put path desc # in Proc path table
Return Proc path number in Regs.A
1
END

SWI 8F
IS$CLOSE
IOMAN
-----------------------------------------------
Get Proc path ptr for A=path#  
Zero that path ptr in Proc Desc
Find path desc
Decrement # of open images
1
<y (current proc ID?)
1
1n
1 Update I/O queue
1 Save caller's stack in PD.REGS
1 Do File Manager Close
1 1
1 Wake up proc's in pd.links
1 1
1 (proc.ID=path.ID?) n-->.
1 1
ly 1
1 Clear path.ID
1<-------1
1
(open images=0?) n-->.
1
Ly 1
I$DETACH device 1
Kill path desc 1
1 1
<y<-----------1
END

2-3-3
INSIDE OS9 LEVEL II
The System
Section 3

SWI 86
I$CHGDIR
IOMAN

Save SWI code for later use
Allocate temp path desc
 1
Do File Manager Chgdir sub (RB$man finds dir desc LSN &
 1
dr$ and puts in Proc Desc)
,<-1---------->,
 1 1
data dir       exec dir
 1 1
(dec user cnt in device table for old dir's device)
(inc user cnt in device table for new dir's device)
(set new device table entry into Proc Desc)
 1
Point to device table entry for this temp path
I$Detach drive
F$Dealloc64 - kill this temp path desc
 1 END

PROCESS DESCRIPTOR DEFAULT DIR ENTRIES:

<table>
<thead>
<tr>
<th>data</th>
<th>exec</th>
<th>from</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-21</td>
<td>25-26</td>
<td>Device table entry ptr (IOMAN)</td>
</tr>
<tr>
<td>22</td>
<td>27</td>
<td>Drive number (not used) (RB$man)</td>
</tr>
<tr>
<td>23-24</td>
<td>28-29</td>
<td>Dir file desc LSN (RB$man)</td>
</tr>
</tbody>
</table>

SWI 89
I$READ
IOMAN

Find path desc
 1
  (read attr?) n----------> 'No Permission'
  1
  ly
,<-n (path desc in use?) 1
  1
  ly
  1
Place in I/O Queue 1
  1
  wakeup ------------->1
  1
  
  
  1 Do File Manager Read sub
Wake up others in I/O Queue
Clear path user if still us (PD.CPR)
  1
END

2-3-4
Subroutine IOMAN

ALLOCATE PATH DESCRIPTOR (Open, Create)

Get pd's base (D.PthDBT)
Allocate 64 byte block
Set user cnt=1, mode=mode requested

Point to pathname
Skip blanks

.==y (1st char='/'?)
   If '/', it's full pathname;
   Else use default dirs for this
   process descriptor.

dir type?

-------------

data dir     exec dir

(get device tbl entry from Proc Desc)

(entry=0?) y-------------------------->

Point to device desc name

--------------

Parse name of device

(error?) y-------------------------->

Attach device

Save table ptr in path desc

(attach err?) y-------------------------->

Get device desc init size 'Bad Pathname'

Move up to 32 bytes to path desc

Deallocation pd block

END Error End
**INSIDE OS9 LEVEL II**

*The System*  
*Section 4*

---

**IRQ HANDLING**

I have included this general text for the hackers out there.

Technical notes on the flow of hardware interrupt handling in OS9 L-I CoCo ver 1.X or 2.0, and OS9 L-II Gimix ver 2.0 or CoCo 1.X.

The 6809 has three hardware interrupt lines, NMI, FIRQ, and IRQ. This doc concentrates on the IRQ, which is the one used by OS9 for it's clock and I/O device polling routines.

I'll cover the various paths OS9 may take when it receives an IRQ, which don't the current level, revision & system state. Note that because I only touch on IRQ-related code, other variables are involved.

**IRQ'S - CLOCKS and DEVICES**

There are two main source categories of IRQ's: clock and device. They're both vectored to the same handler at their start, but branch differently. (CoCo OS-9 adds the VIRQ and FIRQ, but they end up being treated as an IRQ.)

The timesharing type has to do with updating the D.Time variables and calling the kernel's D.Clock process-switching algorithm. It comes from a regular timed interrupt source, such as the 60Hz Vertical Sync on the CoCo, or a clock chip or timer on other systems.

The other type is from a device asking for service. Usually that device's driver has entered an F$IRQ request, so that the OS will know where to vector, after the polling routine has found that IRQ source device.

**BASIC INTERRUPT HANDLING**

All 6809 machines fetch their cpu interrupt vectors from a ROM that can be read at logical addresses FFFX. The IRQ vector is at FFF8-F9.

**Level-I CoCo 1/2**

The ROM in these computers vectored to 010C, which contains a BRA to 0121, which does a JMP [D.IRQ].

**Level-I Coco 3**

The new ROM vectors IRQs to FEF7, where it does a LBRA to 010C, maintaining compatibility with 1.X or 2.0 OS-9. See CoCo 1/2 above. L-II of course needs the FEXX page pseudo-vectors so that there is always IRQ handling code across all task maps.
Level-II Task Switching

In Level-II, interrupts are ROM-vectored to the code at the top of OS9p1. This code lies within the page that is mapped across all task maps (on some systems, an interrupt causes a hardware reset of the task register to the system map instead, so a user has the full 64K available). In either case, the task register is set to the SysTask, the Direct Page register is set to zero, and then JMP [D.IRQ] D.IRQ defaults to the IntXfr (interrupt transfer) code in OS9p2, which does what boils down to a JMP [D.XIRQ]. This is changed by the Clock module.

OS-9 VECTOR INITIALIZATION

When OS9 first cranks up, it sets the following:

D.UsrIRQ - kernal user-irq routine
D.SysIRQ - " system "
            both of which will and up JSR'ing [D.Poll]
D.SvcIRQ - has D.SysIRQ in it
D.IRQ - kernal JMP [D.SvcIRQ]
D.Poll - kernal COMB, RTS

This means that initially all IRQ's go thru the kernal to [D.SvcIRQ] back to the kernal's own Sys/UsrIRQ code, which then calls [D.Poll] to find the source. As the kernal does not do polling, and IOMAn isn't initialized yet, D.Poll returns an error. The Sys/UsrIRQ code then shuts off IRQ's by setting the CC bits as a precaution.

TRANSFER TO SYSTEM STATE - Level-I or II

Whether a program is in the user or system state when an interrupt occurs affects what D.SvcIRQ contains.

If in user state, it contains the vector constant copied from D.UsrIRQ. The routine in OS9p1 at that address saves the task's SP, sets SWI vectors to use system vectors, and copies D.SysIRQ into D.SvcIRQ.

The OS9p1 routine at [D.SysIRQ] does not save or set up anything as you are already in the system state. This helps speed interrupt handling.

IOMAN INIT

When the first I$Call is made, the kernal links to and initializes IOMAN (I/O MANager). Ioman inserts a vector to itself in D.Poll. From then on, IRQ's still go thru the kernal [D.SvcIRQ] to the Sys/UsrIRQ code, but their call to [D.Poll] is now honored by ioman, which does the source searching (polling).

Also on the init call, ioman sets up several tables. These are the device table [D.DevTb1], polling table [D.PollTb1], and on the CoCo the VIRQ (virtual irq) table [D.CltTab].

These tables will be used by ioman for keeping track of active devices, inserting and deleting FSIRQ entries, and by ioman's D.Poll routine in finding the source of an IRQ.
CLOCK INIT and OPERATION

We must include Clock modules here because they are important in the IRQ heirarchy. A side note: some clock modules keep their device address in the M$Size (data size) portion of their module header.

Clock modules keep track of the real time. Interrupts usually are vectored almost directly to them, and they decide for themselves if a clock IRQ was involved. In effect, a special device driver IRQ routine.

They are not in a polling table because a) the clock must be serviced quickly, and b) they may jump directly or thru another module to the kernal's timesharing routine (D.Clock) and so cannot be called as a subroutine such as device IRQ handlers are.

When the first F$STime call is made (best from SysGo), OS9p1 links to any module called "Clock", and JSR's to it's entry point. There the Clock module inserts itself into the system D.IQR vector, so that it gets called first.

After that, IRQ's come to Clock, who checks to see if it's timer was the source. If so, it updates the time variables as needed, and jumps via D.Clock to the kernal (L-II jumps via D.XIRQ to the kernal).

If the timer or clock chip was NOT the IRQ source, then Clock jumps [D.SvcIRQ] so that OS9 can check for the correct device.

Exception #1: on the CoCo L-1 ver 1.X, the IRQ's go first to CClO (so it could time the disk motors), then to Clock via [D.AltIRQ], then Clock continued by [D.Clock].

Exception #2: on the CoCo L-1 ver 2.0, Clock jumps via [D.AltIRQ] to the CCIO keyboard scan. CCIO finishes the jump to [D.Clock].

IOMAN IRQ POLL SYNOPTIC

As we know now, when the Clock's D.IQR code finds that an IRQ has occurred from other than it's IRQ, the IOMan D.Poll vector is eventually called.

IOMan looks thru the Polling Table, which has been presorted by device priority. Each Q$POL address is read, XOR'd with the Q$FLIP byte, AND'd with the Q$MASK byte, and if is not-$00 after all that, the Q$SERV routine in the driver for that device is called to service and clear that IRQ.

If the driver service code finds that a mistake has been made in it's selection, it can set the C bit, and IOMan will continue the search thru the table. See D.SvcIRQ above.
INSIDE OS9 LEVEL II
The System
Section 4

IRQ FLOWCHARTS

CoCo Level I

IRQ

1

ROM: jmp [D.IRQ]  
(was it clockirq?) y--------->update time

nl

jmp [D.SvcIRQ]  

jmp [D.SvcIRQ]

(D.SvcIRQ)----- or ------(D.SysIRQ)

l

1

jmp [D.Clock]

D.SvcIRQ = D.SysIRQ  
jsr [D.Poll]  
update ticks

jsr [D.Poll]

l<-------------------y (ticks>0 or SysState?)

rti

nl

D.SvcIRQ = D.SysIRQ

rti

choose next proc

D.SvcIRQ = D.SysIRQ

D.Poll

scan devices, do driver irq sub

rts

Level II

IRQ

1

ROM: jmp to allmap page (XPEXX)

TaskReq = SysTask

jmp [D.IRQ]:  
(was it clockirq?) y--------->update time - - - - - - - - - - - - - - -

nl

D.SvcIRQ = D.Poll

D.SvcIRQ = D.Clock

D.SvcIRQ = D.Virq

jmp [D.XIRQ]:

(D.SysIRQ)----- or ------(D.SysIRQ)

l

SP = D.SysStk

jsr [D.SvcIRQ]

jsr [D.SvcIRQ]

l

(slice up?) n--------

ly

choose proc to run

l

D.XIRQ = D.SysIRQ

switch task to user

rti

D.XIRQ = D.SysIRQ

D.Poll:

find source, driver IRQ sub

rts

D.Clock:

update ticks

rts

D.Virq:

update Virq table

call D.Poll if Virq

jsr [D.KbdIRQ] scan

check & do alarm sig

jmp [D.Clock]

rts
NOTES:

All code is OS9pl, except D IRQ/D Virq-->Clock, and D.Poll-->IOMan. In most cases, IRQ's (and FIRQ's) are not reenabled until the RTI.

The L-II D.Clock is a subroutine, but the L-I D.Clock both updates the ticks, and then falls through to the timeshare routine.

Notice that if an interrupt occurs while in , other processes get a chance to run if the current process is out of time.

GENERAL NOTES:

virqs end up as irqs

Just after the end of the OS9pl module are the offsets to the following default code within it:

D.Clock routine
D.SW13  (these are D.X... in Level-II)
D.SW12
D.FIRQ
D.IRX
D.SWI
D.NMI

IRQ-RELATED DP. VARS and SYSTEM TABLES

The following are the Direct Page ($00XX) variables that have to do with interrupt processing, and their addresses on the CoCo and GIMIX machines. Each contains a two-byte vector to the code within a System module that handles it, or point to a table.

Your system may vary, so check your OS9Def$ file, if you don't own one of those computers. Addresses are included simply to give a rock to cling to.

<table>
<thead>
<tr>
<th>NAME</th>
<th>L-I</th>
<th>L-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.Init</td>
<td>2A-2B</td>
<td>24-25</td>
</tr>
<tr>
<td>D.DevTbl</td>
<td>60-61</td>
<td>80-81</td>
</tr>
<tr>
<td>D.Poll1b</td>
<td>62-63</td>
<td>82-83</td>
</tr>
<tr>
<td>D.FIRQ</td>
<td>30-31</td>
<td>F6-F7</td>
</tr>
<tr>
<td>D.IRX</td>
<td>32-33</td>
<td>F8-F9</td>
</tr>
<tr>
<td>D.NMI</td>
<td>36-37</td>
<td>FC-FD</td>
</tr>
<tr>
<td>D.SvcIRQ</td>
<td>38-39</td>
<td>CE-CF</td>
</tr>
<tr>
<td>D.Poll</td>
<td>3A-3B</td>
<td>26-27</td>
</tr>
<tr>
<td>D.AltIRQ</td>
<td>6B-6C</td>
<td></td>
</tr>
<tr>
<td>D.Clock</td>
<td>81-82</td>
<td>E0-E1</td>
</tr>
<tr>
<td>D.C17b</td>
<td>86-87</td>
<td>B0-B1</td>
</tr>
<tr>
<td>D.KbdIRQ</td>
<td>B2-B3</td>
<td></td>
</tr>
<tr>
<td>D.XIRQ</td>
<td>E8-E9</td>
<td></td>
</tr>
</tbody>
</table>
Then there are the Direct Page variables that contain initialized vector constants, so that interrupts may be handled differently depending upon the OS state:

D.UsrIRQ 3C-3D CA-CB User state D.SvcIRQ vector
D.SysIRQ 3E-3F C4-C5 System state D.SvcIRQ vector

IOMAN TABLES --------------------------------------

The size of these tables is calculated from the DEVCNT and POLCNT entries in the system INIT module.

DEVICE TABLE ENTRIES
V$DRIV 00-01 Driver module address
V$STAT 02-03 Device static storage
V$DESC 04-05 Device Descriptor
V$PMGR 06-07 File Manager
V$USR 08 Device User Count
DevSiz equ .

POLLING TABLE ENTRIES
Q$POLL 00-01 Polling address (device status byte address)
Q$FLIP 02 Flip byte for negative logic IRQ bits
Q$MSK 03 Mask byte for IRQ status bit
Q$SERV 04-05 Driver IRQ service routine
Q$STAT 06-07 Device static memory pointer
Q$PRT 08 Device polling priority (position in table)
PolSiz equ .
INSIDE OS9 LEVEL II

Devices
## INSIDE OS9 LEVEL II
### Devices
#### Section 1

**OS9 I/O**

---

**RBFMAN FILE**

---

PD.-- path descriptor vars  
V$-- device table  
V.-- device static storage  
QS-- IRQ poll table  
DD.-- drive tables (LSN 0)  
P$-- process descriptor

---

Opening a disk (RB) file takes the following steps:

<table>
<thead>
<tr>
<th>#</th>
<th>VAR</th>
<th>MOD</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PD.PD</td>
<td>IOMAN</td>
<td>Allocates a 64-byte block path descriptor.</td>
</tr>
<tr>
<td></td>
<td>PD.MOD</td>
<td></td>
<td>Sets access mode desired.</td>
</tr>
<tr>
<td></td>
<td>PD.CNT</td>
<td></td>
<td>Sets user cnt=1 for this path desc.</td>
</tr>
<tr>
<td>2</td>
<td>PD.DEV</td>
<td>IOMAN</td>
<td>Attaches the device (drive) used.</td>
</tr>
<tr>
<td></td>
<td>V$STAT</td>
<td></td>
<td>Allocates memory for device driver (CCDisk).</td>
</tr>
<tr>
<td></td>
<td>V.PORT</td>
<td></td>
<td>Sets device address in driver static memory.</td>
</tr>
<tr>
<td>3</td>
<td>V.NDRV</td>
<td>CCDISK</td>
<td>The driver's init subroutine is called to</td>
</tr>
<tr>
<td></td>
<td>V.TRK</td>
<td></td>
<td>initialize the device, and static memory</td>
</tr>
<tr>
<td></td>
<td>DD.TOT</td>
<td></td>
<td>(drive tables) to default values.</td>
</tr>
<tr>
<td>4</td>
<td>Q$POLL</td>
<td>OS9</td>
<td>Sets up IRQ polling table entry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( address, flip &amp; mask bytes, service add,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>static storage, priority of IRQ )</td>
</tr>
<tr>
<td>5</td>
<td>V$DRIV</td>
<td>IOMAN</td>
<td>Sets up rest of device table.</td>
</tr>
<tr>
<td></td>
<td>V$DESC</td>
<td></td>
<td>( module addresses of desc, driver, mgr)</td>
</tr>
<tr>
<td></td>
<td>V$PMGR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>V$USR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sets user count of device=1</td>
</tr>
<tr>
<td>6</td>
<td>PD.OPT</td>
<td>IOMAN</td>
<td>Copies device desc info to path desc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( drive #, step rate, density, tracks,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sides, interleave, seg alloc size )</td>
</tr>
<tr>
<td>7</td>
<td>PD.BUF</td>
<td>RBFMN</td>
<td>Allocates buffer for file use.</td>
</tr>
<tr>
<td></td>
<td>PD.BVT</td>
<td></td>
<td>Copies device table entry for user.</td>
</tr>
<tr>
<td></td>
<td>PD.TAB</td>
<td></td>
<td>Calc's drive table add for quick ref'rence.</td>
</tr>
<tr>
<td></td>
<td>DD.TOT</td>
<td>CCDISK</td>
<td>Copies LSN 0 init info to drive table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( diskette's format, root dir, ID, attr's,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>number of tracks, sectors, bitmap size )</td>
</tr>
<tr>
<td></td>
<td>DD.RES</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PD.DSK</td>
<td>RBFMN</td>
<td>Gets disk ID and finds the file:</td>
</tr>
<tr>
<td></td>
<td>PD.DPD</td>
<td></td>
<td>LSN of directory file desc</td>
</tr>
<tr>
<td></td>
<td>PD.DCP</td>
<td></td>
<td>Entry # of pathname in directory file</td>
</tr>
<tr>
<td></td>
<td>PD.PD</td>
<td></td>
<td>LSN of pathname's file desc</td>
</tr>
<tr>
<td></td>
<td>PD.CP</td>
<td></td>
<td>Current file pos</td>
</tr>
<tr>
<td></td>
<td>PD.SIZE</td>
<td></td>
<td>File size</td>
</tr>
<tr>
<td></td>
<td>PD.SBL</td>
<td></td>
<td>Offset from beginning of file segment</td>
</tr>
<tr>
<td></td>
<td>PD.SBP</td>
<td></td>
<td>LSN of file segment</td>
</tr>
<tr>
<td></td>
<td>PD.Ssz</td>
<td></td>
<td>Segment size in sectors</td>
</tr>
<tr>
<td></td>
<td>PD.ATT</td>
<td></td>
<td>File attributes (DSEWR)</td>
</tr>
<tr>
<td>8</td>
<td>P$PATH</td>
<td>IOMAN</td>
<td>Puts path desc # in proc desc I/O table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns table pointer to user as path number.</td>
</tr>
</tbody>
</table>
DEVICE DRIVER ENTRIES

INIT

U = device static memory
Y = device descriptor
CC.B = error code

. Set V.NDRV to number drives controller handles
. Set DD.TOT to non-zero value so RBPMAN can read LSN 0
. Set V.TRK to high number if driver controls seek op code
* Use F$IR to place driver IRQ service routine in poll table
. Init controller
* Copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0

READ

U = device static memory
Y = path descriptor
CC.B = error code
B, X = LSN

. Get PD.BUF buffer address from path descriptor
. Get PD.DRV drive number from path descriptor
. Send LSN converted to track and sector to controller
* Copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0
. Read the data into the buffer if not a DMA controller
. If LSN 0, copy DD.SIZ bytes into drive table

WRITE

U = device static memory
Y = path descriptor
CC.B = error code
B, X = LSN

. Get PD.BUF buffer address from path descriptor
. Get PD.DRV drive number from path descriptor
. Send LSN converted to track and sector to controller
. Write the data into the buffer if not a DMA controller
* Copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0

GETSTT

U = device static memory
Y = path descriptor
A = status call
CC.B = error code

. Do wildcard driver call if not handled by IOMAN/RBPMAN

PUTSTT

TERM

U = device static memory
CC.B = error code

* Wait for any I/O to complete
* Disable any device IRQ's
* Remove device from IRQ polling table

* IRQ Service Routine

. Kill IRQ request if necessary
. Send wakeup signal to V.Wake
. Clear V.Wake
. Clear Carry bit
. RTS
* Interrupt driven devices only!

---

**DEVICE VARIABLES**

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V.PAGE</td>
<td>00</td>
<td>Port extended address</td>
</tr>
<tr>
<td>V.PORT</td>
<td>01-02</td>
<td>Device address</td>
</tr>
<tr>
<td>V.LPRC</td>
<td>03</td>
<td>Last active process ID (not used)</td>
</tr>
<tr>
<td>V.BUSY</td>
<td>04</td>
<td>Active process ID (dev busy flag) 0=not busy</td>
</tr>
<tr>
<td>V.WAKE</td>
<td>05</td>
<td>Process ID to awake after command completed</td>
</tr>
<tr>
<td>V.USER</td>
<td></td>
<td>Beginning of file mgr/driver var's</td>
</tr>
<tr>
<td>V.NDRV</td>
<td>06</td>
<td>Number drives controller can handle</td>
</tr>
<tr>
<td></td>
<td>07-0E</td>
<td>Reserved</td>
</tr>
<tr>
<td>DRVBEG</td>
<td>.</td>
<td>Beginning of drive tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(One table for each drive, up to V.NDRV)</td>
</tr>
</tbody>
</table>

This section of each table copied from LSN 0 of disk. Dr#0

| DD.TOT | 00-02  | Number of sectors                               |
| DD.TKS | 03     | Number of tracks                                 |
| DD.MAP | 04-05  | Number bytes in allocation map                   |
| DD.BIT | 06-07  | Sectors/bit in map (sectors/cluster)             |
| DD.DIR | 08-0A  | LSN of root directory                            |
| DD.OWN | 0B-0C  | Owner's user number                              |
| DD.ATT | 0D     | Disk attr (D S PE PW PR E W R)                   |
| DD.DSK | 0F-0F  | Disk ID                                          |
| DD.FMT | 10     | Disk format                                      |
| DD.SPT | 11-12  | Sectors/track                                    |
| DD.RES | 13-14  | Reserved                                         |
| DD.SIZ | .      | Size of bytes to copy from LSN 0                 |
| V.TNAK | 15-16  | Current track                                    |
| V.BMB  | 17     | Bit map in use flag                              |
| V.FileHd| 18-19  | Open file list                                   |
| V.DiskID| 1A-1B  | Disk ID                                          |
| V.BMapsz| 1C     | Bitmap size in sectors                           |
| V.MapSct| 1D     | Lowest reasonable bitmap sector                  |
| V.ResBit| 1E     | Reserved bit map sector                           |
|       | 1F-25  | Reserved                                         |

**DRVMEM**

<table>
<thead>
<tr>
<th></th>
<th>Drive table size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(other drive tables follow)</td>
</tr>
</tbody>
</table>

Drive table address = DRVBEG + ( PD.DRV * DRVMEM )
Also found in PD.DVTB
### Module Insight OS9 Level II

#### Devices

**Section 1**

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M$ID</td>
<td>00-01</td>
<td>Sync bytes ($87CD)</td>
</tr>
<tr>
<td>M$Size</td>
<td>02-03</td>
<td>Module size</td>
</tr>
<tr>
<td>M$Name</td>
<td>04-05</td>
<td>Offset from start to module name string</td>
</tr>
<tr>
<td>M$Type</td>
<td>06</td>
<td>Type/lang ($F1)</td>
</tr>
<tr>
<td>M$Revs</td>
<td>07</td>
<td>Attr/revision</td>
</tr>
<tr>
<td>M$Parity</td>
<td>08</td>
<td>Header parity</td>
</tr>
<tr>
<td>M$FMgr</td>
<td>09-0A</td>
<td>File manager name offset</td>
</tr>
<tr>
<td>M$Dev</td>
<td>0B-0C</td>
<td>Driver name offset</td>
</tr>
<tr>
<td>M$Mode</td>
<td>0D</td>
<td>Device capabilities</td>
</tr>
<tr>
<td>M$Port</td>
<td>0E-10</td>
<td>Device extended address</td>
</tr>
<tr>
<td>M$Opt</td>
<td>11</td>
<td>Number of options in initialization table</td>
</tr>
<tr>
<td>IT.DTP</td>
<td>12</td>
<td>Device type (1=RBF)</td>
</tr>
<tr>
<td>IT.DRV</td>
<td>13</td>
<td>Drive number (0...n)</td>
</tr>
<tr>
<td>IT.STP</td>
<td>14</td>
<td>Step rate: 0-30 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-20 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-12 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-6 ms</td>
</tr>
<tr>
<td>IT.TYP</td>
<td>15</td>
<td>Device type: bit0-0=5 1/4 1=8 inch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit5-0=noncoco 1=coco</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit6-0=os9std 1=nonstd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit7-0=floppy 1=hard</td>
</tr>
<tr>
<td>IT.DNS</td>
<td>16</td>
<td>Density: bit0-0=single 1=double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit1-0=48 tpi 1=96 tpi</td>
</tr>
<tr>
<td>IT.CYL</td>
<td>17-18</td>
<td>Cylinders (tracks)</td>
</tr>
<tr>
<td>IT.SID</td>
<td>19</td>
<td>Sides</td>
</tr>
<tr>
<td>IT.VFY</td>
<td>1A</td>
<td>0= verify disk writes</td>
</tr>
<tr>
<td>IT.SCT</td>
<td>1B-1C</td>
<td>Sectors/track</td>
</tr>
<tr>
<td>IT.TOS</td>
<td>1D-1E</td>
<td>Sectors/track (track 0)</td>
</tr>
<tr>
<td>IT.ILV</td>
<td>1F</td>
<td>Sector interleave</td>
</tr>
<tr>
<td>IT.SAS</td>
<td>20</td>
<td>Minimum #sectors/segment alloc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End of option table</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>Name strings here.</td>
</tr>
</tbody>
</table>

---

3-1-4
### INSIDE OS9 LEVEL II

**Devices**  
Section 1

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD.PD</td>
<td>00</td>
<td>Path number</td>
</tr>
<tr>
<td>PD.MOD</td>
<td>01</td>
<td>Access mode 1-read 2-write 3-update</td>
</tr>
<tr>
<td>PD.CNT</td>
<td>02</td>
<td>Number of paths using this path desc</td>
</tr>
<tr>
<td>PD.DEV</td>
<td>03-04</td>
<td>Device table entry address</td>
</tr>
<tr>
<td>PD.CPR</td>
<td>05</td>
<td>Current Proc ID using this path for I/O</td>
</tr>
<tr>
<td>PD.RGS</td>
<td>06-07</td>
<td>Address of user's register stack</td>
</tr>
<tr>
<td>PD.BUF</td>
<td>08-09</td>
<td>Data buffer (256 bytes) address for RBF/drvr</td>
</tr>
<tr>
<td>PD.SMF</td>
<td>0A</td>
<td>Buffer state: see below</td>
</tr>
<tr>
<td>PD.CP</td>
<td>0B-0E</td>
<td>Current file position</td>
</tr>
<tr>
<td>PD.SIZ</td>
<td>0F-12</td>
<td>File size</td>
</tr>
<tr>
<td>PD.SBL</td>
<td>13-15</td>
<td>File beginning segment number (FSN)</td>
</tr>
<tr>
<td>PD.SBP</td>
<td>16-18</td>
<td>Actual segment beginning LSN</td>
</tr>
<tr>
<td>PD.SSE</td>
<td>19-1B</td>
<td>Segment size in sectors</td>
</tr>
<tr>
<td>PD.DSK</td>
<td>1C-1D</td>
<td>Disk ID</td>
</tr>
<tr>
<td>PD.DTB</td>
<td>1E-1F</td>
<td>Drive table address for this drive</td>
</tr>
</tbody>
</table>

This section copied by IOMAN from the Device Descriptor:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD.OPT</td>
<td>20</td>
<td>Device class 0=SCF 1=RBF 2=PIPE</td>
</tr>
<tr>
<td>PD.DRV</td>
<td>21</td>
<td>Drive number 0-n</td>
</tr>
<tr>
<td>PD.STP</td>
<td>22</td>
<td>Step rate</td>
</tr>
<tr>
<td>PD.TYP</td>
<td>23</td>
<td>Device type 5,8,hard</td>
</tr>
<tr>
<td>PD.DNS</td>
<td>24</td>
<td>Disk density</td>
</tr>
<tr>
<td>PD.CYL</td>
<td>25-26</td>
<td>Number of tracks (cylinders)</td>
</tr>
<tr>
<td>PD.SID</td>
<td>27</td>
<td>Number of sides</td>
</tr>
<tr>
<td>PD.VFY</td>
<td>28</td>
<td>Verify flag 0-do verify on write</td>
</tr>
<tr>
<td>PD.SCT</td>
<td>29-2A</td>
<td>Sectors/track</td>
</tr>
<tr>
<td>PD.TOS</td>
<td>2B-2C</td>
<td>Sectors/track (track 0)</td>
</tr>
<tr>
<td>PD.ILV</td>
<td>2D</td>
<td>Sector interleave</td>
</tr>
<tr>
<td>PD.SAS</td>
<td>2E</td>
<td>Segment allocation size in sectors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD.TFM</td>
<td>2F</td>
<td>DMA transfer mode</td>
</tr>
<tr>
<td>PD.Exten</td>
<td>30-31</td>
<td>Path extension (not used)</td>
</tr>
<tr>
<td>PD.Stoff</td>
<td>32</td>
<td>Sector/track offset</td>
</tr>
<tr>
<td>PD.ATT</td>
<td>33</td>
<td>File attributes (D S PE PW PR E W R)</td>
</tr>
<tr>
<td>PD.FD</td>
<td>34-36</td>
<td>File descriptor LSN (list of segments)</td>
</tr>
<tr>
<td>PD.PFD</td>
<td>37-39</td>
<td>Dir file desc LSN (of dir holding file)</td>
</tr>
<tr>
<td>PD.DCP</td>
<td>3A-3D</td>
<td>File dir ptr (filename entry in dir file)</td>
</tr>
<tr>
<td>PD.DVT</td>
<td>3E-3F</td>
<td>Device table entry address for user</td>
</tr>
</tbody>
</table>

**Buffer state flag bits:**

- $01$ - buffer modified
- $02$ - current sector
- $04$ - file desc in buffer
- $08$ - end of file sector
- $10$ - end of file
- $20$ - in disk driver
- $40$ - buffer busy
<table>
<thead>
<tr>
<th>Template</th>
<th>DEVICE DESCRIPTOR</th>
<th>RBFMAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFP1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USE DEFS/OS9defs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USE DEFS/RBFdefs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type SET Devic+Obj</td>
<td></td>
<td></td>
</tr>
<tr>
<td>revs SET ReEnt+l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOD rend, devnam, type, revs, fnnam, drvnam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCB $FF</td>
<td>all access modes=b</td>
<td></td>
</tr>
<tr>
<td>FCB $FF, $FF, $40</td>
<td>device address=b</td>
<td></td>
</tr>
<tr>
<td>FCB opt1</td>
<td>option length=b</td>
<td></td>
</tr>
<tr>
<td>optns EQU *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCB DT, RBF</td>
<td>type = 1 for RBFman devices=b</td>
<td></td>
</tr>
<tr>
<td>FCB $03</td>
<td>drive number (0..n)=b</td>
<td></td>
</tr>
<tr>
<td>FCB $02</td>
<td>step rate=b</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCB $40</td>
<td>device type:</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>bit0= 0=5 1/4</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>bit5= 0=noncoco</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>bit6= 0=os9std</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>bit7= 0=floppy</td>
<td></td>
</tr>
<tr>
<td>FCB $01</td>
<td>density:</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>bit0= 0=single</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>bit1= 0=48 tpi</td>
<td></td>
</tr>
<tr>
<td>FCB $00, $23</td>
<td>cylinders (tracks)</td>
<td></td>
</tr>
<tr>
<td>FCB $01</td>
<td>sides</td>
<td></td>
</tr>
<tr>
<td>FCB $01</td>
<td>0= verify disk writes</td>
<td></td>
</tr>
<tr>
<td>FCB $00, $12</td>
<td>sectors/track</td>
<td></td>
</tr>
<tr>
<td>FCB $00, $12</td>
<td>sectors/track (track 0)</td>
<td></td>
</tr>
<tr>
<td>FCB $01</td>
<td>sector interleave</td>
<td></td>
</tr>
<tr>
<td>FCB $01</td>
<td>minimum #sectors/segment alloc</td>
<td></td>
</tr>
<tr>
<td>opt1 EQU *=optns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>devnam FCS /D3/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fnnam FCS /RBF/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drvnam FCS /CCD1isk/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rend EQU *</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a typical RBF device descriptor. You may modify the constants and names (devnam, fnnam) to suit your device name, driver, and characteristics.
Ron - ok, ok <heh-heh>. Have you tried formatting the disk anyway? I can't remember now, but I don't think the desc extensions are used there. Anyway try one of these:

- DIVA 0A or 09
- DIVY 0100 0080
- DIVU 0302 0101

DIVA is the # of bits used for the cylinder number.
DIVY is the # of heads * sectors/trk * shift value.
DIVU mask (# of bits set) is (DIVA-8) bits. The DIVU shift is DIVA-8.
If you've disassembled the driver, you'll see that you end up with the sectors remaining in D (shifted to the left), with the cyl hi in the last one or two bits of B. They mask off those bits, and put them as the cyl hi value. Then D must be shifted right to get back in the correct position. Thus the shift value is dependent upon how many cylinders you have.
I THINK either of the two sets of values above will work. Also I think your drive is 15meg, not 20.
### Disk Format

<table>
<thead>
<tr>
<th>LSN 0 (ID sector)</th>
<th>DD.vars</th>
<th>LSN FORMATS</th>
<th>RBDMAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD.TOT 00</td>
<td>Number disk sectors</td>
<td>DD.ATT 00 DSPEPWDR</td>
<td></td>
</tr>
<tr>
<td>DD.TKS 03</td>
<td>Number tracks</td>
<td>DD.OWN 01 Owner ID</td>
<td></td>
</tr>
<tr>
<td>DD.MAP 04</td>
<td>Bytes in alloc map</td>
<td>DD.DAT 03 Last YMD:HM</td>
<td></td>
</tr>
<tr>
<td>DD.BIT 06</td>
<td>Sectors / cluster</td>
<td>DD.LNK 08 Link count</td>
<td></td>
</tr>
<tr>
<td>DD.DIR 08</td>
<td>Root dir LSN</td>
<td>DD.SIZ 09 File bytes</td>
<td></td>
</tr>
<tr>
<td>DD.OWN 09</td>
<td>Owner's user num</td>
<td>DD.DCR 00 Date create</td>
<td></td>
</tr>
<tr>
<td>DD.ATT 00</td>
<td>Disk attributes</td>
<td>DD.SEG 10 Segment list</td>
<td></td>
</tr>
<tr>
<td>DD.DSK 0E</td>
<td>Internal disk ID</td>
<td>Seg list:</td>
<td></td>
</tr>
<tr>
<td>DD.FMT 10</td>
<td>Format, dens, sides</td>
<td>Up to 48 5-byte</td>
<td></td>
</tr>
<tr>
<td>DD.SPT 11</td>
<td>Sectors / track</td>
<td>Reserved</td>
<td>entries: 3LSN, 2size</td>
</tr>
<tr>
<td>DD.RES 13</td>
<td>Reserved</td>
<td>DD.BT 15 Bootstrap LSN start</td>
<td></td>
</tr>
<tr>
<td>DD.BSSZ 18</td>
<td>Boot size in bytes</td>
<td>DD.DIR</td>
<td>Dir file:</td>
</tr>
<tr>
<td>DD.DAT 1A</td>
<td>Create time YMD:HM</td>
<td>DD.NAME 32 bytes</td>
<td>29 bytes-name</td>
</tr>
<tr>
<td>DD.NAM IP</td>
<td>Disk name(32 bytes)</td>
<td>3 bytes-LSN desc</td>
<td></td>
</tr>
</tbody>
</table>

### LSN 1 (Bit map)

Each bit - 1 cluster of the number of sectors from DD.BIT.

---

Each disk file has at least one sector: the File Desc. This sector (see format above) contains the segment list, which is a list of the sectors used by that file. Each 5-byte entry (in order) points to the next block of sectors: the beginning LSN of the block, and the number of contiguous LSN's from and including the beginning block LSN.

Thus, if your disk files got so fragmented that the file could not be held in 48 blocks of any number of neighboring sectors, the File Desc couldn't handle it. This is extremely unlikely, of course.

The sectors pointed to in the segment list contain the file itself, which might be a m/l program, an ASCII file, or a list of other files.

A file that consists of a list of other files is assigned (by the Attr or Makdir commands) the Directory attribute. The list of files, and THEIR File Desc sector, is kept in a special order (see Dir file above right).

The directory file can have an essentially unlimited number of 32-byte entries consisting of the file name (up to 29 char) and the 3-byte LSN of the filename's File Desc sector. Note that the first two filenames are automatically inserted by RBDMan and they are '.' and '..', which point respectively to the dir file's own File Desc, and the File Desc of the dir file just above it in hierarchy.

DD.DIR points to the LSN of the first File Desc which has the Directory attribute, and is a list of all the files and directory files that you see when you do a 'Dir' of the device holding the disk.

---

3-2-1
CREATE File Mgr Entry RBFRAN

Drop bit 7 of attr parm
Find file LSN
  (file exits?) y----------------->'File Exists'
  ln
  (dir found?) n----------------->'Path not Found'
  ly
Get segment PSN of dir file
Get size of dir file
Allocate >=one sector (segment)
Save number of sectors alloc'd
Save new segment PSN
Seek start of dir file
  l<--------------.
Get 32 byte entry 1
  .<--y (empty spot?)
  1
  ln
  1
  Point to next 32
  1
  (error?) n---------------->l
  1
  ly
  1 (eof ?) n---------------->Error End
  1
  ly
  1 Extend file by 32
  1 Update file size
  1 Read new sector
  l<-----------

Clear 32 bytes
Move <=29 name chars to buffer
Move alloc'd desc LSN to buffer
Write out updated dir file LSN
  1
Clear buffer
State=dir desc
Insert file attr. user ID, time, date
Set link count=1
  1
Check number sectors alloc'd
  1
  .<--y (any sectors left?)
  1
  ln
  1 Set first seg LSN=desc LSN+1
  1
  Set first seg size=sectors+1
  l<----------
Write out file desc LSN
Put file desc LSN in path desc
Zero file size, pos in path desc
Seek 0 in new file
  1
END

3-2-2
Find dir LSN

Find dir LSN

---y (file desc PSN?)

ln (@ - open whole device)

1 (mode=dir?) y----------> 'SD6 error'

ln

Zero seg begin PSN.FSN

Get #sectors from drv table

Store as pd.segment size

Store*256 as pd.file size

1

END

----------------

Check file attr err------> 'No Permission'

PD.pos, FSN, msb seg size=0

Move file attr fm buffer to pd.attr

Move first LSN & segment size to path desc

Move file size to pd.file size

END

Path desc var's:

PD.CP 0B 4 Current file position

PD.SIZ 0F 4 File size

PD.SBL 13 3 Segment beginning file sector (FSN)

PD.SBP 16 3 Segment beginning disk sector (LSN)

PD.SSZ 19 3 Segment size in sectors

PD.ATT 33 1 File attr (D S PE PW PR E W R)

PD.FD 34 3 File desc PSN (the list of sectors for file)

PD.DFD 37 3 Dir file desc PSN (one level up from 34)

PD.DCP 3A 4 Dir file entry pointer to this filename

The FSN, as I call it, is the offset in sectors from the beginning of the actual file position.

The LSN is the actual disk sector that the FSN is equal to.

The PSN is also the actual disk LSN.
CLOSE

File Mgr Entry

RBFMAN

$D3EB

(images=0?) n-------->END

ly

(mode=write?) n----------------->.

ly 1

(file desc ?) n------------------>1

ly 1

Insert date in desc buff

Move file size to desc buff 1

Check disk ID & write buff

Check EOF status

1

END

CHGDIR

File Mgr Entry

RBFMAN

$D43A

Open pathname

1

.----1----.

1 1
data exec

1

Put dr# & file desc LSN in Proc Desc

Return buffer 1

END

SEEK

File Mgr Entry

RBFMAN

$D4FA

.<=n (sector in buffer?)

1

ly

1 Get pos of buff start

1<=y (seek within buff?)

1

ln

1 Get buff within seek

1-------->

Set new pd.pos

1

END
State altered
Request buffer, set PD.BUF
PD.file desc PSN=0
PD.disk ID=0

(1st char='/'?) n------>
ly l
Get device name l

(1st char='@'?)
ln
PD.file desc PSN= Proc Desc default
data/exec dir desc PSN
l<------------------1
PD.DVT=PD.DEV
PD.DTB=static mem+drvbg+u(dif! * drvmem)
l
.<--y (was 1st char '@'?)
ln
Read LSN 0
PD.disk ID=disk ID l
l
l<--y (PD.file desc PSN=0?)
ln
PD.file desc PSN = root dir PSN
l
l
l<------>

Save ptr to pathname

Read file desc LSN

(1st char '!'?) n--------------------->
ly l
Check file attr err=>'No Permsn'
Read 32 bytes
l
l<-------->

Pt to next filename l
Save ptr to name
l
l<---------1
l<--y (unused entry?)

l<--n (same names?) * FOUND NAME *
l
Set PD.dir file PSN & entry ptr
PD.file desc PSN=this LSN
l
l
l (at end of file?) y---------->'EOF error'
l<--------ln

Returns last dir file PSN & entry found.
File desc PSN = the LSN at that dir file position.
IF '"', PSN=0, size= entire disk

3-2-5
INSIDE OS9 LEVEL II
Devices
Section 3

OS9 I/O

SCFMAN FILE

PD-. path descriptor vars   V$-- device table
V. -- device static storage   Q$-- IRQ poll table
P$-- process descriptor

Opening a serial (SCF) file takes the following steps:

#   VAR     MOD   ACTION

1   PD,P0   IOMAN   Allocates a 64-byte block path descriptor.
   PD,M0   IOMAN   Sets access mode desired.
   PD,CN   IOMAN   Sets user cnt=1 for this path desc.

2   PD,DEV   IOMAN   Attaches the device (driver) used:
   V$STAT   IOMAN   Allocates memory for device driver (RS232).
   V,FORT   IOMAN   Sets device address in driver static memory.

3   RS232   IOMAN   The device's init subroutine is called to
           initialize the device.

4   Q$POLL   OS9   Sets up IRQ polling table entry.
           ...   (address, flip & mask bytes, service add,
           Q$PRTY   OS9   static storage, priority of IRQ)

5   V$DRIV   IOMAN   Sets up rest of device table.
   V$DESC   IOMAN   (module addresses of desc, driver, mgr)
   V$FMR   IOMAN   VTAM
   V$USR   IOMAN   Sets user count of device=1

6   PD,O0T   IOMAN   Copies device desc info to path desc.
           ...   (upper/lower case, if, lines/page,
           PD,XOF   IOMAN   file chars, baud rate, echo device )

7   PD,BUF   SCFMAN   Allocates 1 byte buffer.
   V,LNE   SCFMAN   Copies desc lines/page to lines left var.
   PD,BV2   SCFMAN   I$Attach echo device, set dev table ptr.

8   P$PATH   IOMAN   Puts path desc # in proc desc I/O table.
           Returns table pointer to user as path number.

2,3,4,5 only if first time for that device,
              else V$USR = V$USR + 1
              PD,DEV = device table entry.

4 only if device uses IRQ's.

DEVICE DRIVER ENTRIES

INIT   U = device static memory   CC,B < error code
       Y = device descriptor

. Initialize any static vars, constants
. Use P$Irq to place driver IRQ service routine in poll table
. Init controller

3-3-1
INSIDE OS9 LEVEL II
Devices
Section 3

READ
U = device static memory
Y = path descriptor
A <char
CC, B < error code

. Get next char from input buffer in static memory
. If none: copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0

WRITE
U = device static memory
Y = path descriptor
A < char
CC, B < error code

. Put char into static memory output buffer
. Enable ready-to-transmit interrupt
. If full: copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0

GETSTT
U = device static memory
Y = path descriptor
A = status call

. Do wildcard driver call if not handled by IOMAN/RBFman

PUTSTT

TERM
U = device static memory
Y = path descriptor
CC, B < error code

. Wait for output buffer to empty
. Disable any device IRQ's
. Remove device from IRQ polling table

IRQ Service Routine

. Read data if necessary into input buffer.
. If pause char read, set V.PAUS of memory area V.DEV2 <> 0.
. If quit or keybd interrupt is read, send appropriate signal to the last user (V.LPRC) and error code=char.
. Write the output buffer to device until it is empty, disable ready-to-transmit interrupt.
. Send wakeup signal to V.WAKE
. Clear V.WAKE
### INSIDE OS9 LEVEL II

#### Devices

#### Section 3

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V.PAGE</td>
<td>00</td>
<td>Port extended address</td>
</tr>
<tr>
<td>V.PORT</td>
<td>01-02</td>
<td>Device address</td>
</tr>
<tr>
<td>V.LPRC</td>
<td>03</td>
<td>Last active process ID</td>
</tr>
<tr>
<td>V.BUSY</td>
<td>04</td>
<td>Active process ID (dev busy flag) 0=not busy</td>
</tr>
<tr>
<td>V.WARE</td>
<td>05</td>
<td>Process ID to awake after command completed</td>
</tr>
<tr>
<td>V.USER</td>
<td>.</td>
<td>Beginning of file mgr/driver var's</td>
</tr>
</tbody>
</table>

| V.TYPE | 06     | Device parity type                              |
| V.LINE | 07     | Lines til end of page                           |
| V.PAUS | 08     | Pause request 0=None                             |
| V.DEV2 | 09-0A  | Echo device memory area                         |
| V.INTR | 0B     | Interrupt char                                  |
| V.QUIT | 0C     | Quit char                                        |
| V.PCHR | 0D     | Pause char                                       |
| V.ERRR | 0E     | Error collector                                  |
| V.XON  | 0F     | X-ON char                                        |
| V.XOFF | 10     | X-OFF char                                       |
| V.PDLHD | 11-15 | used by Japanese computers                     |
| V.PDLHD | 16-17 | Path desc's head link for device users           |
| V.SCF  | .      | End of SCFman vars                              |

1D= Free for device driver vars

---

V.LPRC is used by the IRQ routine. If a quit or interrupt char is received, the routine should signal the last process to use the device with the signal associated with that char.

This is why the Shell usually catches your <shift-brk> or <brk> multi-task/ abort keystrokes, and takes the appropriate action. Note that if your program uses the device itself, you get the strange alternating set of Shell/ program messages.
## INSIDE OS9 LEVEL II

### Devices

#### Section 3

---

### Module

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSID</td>
<td>00-01</td>
<td>Sync bytes ($87CD)</td>
</tr>
<tr>
<td>MSSize</td>
<td>02-03</td>
<td>Module size</td>
</tr>
<tr>
<td>MSName</td>
<td>04-05</td>
<td>Offset from start to module name string</td>
</tr>
<tr>
<td>MSType</td>
<td>06</td>
<td>Type/lang ($F1)</td>
</tr>
<tr>
<td>MSRevs</td>
<td>07</td>
<td>Attr/revision</td>
</tr>
<tr>
<td>MSParity</td>
<td>08</td>
<td>Header parity</td>
</tr>
</tbody>
</table>

| M筢Mqr | 09-0A  | File manager name offset     |
| MSPDev | 0B-0C  | Driver name offset           |
| MSMode | 0D     | Device capabilities          |
| M Sabha | 0E-10  | Device extended address      |
| MSOpt | 11     | Number of options in initialization table: |

- **IT.DTP** | 12 | Device type (0=SCF) |
- **IT.UPC** | 13 | Case: 0= U/l 1=Upper only |
- **IT.BSR** | 14 | Backspace: 0=bsp only 1=bsp, space, bsp |
- **IT.DLO** | 15 | Delete: 0=bsp over line 1=<cr> |
- **IT.EKO** | 16 | Echo: 0=no echo |
- **IT.APR** | 17 | Auto linfeed: 0=no auto linfeed |
- **IT.NUL** | 18 | Null: number of delay nulls sent after <cr> |
- **IT.PAU** | 19 | Pause: 0=no pause at end of page |
- **IT.PAG** | 1A | Lines per page |

- **IT.BSP** | 1B | Backspace code char from device |
- **IT.DEL** | 1C | Delete-line code from device |
- **IT.EOR** | 1D | End of record code from device |
- **IT.EOF** | 1E | End of file code from dev ('EOF' is echoed) |
- **IT.RPR** | 1F | Reprint line code from device (buffer echoed) |
- **IT.DUP** | 20 | Duplicate line code (all buffer echoed) |
- **IT.PFC** | 21 | Pause code from device |
- **IT.INT** | 22 | Interrupt code from device |
- **IT.QUIT** | 23 | Quit code from device |

- **IT.BSE** | 24 | Backspace code echoed to echo device |
- **IT.OVF** | 25 | Line too long code to echo (bell) |
- **IT.PAR** | 26 | Parity: init byte for ACIA control register |
- **IT.BAU** | 27 | Baud rate |
- **IT.D2P** | 28-29 | Echo device name offset |
- **IT.XON** | 2A | X-on char |
- **IT.XOFF** | 2C | X-off char |
- **IT.COL** | 2C | Number of columns |
- **IT.ROW** | 2D | Number of rows |

End of option table.

**2E-**

Name strings here.

---

3-3-4
## Inside OS9 Level II

### Devices

#### Section 3

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD.P</td>
<td>00</td>
<td>Path number</td>
</tr>
<tr>
<td>PD.MOD</td>
<td>01</td>
<td>Access mode 1=read 2=write 3=update</td>
</tr>
<tr>
<td>PD.CNT</td>
<td>02</td>
<td>Number of paths using this path desc</td>
</tr>
<tr>
<td>PD.DEV</td>
<td>03-04</td>
<td>Device table entry address</td>
</tr>
<tr>
<td>PD.CPR</td>
<td>05</td>
<td>Current Proc ID using this path for I/O</td>
</tr>
<tr>
<td>PD.RGS</td>
<td>06-07</td>
<td>Address of user's register stack</td>
</tr>
<tr>
<td>PD.BUF</td>
<td>08-09</td>
<td>Data buffer (256 bytes) if used</td>
</tr>
<tr>
<td>PD.FST</td>
<td></td>
<td>Beginning of SCFman vars</td>
</tr>
<tr>
<td>PD.DV2</td>
<td>0A-0B</td>
<td>Echo device table ptr (output)</td>
</tr>
<tr>
<td>PD.RAW</td>
<td>0C</td>
<td>Edit flag 0=read/write 1=readln/writeln</td>
</tr>
<tr>
<td>PD.MAX</td>
<td>0D-0E</td>
<td>Readline max char cnt</td>
</tr>
<tr>
<td>PD.MIN</td>
<td>0F</td>
<td>Device use flag 0=my devices</td>
</tr>
<tr>
<td>PD.STS</td>
<td>10-11</td>
<td>Status routine module address</td>
</tr>
<tr>
<td>PD.STN</td>
<td>12-13</td>
<td>Reserved for status routine</td>
</tr>
<tr>
<td></td>
<td>14-1F</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

This section copied by IOMAN from the Device Descriptor:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD.OPT</td>
<td>20</td>
<td>Device class 0-SCF 1-RBF 2-PIPE</td>
</tr>
<tr>
<td>PD.UPC</td>
<td>21</td>
<td>Case 0=upper and lower 1=upper only</td>
</tr>
<tr>
<td>PD.BSD</td>
<td>22</td>
<td>Backspace 0=bsp 1=bsp, space, bsp</td>
</tr>
<tr>
<td>PD.DLO</td>
<td>23</td>
<td>Delete 0=bsp over line 1=cr/lf</td>
</tr>
<tr>
<td>PD.ECH</td>
<td>24</td>
<td>Echo 0=no echo</td>
</tr>
<tr>
<td>PD.ALF</td>
<td>25</td>
<td>Auto if 0=no auto line feed after cr</td>
</tr>
<tr>
<td>PD.NUL</td>
<td>26</td>
<td>Null cnt nulls sent after cr/lf for delay</td>
</tr>
<tr>
<td>PD.PAU</td>
<td>27</td>
<td>Pause lines left before pause; 0=no pause</td>
</tr>
<tr>
<td>PD.PAG</td>
<td>28</td>
<td>Lines / page</td>
</tr>
<tr>
<td>PD.BSP</td>
<td>29</td>
<td>Backspace char</td>
</tr>
<tr>
<td>PD.DEL</td>
<td>2A</td>
<td>Delete-line char</td>
</tr>
<tr>
<td>PD.EOF</td>
<td>2B</td>
<td>End of line char (normally $0D, 0=til EOF)</td>
</tr>
<tr>
<td>PD.EOF</td>
<td>2C</td>
<td>End of file char (read only)</td>
</tr>
<tr>
<td>PD.RPR</td>
<td>2D</td>
<td>Reprint line char</td>
</tr>
<tr>
<td>PD.DUP</td>
<td>2E</td>
<td>Duplicate last line char</td>
</tr>
<tr>
<td>PD.PSC</td>
<td>2F</td>
<td>Pause char</td>
</tr>
<tr>
<td>PD.INT</td>
<td>30</td>
<td>Keyboard interrupt char (ctrl-C)</td>
</tr>
<tr>
<td>PD.QUT</td>
<td>31</td>
<td>Keyboard abort char (ctrl-Q / Break)</td>
</tr>
<tr>
<td>PD.BSE</td>
<td>32</td>
<td>Backspace echo char</td>
</tr>
<tr>
<td>PD.OVF</td>
<td>33</td>
<td>Line overflow char (Bell code)</td>
</tr>
<tr>
<td>PD.PAR</td>
<td>34</td>
<td>Device init byte (parity)</td>
</tr>
<tr>
<td>PD.BAU</td>
<td>35</td>
<td>Baud rate code</td>
</tr>
<tr>
<td>PD.DTP</td>
<td>36-37</td>
<td>Offset to DEV2 name string</td>
</tr>
<tr>
<td>PD.XON</td>
<td>38</td>
<td>X-ON char for ACTA</td>
</tr>
<tr>
<td>PD.XOFF</td>
<td>39</td>
<td>X-OFF char</td>
</tr>
</tbody>
</table>

Input of a keyboard INT/QUT character returns that char as the I/O error code, and sends an interrupt/abort signal to the last active user process of this path.
INSIDE OS9 LEVEL II
Devices
Section 3

====================================================================
Template DEVICES DESCRIPTOR SCFMAN
====================================================================

ifp1
use /dd/defs/defsfile
dnc

type SET DEVIC+OBJECT
revis SET REENT+1

MOD len, nam, type, revis, mgd, drvr

FCB READ. + WRITE. mode
FCB $FF
FCB $FF00 device address
FCB opt-*-1 option byte cnt
FCB DT.SCF SCF device

FCB 0 case= UPPER and lower
FCB 1 backspace= bs sp bs
FCB 0 delete= bs over line
FCB 1 auto echo
FCB 1 auto line feed
FCB 0 no nulls on CR
FCB 0 no page pause
FCB 24 lines per page
FCB 08 backspace char
FCB $18 delete line char
FCB $0D end of record char
FCB 0 no end of file char
FCB 04 reprint line char
FCB 01 dup last line char
FCB $17 pause char
FCB 3 abort char
FCB 5 interrupt char
FCB $0B backspace echo char
FCB 07 line overflow (bell)
FCB 0 printer type
FCB 4 baud rate=2400
FDB echo echo device

opt EQU *

nam FCS "Remote"

mgr FCS "SCF" file mgr name
drvr FCS "CCIO" driver name
echo FCS "T1" echo device

EMOD

len EQU *

END

====================================================================
Using 'Shell </remote >/t1 >/t1' allows you to use the CoCo keyboard while visual output is redirected (and input echoed) to a terminal display connected to the RS-232 port.
====================================================================

3-3-6
Opening a pipe (PIPEMAN) file takes the following steps:

<table>
<thead>
<tr>
<th>#</th>
<th>VAR</th>
<th>MOD</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PD.PD</td>
<td>IOMAN</td>
<td>Allocates a 64-byte block path descriptor.</td>
</tr>
<tr>
<td></td>
<td>PD.MOD</td>
<td></td>
<td>Sets access mode desired.</td>
</tr>
<tr>
<td></td>
<td>PD.CNT</td>
<td></td>
<td>Sets user cnt=1 for this path desc.</td>
</tr>
<tr>
<td>2</td>
<td>PD.DEV</td>
<td>IOMAN</td>
<td>Attaches the device used:</td>
</tr>
<tr>
<td></td>
<td>VSSTAT</td>
<td></td>
<td>Allocates memory for device driver (none).</td>
</tr>
<tr>
<td></td>
<td>V.PORT</td>
<td></td>
<td>Sets device address in driver static memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( address = 00 0000 )</td>
</tr>
<tr>
<td>3</td>
<td>PIPER</td>
<td></td>
<td>The driver's init subroutine is called to initialize the device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(does nothing).</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>No interrupts used by PIPER.</td>
</tr>
<tr>
<td>5</td>
<td>VSDRIV</td>
<td>IOMAN</td>
<td>Sets up rest of device table.</td>
</tr>
<tr>
<td></td>
<td>V$DESC</td>
<td></td>
<td>( module addresses of desc, driver, mgr)</td>
</tr>
<tr>
<td></td>
<td>V$FMGR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>V$USR$S</td>
<td></td>
<td>Sets user count of this pipeman=1</td>
</tr>
<tr>
<td>6</td>
<td>PD.OPT</td>
<td>IOMAN</td>
<td>Copies device desc info to path desc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(just type= Pipe)</td>
</tr>
<tr>
<td>7</td>
<td>PD.BUF</td>
<td>PIPMN</td>
<td>Allocates 256-byte buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sets begin, end, nextchar ptrs in PD.</td>
</tr>
<tr>
<td>8</td>
<td>P$PATH</td>
<td>IOMAN</td>
<td>Puts path desc # in proc desc I/O table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns table pointer to user as path number.</td>
</tr>
</tbody>
</table>

2,3,5 only the very first time a pipe is used,
else V$USR$S = V$USR$S + 1
PD.DEV = device table entry
4 not used at all

Note that both the driver and descriptor (Piper, Pipe) are only dummy modules, there just to make IOMAN happy.
### INSIDE 0S9 LEVEL II
#### Devices
#### Section 4

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD.PD</td>
<td>00</td>
<td>Path number</td>
</tr>
<tr>
<td>PD.MOD</td>
<td>01</td>
<td>Access mode 1=read 2=write 3=update</td>
</tr>
<tr>
<td>PD.CNT</td>
<td>02</td>
<td>Number of paths using this path desc</td>
</tr>
<tr>
<td>PD.DEV</td>
<td>03-04</td>
<td>Device table entry address</td>
</tr>
<tr>
<td>PD.CFR</td>
<td>05</td>
<td>Current Proc ID using this path for I/O</td>
</tr>
<tr>
<td>PD.RGS</td>
<td>06-07</td>
<td>Address of user's register stack</td>
</tr>
<tr>
<td>PD.BUF</td>
<td>08-09</td>
<td>Data buffer (256 bytes) address each Create</td>
</tr>
<tr>
<td>PD.FST</td>
<td></td>
<td>Beginning of Pipeman vars</td>
</tr>
</tbody>
</table>

0A  Read user
0B  Number read users
0C  Read signal
0D  End of line char

0E  Write user
0F  Number write users
10  Write signal
11  Not used

12-13 End of buffer
14-15 Pointer to next address to store char
16-17 Pointer to next address to read char
18  Data flag 0=no data in circular buffer

Pipeman uses no static memory. Instead, it allocates a 256 byte buffer each time a 'file' is created. This buffer is returned when the last user has closed a path to it, or there are no more readers.

Note: these are for Level One. I haven't had a chance to check on L-II vars, but the concept will be the same, with the exception that Pipeman will do an F$Move of the data between process maps.
INSIDE OS9 LEVEL II
Devices
Section 5

GENERAL DRIVER NOTES

LEVEL TWO DEVICE ADDRESSES

(Message from me to CompuServe OS9 Forum 24Mar87:)

Finally went looking for the reason why I've been telling everyone that their extended device addresses had to be $07FXXX instead of the old L-I $FFXXXX. Here's the dope:

L-II IOMan (just like a GIMIX) takes the address ($07FF) top bytes, and converts it to an I/O block number... on the CoCo, it translates to block $3F. Well, this makes sense as far as it goes, as extended address $07FXXX is indeed the top of mem; that is, the last block or $3F block.

It then looks to see if that block is already mapped into the system 64K map... if it’s block $3F, it already is, cuz that’s the kernel and I/O area from $E000-FFFF.

BUT! If the extended address does NOT translate out to $3F ($FFFF = block number $FF!!), then it maps that block into the system map. And ignores it as RAM cuz it's obviously I/O, right? So you just lost 8K in your System 64K map.

8K is a lot to take away from the system map, and that's when those of you using Rogue got the dreaded 207 error for no seeming reason.

You also got the error if it couldn't map the block in. This error number has been changed to 237 (no ram), in the latest versions, btw.

Since the converted logical address would also be wrong, some things died. Devices with hard coded addresses had fewer problems.

That's the scoop, guys... so make sure to use the $07FXXX if writing up new device descriptors. That is, offset $0E in your device descriptor must be = $07 and the next =$FX.

On the other hand, $00 0XXX should be okay also, as block 00 is also always in the system map.

$CF SPECIAL CHARs

As you know, SCF drivers are responsible for sending either an S$Abort (for character matching V.QUIT) or S$Intrpt (char = V.INTR) signal to the last process (V.LPRC) that used the device.

A note about the above... character matching is done against the V.xxx static memory variables, NOT against the path descriptor PD.yyy equivalents. This is even though the V.xxx were set by SCF to the PD.yyy characters when the process gained the use of the device.

Why not just use the PD stuff? Because most devices are IRQ-driven, and there's no easy way for OS9 to get the path descriptor pointer to the asynchronous IRQ code that is servicing that driver. Hence they are copied to the V.xxx driver memory which IS known, as IOMan has it in it's interrupt polling table.
RBF THINGS

The Device Descriptor describes the maximum capabilities of the device; the Path Descriptor is used for variables pertaining to the file itself (pos, length, lsn's, dirs, etc); and the Drive Tables are for info about THAT one diskette currently in the drive (format, tracks, sectors, bitmap size, root dir, id, which track the head is pointing to, whether a process is changing the bit map, etc).

Those of you who write RAMdisk drivers usually follow the lead of the floppy drivers. Okay, but some parts are different. For example in your Init, you should probably set the DD.TOT to the actual sector size of the "drive". And unless you wish to use it as some kind of flag, there is NO need to do anything to DD.TRAK. That's done there only so floppy drives can restore to track zero the first time they're called. If your driver doesn't need it, don't mess with it.

IRQ's On LEVEL TWO

Let's take a quick look at how ACIAPAK sets up for interrupts, to give other driver writers some help.

ACIAPAK Init Routine:

Does an F$IRQ call
    Stops all interrupts
    Resets the CART PIA line for no Multi-Pak FIRQ's
    Gets Direct Page 0092 (GIME IRQ register shadow)
    OR's it with 01 to enable CART-->IRQ conversions
    Stores that value back at 0092 and FF92
    Restores the CC register
    Sets the MPI slot for CART from slot 0

What CLOCK Does on Interrupt:

    On an IRQ, Clock read GIME FF92 IRQ register
    OR'd that value into Direct Page 00AF
    JSR'd the Interrupt Polling Routine...

ACIAPAK Interrupt Routine:

    Get Direct Page 00AF (contains FF92 IRQ read by Clock)
    NOT with 01 to indicate that CART IRQ was read
    Store that value back at 00AF
    Do the interrupt routine
    Go back and check for another IRQ before RTS

OTHER L-II DRIVER CHANGES

Because the system map is so much like under L-I, only a few changes must be made. The most obvious is the interrupt handling, as discussed above. Timing loops have to compensate for the 2Mhz speed, also.

For RBF devices that must change slots, the main (and sometimes almost only) change is that D.DMAReq has moved from 006A to 008A.

3-5-2
The file managers take care of moving data between system maps, so many old drivers will work fine (once the descriptor address is changed as pointed out). For example, once the address has been changed the Disto Parallel Printer port driver works.

One last note: CC3DISK no longer turns on precompensation on the inner tracks. Supposedly most drives never needed it.
INSIDE OS9 LEVEL II

Windows
THE WINDOW DRIVERS

The windowing system on the CoCo-3 is composed of the window device descriptors, the main driver CC3IO specified in those descriptors, and several co-modules that handle window output.

The modules and a schematic of their relationship:

Term        - Actually, the W0 descriptor OR a VDG descriptor
W1-W7       - Window descriptors
W           - Special window descriptor
CC3IO        - Keyboard scanning (60 times a second if key down)
               Joystick/mouse reads
               Some stat calls
VDGInt      - Emulates L-I v2.0 gfx environment
               Adds hires gfx screens mapped into proc space
WindInt     - Preprocessor for hi-level windowing/menu calls
               plus window codes
GrfInt      - Preprocessor for window codes
               Some stat calls
GrfDrv      - Text/gfx display

----------------------------------------
IOMAN
  1
  CC3IO - Term W W1 W2 .. Main driver/desc
  1
  .<----either-1-or----------.
  1    1    1
  1    GrfInt    WindInt Output processing
  1    1------------
VDGInt
  1
   GrfDrv Screen data

(video output)

COMPARISON WITH OTHER I/O DEVICES

Like other OS9 devices, reading and writing and stat calls are done through a main driver. Each device has its own address, static memory, and has an input buffer for type-ahead. Outputted characters are not queued, but go straight to the screen.

Unlike others, though, each window also shares the same input device (the keyboard or mouse). They also share use of the GIME chip. This means that some way must be used to keep track of which window sets up its display on the GIME, and which window gets the input from the keyboard. For this purpose, all of the window devices also share a common or global memory.
This global memory is located at in block 00, extended address 001000-001FFF, and is always mapped in for the CoCo terminal driver modules to use. A very preliminary and cursory look at this memory area is provided in the next section of the book.

The /W descriptor also introduces a new technique. This wildcard device flags CC3IO to open the next free window in place of it. I think that requesting the name from a path opened using /W will instead return /Wx instead (x=number).

Instead of hardcoding window numbers, good L-II programs that need to open another virtual terminal should use /W.

**CC3IO**

CC3IO is very similar to it's L-I (ver 2.0) counterpart, CC1O. Some of it's code is even the same for the keyboard, lo-res mouse read, and so on. However, where CC1O used CO80 or CO32 as comodules to handle the screen output, CC3IO now passes codes on to the GrfInt/GrfDrv or VDGInt comodules. (The name "CO80" can still be found within CC3IO, but was probably there just for debugging purposes, as it is no longer used.)

**VDGINT**

VDGInt contains the equivalent of the Level One CO32 and GRFO modules. It handles the 32x16 text screens, semi-graphics and original VDG-style graphics screens.

Because of this emulation, you can still run many older programs that ran on the CoCo-1/2's, including TSEDIT.

In addition, VDGInt provides for new screens that allow speed-dependant programs to take advantage of the CoCo-3's high resolution graphics. Unlike the GrfInt screens that are not mapped into a program's space, VDGInt graphics screens are. This means that games like Koronis Rift can directly access the screen memory to be displayed, allowing much faster updating of the screen than by using escape codes.

VDG text screens are normally allocated from the system map, as allocating a full 8K block just for a 512 byte display would be wasteful. To provide compatibility, the use of the SS.AlphaS GetStat call WILL map the screen into the caller's task space (since it returns the address within a logical 64K area), along with any other system variables that just happened to be in the same system map block. For this reason, programs that use this call should be careful to stay within the 32x16 screen area, lest they accidentally write over crucial system data.

Windows within a screen are not provided for, although it is possible to set up more than one VDG screen. And, you can still <CLEAR-key> between these screens and normal windowing screens.

Character and graphics functions are not provided for the CoCo-3 specific modes. The only text output is through use of the 32x16 character display.
GRFiNT/WINDINT

GRFiNT takes the parameters passed with a window code (as when you do a "display 1b 31 5 38"), checks them for values exceeding limits or specifications, and stores the possibly converted parameters in the system map global memory and window tables.

GRFiNT then calls GRFDrv with an internal code, which is used as a table index to call the appropriate GRFDrv subroutine for any screen manipulation.

WINDINT will be included with the Multiview graphics shell package. It will take the place of GRFiNT, providing the same calls plus adding new ones for creating pull-down menus, boxed windows, scroll bars and other hi-level windowing abilities.

GRFDrv

GRFDrv is the module that does any actual storage or drawing of data on the screen. It also handles allocation of screen memory and buffers. In other words, anything specific to the CoCo-3.

Both GRFiNT and WINDINT will use GRFDrv as the driver that manipulates the video data. By breaking things up this way, it's possible for perhaps just a new GRFDrv to be written for other display devices, or the next CoCo.

The most unique aspect of the GRFiNT/GRFDrv combination for lovers of L-II is that it's code size, and the need to have direct access to so much memory (like 32K for each gfx screen), caused the authors of CoCo-3 L-II to adopt what amounts to an extension of the 64K system map into another 64K space to handle the memory needed.

A CLOSER LOOK:

CC3IO

On initialization, CC3IO inserts its IRQ handler vector into D.AltIRQ at $00B2 in the direct page variables. It also sets vectors for window select, mouse reads and the terminal bell (this is used by CLOCK's F$Alarm call).

Depending on the device type ($80= window, else= VDG), it will link or load, and initialize the Interface module required. Obviously, VDG device types use VDGInt. Window devices cause CC3IO to first try locating WindInt. If that fails, it then goes after GRFiNT.

On IRQ's, CLOCK calls CC3IO as a subroutine to read the keyboard, check for fire buttons, decrement the mouse scan delay, and send signals to processes needing them.

The Write routine passes all the characters onward to the Interface modules, but can be requested by them to read more than one parameter for escape codes.

The CLEAR key flip between windows is also caught during interrupts, which you can see by holding CLEAR down while doing disk access. Be careful, though - this causes my machine to crash.
Other than that, CC3IO really knows very little about windows.

CC3IO also handles these:

\[
\begin{align*}
\text{GETSTATS} & \quad \text{SETSTATS} \\
\text{SS.ComSt} & \quad \text{SS.ComSt} \\
\text{SS.Mouse} & \quad \text{SS.Mouse} \\
\text{SS.Montr} & \quad \text{SS.Montr} \\
\text{SS.KeySns} & \quad \text{SS.KeySns} \\
\text{SS.Joy} & \quad \text{SS.Tone} \\
& \quad \text{SS.GIP} \\
& \quad \text{SS.SSig} \\
& \quad \text{SS.MsSig} \\
& \quad \text{SS.Relea} \\
& \quad \text{SS.Open}
\end{align*}
\]

GRFIN T

GrfInt has six entry points, Init, Write, Getstt, Setstt, Term, and SetWindow. At offset 0026 begins the window escape code table, each entry made up of a parameter count, vector, and a code byte to be used for internal GrfDrv calls.

On initialization, GrfInt links or loads "grfdrv" or ".../CMDS/grfdrv". GrfDrv MUST end up on an 8K block exact boundary, which is why it should be loaded off disk. GrfInt calls GrfDrv's Init routine and then unlinks it. This causes GrfDrv to be unmapped from the system task, which is okay as GrfDrv has already moved itself over to the second system map.

GrfInt moves a default palette into global memory where other modules may find it. This table is listed later.

GrfInt sets up the window entry tables, screen tables, and requests system memory for the graphics cursor tables.

As said before, it handles the task of getting all the parameters for the window display codes. It checks for a valid window destination. Parameters are collected and passed onto GrfDrv for execution.

Loading of Get/Put buffers is partially taken care of here, too. GrfInt reads in up to 72 bytes at a time into a global buffer for GrfDrv to read from.

It also sets the page length according the window size, does most of the window Select routine, and computes relative coordinates.

GRFIN T also handles these:

\[
\begin{align*}
\text{GETSTATS} & \quad \text{SETSTATS} \\
\text{SS.ScSiz} & \quad \text{SS.Open} \\
\text{SS.Palett} & \quad \text{SS.MpGPB} \\
\text{SS.FBRegs} & \quad \text{SS.DfPal} \\
\text{SS.DfPal} & \quad \text{SS.DfPal}
\end{align*}
\]
GRFDRV

After being loaded by GrfInt or WindInt, GrfDrv is called to initialize itself. It sets up the second task map (Task One, which is reserved, as is task zero, for the system use) to contain itself, global system memory, and areas for swapping in buffers and screens to access. This map looks like:

<table>
<thead>
<tr>
<th>Logical Block Address</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0000-1FFFF</td>
<td>System Global Memory</td>
</tr>
<tr>
<td>1 2000-3FFFF</td>
<td>Buffers mapped in here</td>
</tr>
<tr>
<td>2 4000-5FFFF</td>
<td></td>
</tr>
<tr>
<td>3 6000-7FFFF</td>
<td>Grfdrv</td>
</tr>
<tr>
<td>4 8000-9FFFF</td>
<td>Screens mapped in here</td>
</tr>
<tr>
<td>5 A000-BFFFF</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>6 C000-DFFFF</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>7 E000-FDFFF</td>
<td>&quot; &quot;</td>
</tr>
</tbody>
</table>

To get to GrfDrv, GrfInt sets up a new stack with GrfDrv's entry point as the PC, then jumps via direct page vector 00A9 to OS9p1. OS9p1 copies the reserved Task One DAT Image into the GIME's second DAT set, flips over to the GrfDrv map, and does a RTI.

Returning to the normal system map (back to GrfInt) is just the opposite, except the vector at 00A9 is used to flip back to the always set up Task Zero system map.

Interrupts are still enabled on the GrfDrv map, and OS9 saves which system map (0 or 1) it was in when the interrupt occurred. After servicing the interrupt, OS9 resets the DAT to the correct task number.

GrfDrv handles all character writing (text or graphics) and graphics routines (line, point, etc).

It checks for window collisions, sets the GIME, translates colors, handles buffers, and executes terminal codes such as CLS, INSLINE, etc.

Allocation and release of buffer and video memory is also done within GrfDrv.

SCREEN MEMORY

Screen memory is allocated using F$AlHRAM (from high block numbers at the top of memory), because the GIME requires contiguous physical memory for display, and there's a better chance of finding such up there. The OS9 kernel gets program and data blocks from the lower end.

Actually, it really shouldn't matter all that much where you found contiguous RAM, but perhaps they felt it was safer up high. Since we have no ROM blocks to map into DAT Images as a safe area (for blocks not used in a program map), the DAT Free marker used by the CoCo (333E) means that a video page (3E) is all that should get clobbered if a bad program runs amuck through it's logical address space. (That is, unless it should run into the GIME and I/O page at XFFXX!)
Each new window doesn't necessarily take up a lot more memory. If you open a window on a previously allocated screen, it's still going to use that screen memory. It's inside that screen, and so is also inside that memory block or blocks.

Graphics screens are allocated by blocks, since the smallest form uses 16K or two blocks. When all the windows on a screen are closed, all the blocks are returned to free memory.

Text screens are allocated a block at a time, and that block is divided up into at least two screens, if they are both 80 column (4K each) screens. So you can have two 80's, one 80 and two 40's, or four 40's per 8K RAM block. That is, you can if you apply the patch to GrfDrv that's in the BUGS section of this manual. See it for more details.

Obviously, it makes more sense, memory-wise, to use text screens where feasible.

**MISC WINDOW TIPS**

The keyboard mouse toggled on and off by <CTRL-CLEAR> changes the arrow keys into a hires joystick, and the function keys into fire buttons. I believe that it takes over in place of the external right-hand joystick. In this mode, the arrow keys are set up as:

- **Arrow** - move 8 positions
- **Shift-arrow** - move 1 position
- **Ctrl-arrow** - move to far edge

If you've set the proportional switch and are using the stdfonts character set, change the font to C8 02 for a better display.

Each device (TERM, Wx) has a 128 byte input queue. This means that you can go to an inactive window, type something blindly on it. Then if you started a program on that window, what you typed previously will be immediately read. For example, if you typed "dir" on W3, then went back and "shell <>>>/w3&", the dir command would be executed by the new shell.

In most cases, it might be better to use the Forngd, Backngd text color set commands, instead of the Palette command. There are eight colors already provided for, and except for two color graphics windows, should be easier to use and remember.

Want to see what your StdPtrs file looks like? Merge them into a window. Open a 320x192 graphics window for best results. Then "display 1B 4E 0100 0050" to move the graphics cursor to an open spot. Now you can "display 1B 39 CA p", where p=1-7 to see how the various pointers look.
AREAS OF INTEREST

For those who might wish to customize their system by changing some of the module defaults, and could use a quick reference to the tables used, here are some helpful assembly areas:

CC3IO

* Keyboard & Mouse Delay Init (1st device):

  007D 861E lda #30 1/2 second
  007F A78861 sta $61,x set keybd delay constant
  0082 A78829 sta $29,x and first delay
  0085 8603 lda #$03 1/20 second
  0087 A78862 sta $62,x secondary delay

  008A 4A deca A=02
  008B A784 sta ,x ($1000)=02
  008D 6C883C inc $3C,x mouse flag
  0090 8601 lda #$01
  0092 A7883D sta $3D,x right joystick
  0095 8678 lda #120 2 seconds
  0097 A7883E sta $3E,x set button timeout
  009A CCPFFFD ldd #$FFFF
  009D ED8828 std $28,x init keyboard vars
  00A0 ED882B std $2B,x
  00A3 CC0078 ldd #$0078 set ss.mouse for device
  00A6 EDC828 std U0028,U (scan rate & timeout)

-----

* Keyboard Mouse Coord Deltas:

  Normal, Shift, Control

  00F4 0801 fcb 8,1 right
  00F6 027F fdb 639
  00F8 F8FF fcb -8,-1 left
  00FA 0000 fdb 0
  00FC 0801 fcb 8,1 down
  00FE 00BF fdb 191
  0100 F8FF fcb -8,-1 up
  0102 0000 fdb 0

-----

* Special Key Code Table:

  Normal, Shift, Control

  05A2 406000 fcb $40,$60,$00 @
  05A5 0C1C13 fcb $0C,$1C,$13 up
  05A8 0A1A12 fcb $0A,$1A,$12 down
  05AB 0B1810 fcb $0B,$18,$10 left
  05AE 091911 fcb $09,$19,$11 right
  05B1 202020 fcb $20,$20,$20 space
INSIDE OS9 LEVEL II
Windows
Section 1

05B4 303081  fcb $30,$30,$81 0 0 case
05B7 31217C  fcb $31,$21,$7C 1 !
05BA 322200  fcb $32,$22,$00 2 "
05BD 33237E  fcb $33,$23,$7E 3 #
05C0 342400  fcb $34,$24,$00 4 $
05C3 352500  fcb $35,$25,$00 5 %
05C6 362600  fcb $36,$26,$00 6 &
05C9 37275E  fcb $37,$27,$5E 7 '
05CC 38285B  fcb $38,$28,$5B 8 ( [
05CF 39295D  fcb $39,$29,$5D 9 ) ]

05D2 3A2A00  fcb $3A,$2A,$00 : *
05D5 3B2B7F  fcb $3B,$2B,$7F ; +
05DA 3C3C7B  fcb $2C,$3C,$7B , <
05DB 2D3D5F  fcb $2D,$3D,$5F - =
05DE 2E3E7D  fcb $2E,$3E,$7D . > \n05E1 2F3F5C  fcb $2F,$3F,$5C / ? /

GRFINI

* Default Palette Table:
  * whi, blu, blk, grn, red, yel, pur, cyn

02F2 3F090012  FCB $3F,$09,$00,$12,$24,$36,$2D,$1B
02FA 3F090012  FCB $3F,$09,$00,$12,$24,$36,$2D,$1B

GRFDRV

L03C? ldd #$C801 set default font for gfx windows

---------

L08CC  equ * Translate Color For RGB:
pshs x

L08D9  equ * skip if not composite color

---------

4-1-8
```
L08DB equ *

<table>
<thead>
<tr>
<th>FCB</th>
<th>64 Color Translation Table:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00, $0C, $0E, $07, $09, $05, $10</td>
<td></td>
</tr>
<tr>
<td>$1C, $2C, $0D, $1D, $0B, $1B, $0A, $2B</td>
<td></td>
</tr>
<tr>
<td>$22, $11, $12, $21, $03, $01, $13, $32</td>
<td></td>
</tr>
<tr>
<td>$1E, $2D, $1F, $2E, $0F, $3C, $2F, $3D</td>
<td></td>
</tr>
<tr>
<td>$17, $08, $15, $06, $27, $16, $26, $36</td>
<td></td>
</tr>
<tr>
<td>$19, $2A, $1A, $3A, $18, $29, $28, $38</td>
<td></td>
</tr>
<tr>
<td>$14, $04, $23, $33, $25, $35, $24, $34</td>
<td></td>
</tr>
<tr>
<td>$20, $3B, $31, $3E, $37, $39, $3F, $30</td>
<td></td>
</tr>
</tbody>
</table>
```
INSIDE OS9 LEVEL II
Windows
Section 2

* System and CC310 Memory Map (block 00)
* Our personal disasm variable map from Rogue.
* Kevin Darling 14 Feb 87, 30 Mar 87
* Kent Meyers
* Not necessarily accurate for latest versions.
* -----------------------------------------------
* Global and CC310 Memory Starts at $01000:

1000   rmb 1
1001   rmb 1
1002   rmb 1  map side (grfdrv)
1007   rmb 2  grfdrv stack pointer
1009   rmb 1  monitor type (0,1,2)
100A   rmb 1  same as active dev flag
100B   rmb 1  v.type of this dev
100C   rmb 2  device static memory ptr
100E   rmb 1  WindInt map flag?
100F   rmb 6  FS$Alarm time packet
1015   rmb 1  FS$Alarm process id
1016   rmb 1  FS$Alarm signal code
1017   rmb 2  terminal bell vector
1019   rmb 2  ptr to default palette ptr
101B   rmb 1  tone duration in ticks
101C   rmb 1  bell flag
101D   rmb 3

1020   rmb 2  active window devmem
1023   rmb 1  screen changed flag
1024   rmb 1  $80=grf/windint,$02=vdg found
1025   rmb 2
1027   rmb 1  last keybd row fnd
1028   rmb 1
1029   rmb 1  repeat delay cnt now
102A   rmb 5

102F   rmb 1  grfdrv init'd flag
1030   rmb 1  SHIF'T key down
1031   rmb 1  CTRL key down
1032   rmb 1
1033   rmb 1  ALT key down
1034   rmb 1  keysns byte
1035   rmb 1  same key flag
1036   rmb 1  SHIF'T/CLEAR flg
1037   rmb 1
1038   rmb 1  grfdrv init'd flag
1039   rmb 2
103B   rmb 1  mouse sample tick counter

* -----------------------------------------------
* Mouse Packet: ($20 bytes)

4-2-1
INSIDE OS9 LEVEL II
Windows
Section 2

103C rmb 1 00
103D rmb 1 fire bit$, rdflg 01
           bit 0 = fire button $
           bit 1 = side (0 = right, 1 = left)
           bit 6 = set if was keybd mouse
103E rmb 1 timeout constant 02
103F rmb 1 keybd flag 03
1040 rmb 1 04
1041 rmb 1 cntr 05
1042 rmb 2 0 - FFFF cnt 06
1044 rmb 1 fire chg bit 08
1045 rmb 1 fire chg bit 09
1046 rmb 1 up time 0A
1047 rmb 1 up time 0B
1048 rmb 1 chg counter 0C
1049 rmb 1 chg counter 0D
104A rmb 1 down time 0E
104B rmb 1 down time 0F
104C rmb 2 10
104E rmb 2 returned X 12
1050 rmb 2 returned Y 14
1052 rmb 1 16
1053 rmb 1 0 = old, 1 = hires 17
1054 rmb 2 X coordinate 18
1056 rmb 2 Y coordinate 1A
1058 rmb 2 X window 1C
105A rmb 2 Y window 1E

* ----------------------------------------------------------
1060 rmb 1 mouse sample rate
1061 rmb 1 first key delay ticks
1062 rmb 1 secondary repeat ticks
1063 rmb 1 enable kbdmouse toggle flag
1064 rmb 1 one shot ignore CLEAR key flag
1065 rmb 1 fire button dwn (P1 = 01 P2 = 04)
1066 rmb 1 mouse to use (AND 66+67<0:update packet)
1067 rmb 1 mouse coord changed flag
1068 rmb 6 comodule entry vectors...
106A rmb vdgint entry
106E rmb grfdrv entry
1070 rmb 1 move data cntr for buffers
1071 rmb 4 32-bit window alloc map
1075 rmb 2 ptr to 576 byte gfx tables

10BF rmb 1 cc3io L0116 flag (chg mouse?)
10C2 rmb 2 cc3io shift-clear key sub (L0614)
10C4 rmb 2 cc3io set mouse sub (L06AE)
10C6 rmb 1 fire not read: zero if ssig sent
10C7 rmb 16 palette reg data (sys default)
10E7 rmb

1100 rmb x grfdrv variables
1200 rmb x data buffer for gpload
1280 rmb x window tables ($40 each)
1290 rmb x window table base offset used
1A80 rmb x screen tables

4-2-2
INSIDE OS9 LEVEL II
Windows
Section 2

* ----------------------------------------------
* GrfInt/GrfDrv Vars:

grfdrv  equ $0100 use for global offset

110E    rmb 1  char bsw bits
1120    rmb 2  ellipse parms:
1122    rmb 2  .
1124    rmb 2  .
1126    rmb 2  .
112E    rmb 2  windentry now
1130    rmb 2  screen table now
1132    rmb 3  3 byte buffer table
1135    rmb 3  grp,offset
1138    rmb 3  grp,offset returned (new)
113B    rmb 2  end of vars ptr?
113D

1147    rmb 2  HBX, LBX
1149    rmb 2  HBY, LBY
114B    rmb 2  current X
114D    rmb 2  current Y
114F    rmb 2  HSX, LSX
1151    rmb 2  HSY, LSY
1153    rmb 2  Circle, ellipse, arc
1155    rmb 2  Ellipse, arc
1157    rmb 1  GRP
1158    rmb 1  BPN
1159    rmb 1  SVS
115A    rmb 1  PRN
115B    rmb 2  BX putgc
115D    rmb 2  BY putgc
115F    rmb 1
1160    rmb 1  STY marker
1161    rmb 1  fore rgb data WE:06
1162    rmb 1  back rgb data WE:07
1163    rmb 1  bytes/row  SC:04
1164    rmb 2  lset vector?  WE:16
1166    rmb 2  Pset offset  WE:0F
1168    rmb 2  grfdrv lset  WE:14
116A    rmb 2  max x coord  WE:1B
116C    rmb 2  max y coord  WE:1D
116E    rmb 2  X pixel cnt
1170    rmb 2  Y pixel cnt
1172    rmb 2  get/put ow save screen strut
117D    rmb 1  buffer block # (get block)
117E    rmb 2  buffer offset grp/bfn
1180    rmb 2  HBL, LBL
1182    rmb 2  3 byte extended screen address
1185    rmb 2  temp
1187    rmb 16  grfdrv (sysmap 1) DAT Image
1197    rmb 1  temp
1199    rmb 2  this windentry ptr

4-2-3
INSIDE OS9 LEVEL II
Windows
Section 2

119B  rmb 1  counter temp
119C  rmb 1
119D  rmb 2  offset to buffer in block

1280  rmb x  windentries: base=1290

* ---------------------------------------------------
* Window Entry: ($40 each)

  org −$10
-10 W.  rmb 2  screen table ptr
-0E    rmb 1  back wind# link
-0D    rmb 2  screen logical start
-0B    rmb 2  CPX, CPY
-09    rmb 2  SZX, SY
-07    rmb 2  x,y sizes?
-05    rmb 2  cursor address
-03    rmb 1
-02    rmb 1
-01    rmb 1
 00    rmb 1  sty marker byte
 01    rmb 1
 02    rmb 1  X byte cnt (cwarea)
 03    rmb 1  cwarea temp
 04    rmb 2  bytes/row
 06    rmb 2  fore/back prn
 08    rmb 1  def attr byte FU TTTTBBB
 09    rmb 1  char bsw bits: (default=$89)

   80 TChr
   40 Under
   20 Bold
   10 Prop
   08 Scale
   04 Invers
   02 NoCurs
   01 Protect

0A  rmb 1  LSET #
0B  rmb 1  GRP for font
0C  rmb 2  font offset
0E  rmb 1  GRP for PSET
0F  rmb 2  pset offset?
10  rmb 1  LCD mode
11  rmb 1  overlay grp
12  rmb 2  overlay offset
14  rmb 2  ptr to grfdrv LSET table
16  rmb 2  vector (1FDE/1FF4)
18  rmb 1  gcursor BFN
19  rmb 2  gcursor offset
1B  rmb 2  max X coord (0-79,0-639)
1D  rmb 2  max Y coord (etc txt/gfx)
1F  rmb 2  BLength
21  rmb 3  grp/offs et for next gpload
24  rmb 2  screen logical start default
26  rmb 2  cpx,cpy defaults
28  rmb 2  szx,szy
2A  rmb 6  reserved

4-2-4
INSIDE OS9 LEVEL II
Windows
Section 2

* Screen Table: ($20 each)

00  S.  rmb 1  sty marker
01  rmb 1  first block # (used flag)
02  rmb 2  screen logical start
04  rmb 1  bytes/row
05  rmb 1  border prn
06  rmb 1  foregnd prn (software border)
07  rmb 1  backgnd prn
08  rmb 8
10  rmb 16 palette regs (00RGBRGB)

* Gfx Table (32 of 18 bytes each) pt'd to by .75-6

00  rmb 1
01  rmb 2  BX of graphics cursor
03  rmb 2  BY
05  rmb 13

* Internal Screen T Ype marker byte:
* User STY =>Mark ...
   FF  FF  current screen
   00  FF  current screen
   05  01  640 two color
   06  02  320 four
   07  03  640 four
   08  04  320 sixteen
   02  85  80 col
   01  86  40 col

* Device Memory:

   rmb V.SCF
1D  V.  rmb 1  0=window, 2=vdg, 4=?? , 6=grfdrv
1E  rmb 1
1F  rmb 2  parity, baud (also char temp)
21  rmb 1  case flag
22  rmb 1  keysns enable
23  rmb 1  screen change flag
24  rmb 2  keybd ssig id,signal
26  rmb 2  mouse ssig id,signal
   SS.Mouse (X):
28  rmb 1  init'd to $00 mouse sample rate
29  rmb 1  init'd to $78 mouse fire timeout
   SS.Mouse (Y):
2A  rmb 1  mouse to use
2B  rmb 1  ""
2C  rmb 1  parm cnt
2D  rmb 2  parm vector
2F  rmb 2  ptr to parms start
31  rmb 2  ptr to next parm storage
33  rmb 1  last char read buff offset
34  rmb 1  next char read

4-2-5
INSIDE OS9 LEVEL II
Windows
Section 2

35  rmb 1 window entry number
36  rmb 1 dwnum from descriptor
37  rmb 1 internal comod call number
38  rmb x parm storage
51  rmb x
80  rmb $80 read buffer

* -----------------------------------------------------------------------
* Device Descriptor:

2C  DXSz  rmb 1 SZX
2D  DYSz  rmb 1 SYZ
2E  DWNum  rmb 1 window number
2F  DWIni  rmb 1 0=no defaults, 1=use defaults
30  DSTyp  rmb 1 STY
31  DXPos  rmb 1 CPX
32  DYPs  rmb 1 CPY
33  DFCol  rmb 1 Foregnd PRN
34  DBCol  rmb 1 Backgnd PRN
35  DBord  rmb 1 Border PRN

* -----------------------------------------------------------------------
* Get/Put Buffer Header ($20 each?):

00  B.Block  rmb 1 block link
01  B.Offset  rmb 2 offset in block
03  B.Grp  rmb 1 group number
04  B.Bfn  rmb 1 buffer number
05  B.Len  rmb 2 BL length
07  B.XDots  rmb 2 # x dots in char
09  B.YDots  rmb 2 # y dots in char
0B  B.RowsC  rmb 1 # rows in char
0C  rmb 1
0D  rmb 1
0E  B.STyp  rmb 1 sty marker byte
0F  B.BlkSiz  rmb 1 number of blocks
10  rmb $10 reserved
20  ... data

* -----------------------------------------------------------------------
* Internal GrfDrv Call Numbers (from Grfint)

<table>
<thead>
<tr>
<th>#</th>
<th>What</th>
<th>Escape</th>
<th>#</th>
<th>What</th>
<th>Escape</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Init</td>
<td>2C</td>
<td>2C</td>
<td>DEFGB</td>
<td>29</td>
</tr>
<tr>
<td>02</td>
<td>Terminate</td>
<td>2E</td>
<td>2A</td>
<td>KILLBUF</td>
<td>30</td>
</tr>
<tr>
<td>04</td>
<td>DWSET</td>
<td>30</td>
<td>2B</td>
<td>GPLOAD</td>
<td>Move buffer</td>
</tr>
<tr>
<td>06</td>
<td>DWPRTSWM</td>
<td>32</td>
<td>3C</td>
<td>GETBLK</td>
<td>2C</td>
</tr>
<tr>
<td>08</td>
<td>DWEND</td>
<td>24</td>
<td>2D</td>
<td>FUTBLK</td>
<td>36</td>
</tr>
<tr>
<td>0A</td>
<td>OWSET</td>
<td>22</td>
<td>38</td>
<td>Map GP Buffer</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>OWEND</td>
<td>23</td>
<td>3A</td>
<td>Alpha put</td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>CWAREA</td>
<td>21</td>
<td>3C</td>
<td>Control codes</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SELECT</td>
<td>2E</td>
<td>3E</td>
<td>05 xx cursor calls</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>PSET</td>
<td>34</td>
<td>40</td>
<td>1F codes</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>BORDER</td>
<td>31</td>
<td>42</td>
<td>Goto xy</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>PALET</td>
<td>32</td>
<td>44</td>
<td>FUTCRC</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>FONT</td>
<td>34</td>
<td>4E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>GCSET</td>
<td>39</td>
<td></td>
<td>Set Window</td>
<td></td>
</tr>
</tbody>
</table>

4-2-6
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1C</td>
<td>DEFCOLR</td>
<td>30</td>
</tr>
<tr>
<td>1E</td>
<td>LSET</td>
<td>2F</td>
</tr>
<tr>
<td>20</td>
<td>FCOLOR</td>
<td>32</td>
</tr>
<tr>
<td>22</td>
<td>BCOLOR</td>
<td>33</td>
</tr>
<tr>
<td>24</td>
<td>TCHRSH</td>
<td>3C</td>
</tr>
<tr>
<td>26</td>
<td>PROPSW</td>
<td>3F</td>
</tr>
<tr>
<td>28</td>
<td>SCALE</td>
<td>35</td>
</tr>
<tr>
<td>2A</td>
<td>BOLD</td>
<td>3D</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>POINT</td>
<td>42, 43</td>
</tr>
<tr>
<td>4A</td>
<td>LINE</td>
<td>44-47</td>
</tr>
<tr>
<td>4C</td>
<td>BOX</td>
<td>48, 49</td>
</tr>
<tr>
<td>4E</td>
<td>BAR</td>
<td>4A, 4B</td>
</tr>
<tr>
<td>50</td>
<td>CIRCLE</td>
<td>50</td>
</tr>
<tr>
<td>52</td>
<td>ELLIPS</td>
<td>51</td>
</tr>
<tr>
<td>54</td>
<td>ARC</td>
<td>52, 53</td>
</tr>
<tr>
<td>56</td>
<td>FFILL</td>
<td>4F</td>
</tr>
</tbody>
</table>

*-----------------------------*
CHARACTER FONTS -
by Chris Babcock

Each font has a maximum size of 400 bytes.

The first 100 bytes are broken up and scattered around in the area $80$ to $FF$.

The next 300 bytes contain the definitions for the area $20$ to $7F$.

Each character is represented by 8 bytes. If the bit is 1 the pixel will be set and if it is 0 the pixel will not be set (as you would expect.) The graphic mode is always interpreted as mode five for the fonts.

The font color is the foreground palette. This means the font can not be more than two colors, the foreground palette and the background palette for the on/off conditions of the bits.

A font always uses exactly 8 scan lines per character row. The number of pixels across per character can be either 6 or 8. Using a size of six allows up to 53 characters across in 40 column graphic windows and 106 in 80 column graphic windows. Eight pixels allow 40 or 80 in the corresponding graphic windows.

The following is the breakup of the file:

<table>
<thead>
<tr>
<th>Position in file</th>
<th>Character codes represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000$ - $000C$</td>
<td>$C1$-$DA$ and $E1$-$FA$ stored here</td>
</tr>
<tr>
<td>$000D$ - $00FF$</td>
<td>$AA$-$AF$ and $BA$-$BF$ stored here</td>
</tr>
<tr>
<td>$0100$ - $03FF$</td>
<td>$20$-$7F$ stored in this area</td>
</tr>
<tr>
<td>$0170$ - $0177$</td>
<td>$A0$-$A9$ $B0$-$B9$ $C0$ $DB$ $E0$ $FB$-$FF$</td>
</tr>
</tbody>
</table>

Note: All the above reference $2E$ ('.')

Proportional spacing uses a different method of putting characters on the screen. The 8 bytes are checked to find the range of bits used. Then a blank bit is added to the range at the end. This range is used as the character. The driver is not smart enough to do a proper backspace; it always uses a backspace of the number of pixels selected when the buffer was loaded. A text graphic example of this is below using the word "Mistake."
INSIDE 0S9 LEVEL II
Windows
Section 3

Normal:

```
76543210765432107654321076543210765432107654321076543210
X  X     X
XX XX   X     X
X X X   XXXX XXXX XXX X X XXXX
X X X   X     X X X X X X X X
X X x    X XXXX X  X X XX XXXX
X X X   X     X X X X X X X X
X X X   XXXX X     XXX X X X XXXX
```

Proportional:

```
7654321076543210765432107654321076543210765432107654321076543210
X  X     X
XX XX   X     X
X X X   XXXX XXXX XXX X X XXXX
X X X   X     X X X X X X X X
X X X   XXXX X  X X XX XXXX
X X X   X     X X X X X X X X
X X X   XXXX X     XXX X X X XXXX
```

The transparent character option causes only the set bits to be placed on the screen. Bits already set are not removed from the screen as they would be without this option selected. Using this mode allows the text to overlay graphics on the screen without erasing the character block area.

If moving the cursor, change to fonts you're going to use before moving, otherwise the cursor ends up one line down. Unless you're going from 6-6 or 8-8, then okay.

Note that fonts don't have to be real text. You could for example, set up a font of small objects. The ROGUE game uses special fonts to represent people, gold, trapdoors, etc.
00001 nam Window Descriptor - CC3 LII
00002 ttl INSIDE OS9 LEVEL II
00003 * SRC for /W1-W9
00004 * roll your own descriptors
00005 * 1st version -24Jan87
00006 * Copyright 1987 by Kevin Darling
00007
00008 * -----------------------------------------------
00009 * Change these to make a new /Wx descriptor:
00010 * (only "window" need really be changed)
00011 * For Window numbers great than 9, you must
00012 * manually set the dnam at the end.
00013 * The following is just a sample...
00014
00015 0001 window set 1 the window number
00016 0000 cpx set 0 begin col
00017 0000 cpy set 0 being row
00018 001B cols set 27 number cols
00019 000B rows set 11 number rows
00020 0001 mode set 1 (1=40 col text, 2=80 col text,
00021 0002 fore set 2 foregnd and cursor palettes
00022 0000 back set 0 backgnd palette
00023 0004 bord set 4 border palette
00024
00025 * cols should be <= the mode maximum.
00026 * fore+8 is actual foregnd palette, fore is cursor.
00027
00028 * -----------------------------------------------
00029 00F0 devic equ $F0 quicker than defsfie
00030 0001 objct equ $01
00031 0080 reent equ $80
00032 0001 READ. equ $01
00033 0002 WRIT. equ $02
00034 0000 DT.SCF equ 0
00035
00036 0000 87CD0044 mod len,dnam,devic+objct,reent+1,mgr,drv
00037
00038 000D 03 fcb READ.+WRIT. device mode
00039 000E 07 fcb $07
00040 000F FFA1 fdb $FFA0+window port address
00041 0011 1A fcb opts-*1 option byte count
00042 0012 00 fcb DT.SCF device type
00043
00044 0013 00 fcb 0 case=upper and lower
00045 0014 01 fcb 1 backspace mode
00046 0015 00 fcb 0 delete mode
00047 0016 01 fcb 1 echo on
00048 0017 01 fcb 1 auto line feed on
00049 0018 00 fcb 0 no nulls after cr
00050 0019 00 fcb 0 no pause
00051 001A 18 fcb 24 lines per page default (MW)
00052 001B 08 fcb $08 backspace char
00053 001C 18 fcb $18 delete line char
00054 001D 0D fcb $0D end of record char
00055 001E 1B fcb $1B end of file char
00056 001F 04 fcb $04 reprint line char

4-4-1
INSIDE OS9 LEVEL II
Windows
Section 4

00057 0020 01  fcb $01  dup last line char
00058 0021 17  fcb $17  pause char
00059 0022 03  fcb $03  interrupt character
00060 0023 05  fcb $05  quit character
00061 0024 08  fcb $08  backspace echo char
00062 0025 07  fcb $07  line overflow char
00063 0026 80  fcb $80  type=window
00064 0027 00  fcb $00  baud
00065 0028 0036  fdb dnam  echo device
00066 002A 00  fcb $00  xon character
00067 002B 00  fcb $00  xoff character
00068 002C 00  opts equ $  End of Path Desc Options
00069
00070 002C 1B  fcb cols
00071 002D 0B  fcb rows
00072 002E 01  fcb window  window #
00073 002F 01  fcb l  use defaults option
00074 0030 01  fcb mode
00075 0031 00  fcb cpx
00076 0032 00  fcb cpy
00077 0033 02  fcb fore  fgrnd and cursor palette
00078 0034 00  fcb back  backgnd palette
00079 0035 04  fcb bord  border palette
00080
00081 0036 576E  dnam fcc "Wn"
00082 0038 B1  fcb SBO+window
00083 0039 5343C6  mgr fcs "SCF"  file manager
00084 003C 43433349  drv fcs "CC310"  driver
00085
00086 0041 3EF9CA  emod
00087 0044 0000  len equ *
00088
0000 error(s)
0000 warning(s)
$0044 00068 program bytes generated
$0000 00000 data bytes allocated
$0160 00352 bytes used for symbols

0000 S BACK  0004 S BORD  001B S COLS  0000 S CPX  0000 S CPY
00F0 E DEVIC  0036 L DNM  003C L DRV  0000 E DT.SCF  0002 S FORE
0044 E LEN  0039 L MGR  0001 S MODS  0001 E OBJCT  002C E OPTS
0001 E READ.  0080 E REENT  000B S ROWS  0001 S WINDOW  0002 E WRIT.

=====================================================
These are the Tandy-supplied options:
(in same order as descriptor)

```
<table>
<thead>
<tr>
<th>OPTION</th>
<th>W</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
<th>W6</th>
<th>W7</th>
</tr>
</thead>
<tbody>
<tr>
<td>cols</td>
<td>00</td>
<td>1B</td>
<td>OC</td>
<td>28</td>
<td>3C</td>
<td>13</td>
<td>5D</td>
<td>50</td>
</tr>
<tr>
<td>rows</td>
<td>00</td>
<td>0B</td>
<td>0B</td>
<td>0C</td>
<td>0B</td>
<td>0B</td>
<td>0C</td>
<td>18</td>
</tr>
<tr>
<td>wind</td>
<td>FF</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
</tr>
<tr>
<td>deflt</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>mode</td>
<td>00</td>
<td>01</td>
<td>FF</td>
<td>FF</td>
<td>02</td>
<td>FF</td>
<td>FF</td>
<td>02</td>
</tr>
<tr>
<td>cpix</td>
<td>00</td>
<td>00</td>
<td>1C</td>
<td>00</td>
<td>00</td>
<td>3D</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>cpy</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0C</td>
<td>00</td>
<td>00</td>
<td>0C</td>
<td>00</td>
</tr>
<tr>
<td>fore</td>
<td>00</td>
<td>02</td>
<td>00</td>
<td>02</td>
<td>00</td>
<td>02</td>
<td>02</td>
<td>00</td>
</tr>
<tr>
<td>back</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>07</td>
<td>01</td>
<td>07</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>bord</td>
<td>00</td>
<td>04</td>
<td>01</td>
<td>01</td>
<td>04</td>
<td>04</td>
<td>04</td>
<td>01</td>
</tr>
</tbody>
</table>
```

Note that a descriptor with TYPE=1 is a VDG window instead of these (TYPE=80).
INSIDE OS9 LEVEL II

Miscellaneous
SHELL

INFORMATION

CoCo-3 Level Two has a new shell, derived from the original that was used before for both L-I and L-II systems. The changes made were done mostly because of windows and our 8K blocks.

To the user, there are four main new features:

. The ability to redirect multiple paths to the same file, using the <<, >>>, <<<, >>> options.

. The usage of a path number as a device reference: that is, you can redirect a command's standard input, output or error to the current in/out/err paths. To do this, you use the pseudo device names "/0, /1, or /2".

The main use that you'll see of this is inside shell script files. An example should be in your Startup file, where you'll find "setime <1" instead of "setime <term" like you're used to seeing. Since path 1 (standard output) is still the device that you're viewing, the effect is the same, but now the same Setime script will also work with say, an external terminal. This feature gives you more flexibility and less hard-coding of device names.

. The "i=/devicename" option. This is known as the immortal option. What it does is open all three standard paths to the device named, and sets a flag in the shell's data area.

The flag indicates that the shell should not end operations on an End-of-File. This is needed because CC3GO would have no idea where to restart a shell, unlike the older SysGo which could pretty well assume /TERM.

This also provides a quick and dirty tsmon-like way to use an external terminal without it dying on you. Just use something like "shell i=/T2 &" to keep a shell on /T2. You could also have done "shell <<</t2", but that one could die on an EOF.

A related new feature is that if a new shell starts up but gets back an error printing "Shell", then it does die. This might happen if you start a shell and the open-window call fails. The reason is to keep from having phantom shells laying about with no paths open... they'd be impossible to kill.

. The ability to send special shell characters as parameters. Before, if you tried an:  echo hello! , the shell would send 'hello' (without the quotes) to echo, but then take the '!' and try to pipe to the next command, which wasn't there of course.

Now, you can type:  echo "hello!!" , and what echo gets and prints out is: "hello!!", but including the quote marks, unfortunately.

A SMALL PROBLEM

As seen in the flowchart, if the shell can't find a program in memory, it tries reading it's header from the current execution directory. If that fails, it tries to use a file from the data directory as a shell script for a new shell.
The older shells would first F$Link a module into it's own map to get the header information needed for a FSFork of the new process. Unfortunately, with our 8K blocks, it's possible that this link might fail because the new program was too large to fit in the blocks left in a shell's map (normally 5 under ver 2.00.01).

The new L-II shell uses two new OS9 system calls to get around this: F$NMLink and F$NMLoad, both of which do NOT link a module into the caller's map, but instead just return some information from the module's header (like Data Size).

To keep the module link count straight, the shell also does an F$UnLoad, which uses a module's NAME to call unlink.

This is fine. A minor problem can occur, though, if the name of the module that shell wants to unload differs from the module's real name. This can happen if, for example, you had the Ident command on your disk under the filename "ld". What would happen is that when you typed "id", the shell would end up F$NMLoading Ident from your commands directory and executing it. This is normal. But then shell would try to Unload "id", as that's the name it saved from the command line.

The net effect is that Ident would stay linked in the module directory until you manually unlinked it.

Another way this could occur is if you used a partial or full pathname. Examples: "/d1/cmds/bob" or "/.../bob". In neither case will the F$UnLoad call work since those "names" do not match any in memory.

As I said, this is minor, and the shell can be rewritten someday to also read in the real name after it reads the header from disk. I suspect a later version will have this. The point is that you should be aware of this and so not be surprised.

KILLING WINDOW PROCESSES

While we're on the shell, I want to bring up another "gotcha" that makes perfect OS9 sense, but that still took a while to figure out.

Let's say that you began with a shell on TERM. Then you started one on W2 with "shell i=/w1&" and you went over to that one. Now you start another one with "shell i=/w7&" and then moved back to the original TERM window.

There let's say that you kill the shell on W7. You do a Procs and that shell continues to show up with an error 228.

The "gotcha" is that the shell on W1 was the parent of the dead W7 shell, and until you go to W1 and hit a key, the dead shell can't get thru to W1 to report it's death.

A similar thing can bite you worse. If you had started a process on W7 using the same method and it dies while you're doing something important (like editing a file) on the parent's window (W1), then you'll be confused by the death message popping up in the middle of your session.
Now this quirk has been around OS9 forever, but unless you used a lot of terminals, it didn't matter too much. With many windows now, it becomes more important and aggravating.

The partial solution that I use is to always start all my shells on other windows from my first window. That way, I at least know where their deaths will show up (-005 etc). This would go for any program I wanted to run in the background mostly unseen (using "&").

Typing "w" <enter> on the parent shell's window after killing a child is another good idea, as that causes that shell to Wait for the death report without messing up your screen.

MISC

Just wanted to add a couple of things about the shell that don't seem to be well-documented.

Many people falsely assume that "OS9" recognizes that a module is, say, a Basic09 packed I-Code procedure and so "OS9" calls up RUNB to execute it. The truth is that this is all done by the Shell. Trying to fork an I-Code module from a machine language program would fail unless you yourself specified the module as a parameter to RUNB and forked RUNB.

The other small point is that using parenthesis starts a sub-shell. For example, the command " (((echo hi; sleep 500))" would cause 3 sub-shells to be formed, each calling the next. Try this sometime with a Procs command running on another window so you can see all the shells formed.
INSIDE OS9 LEVEL II
Miscellaneous
Section 1

### SHELL

Data Area:
redirected pths

Data Area:
#pages

Data Area:
pathname ptr

Data Area:
parm size

Data Area:
parm size

Data Area:
parm ptr

Data Area:
mem for mod

Data Area:
mem for mod

Data Area:
this char

Data Area:
'(' count

Data Area:
signal storage

* DOCMD SUB *

Exec W, *, CHD, CHX, EX, KILL, X, P, T, SETPR, ;

Find ()'s

Exec &, ! ; # < > >>

Start Process

Undo redirection

Wait if required

RTS

* START PROCESS *

Link to name err-------->

Unlink

Open xfile err-------->

Read hdr

Close file

<y (M/L code?)

Else find lang (Runb, PascalS) < name'

Cmd= 'Shell

Link to cmd/language

Load if necessary

Set mem size

F$fork

F$sleep 1

F$unload cmd name

RTS

5-1-4
This section is not really needed any more, as L-II will be out by the time this gets published. However, for those those who are getting started with L-II by way of the Tandy game disk "Rogue" cat # 26-3297,

**USING Rogue TO MAKE A SYSTEM DISK:**

1- under L-I, format a disk.
2- os9gen that disk using the OS9boot file on Rogue *
3- copy over CMDS dir with grfdrv and shell **.
4- drop back to RSDOS and copy over the L-II kernal with:

```
5 REM Rogue in drive 0, new disk in drive 1.
10 CLEAR 10000
20 FOR SE = 1 TO 18
30 DSEIS 0,34,SE,A$,B$
40 DSKOS 1,34,SE,A$,B$
50 NEXT SE
```

* LR Tech owners may include their driver and desc after copying the new "shell" file and "grfdrv" to it, OR after changing the desc name from "H0" to something else so that the bootup gets shell/grfdrv from the floppy. Then CHX /H0/CMDS.

You should also change the H0 desc byte at $0E from $FF to $07 and reverify that module. That's the extended device address.

** You may include other utilities merged into the Rogue shell file (do an ident on it first!), to be included at startup. The total length of your shell file should be under $1E00 long.

You MUST have Grfdrv and Shell in your CMDS dir. They must also have the "e" attribute set on the files.

Since L-II will map in the entire block of cmdos loaded in a file, you should try to keep things on an n*8K+(8K-512) boundary.

Your L-I mfree, mdir, and procs will NOT work.

PRINTER will work if you change the baud rate to 1/2 before.

One other thing: do NOT unlink Shell in memory. Crash-o!

**MAKING WINDOWS:**

Examples are also in Rogue's MAKE40, MAKE80, MAKEGW shell files.

However, because Rogue does not include the W, and W1-W7 device descriptors, you cannot make more than one window or screen of windows with it. Solution: make a set of window descriptors using the source code elsewhere in this text.
INSIDE OS9 LEVEL II
Miscellaneous
Section 2

Don't worry too much about the default size and palettes, you can send the escape codes to override them anyway. Example:

\`iniz w1 (if you have iniz cmd)
\`display lb 20 2 0 0 30 c 9 0 1 >/w1
\`shell i=/w1 &
\`
(now hit the CLEAR key: you should flip to that screen)

Read the Sept 86 RAINBOW article on windows, plus try out the later examples they give if you have 512K.

[]

Be aware that your CLEAR and @ keys are no longer the same as the CTRL and ALT keys!
BUGS - SOFTWARE

Level Two for the CoCo-3 has gone through many revisions, and most of the bugs have been ironed out over the months. What are left in version 2.00.01 are relatively minor. Not all are listed here. Check the electronic forums for recent updates.

MODULE: Clock
PROBLEM: Bad error code return.
SPECIFICS: Somebody left the '#' sign off of a LDB #E$error.

SOLUTION: Patch and reverify.

Offset Old New
0191 D6 C6

MODULE: IOMan
PROBLEM: Sorts queues wrong.
SPECIFICS: Change first made in L-I 2.0 to insert processes in I/O queues according to priority. Used wrong register.

SOLUTION: Patch and reverify.

Offset Old New
09A6 10 12
09A7 A3 E1

MODULE: GrfDrv
PROBLEM: Non-efficient use of screen memory.
SPECIFICS: Opening a 40 column screen should use the last 2K of an 8K screen block if it's free for use. However, apparently a bad Def was used in MW's source code and GrfDrv cannot match an internal code as a 40 column screen.

SOLUTION: Patch and reverify.

Offset Old New
033A 84 86

MODULE: IOMan
PROBLEM: Cannot have more than one VIRQ device at a time.
SPECIFICS: While Clock gets the size of the VIRQ table from the Init module (as it should), IOMan has a different size hard-coded in. Clock inserts the first entry at the front of the VIRQ table, but the next call starts searching at the end of the table...which turns out to usually be the header of the first module in your bootfile. Symptoms: if your disk drive is still going (waiting for motor time-out), you cannot Iniz a ModPak device. Or, if you Iniz a ModPak device, your drives will never shut off.

SOLUTION: Easiest patch is to the INIT Module, to change the number of IRQ/VIRQ devices down from 15 to say, 12.

Offset Old New
000C 0F 0C
MODULE: CC3IO
PROBLEM: SS.Monr getstat call bad.
SPECIFICS: Although the manual doesn't mention it, CC3IO also supports getting the current monitor type set by montype. The value (0,1,2) is returned in the X register. The code in CC3IO should have been a STD R$X instead of STB R$X though.

SOLUTION: Patch and reverify.

Offset Old New
 07D2  E7  ED

BUGS - HARDWARE

The GIME chip itself, on many machines, has problems with map changes causing "snow" on the screen, horizontal scrolling difficulties, and a few other items.

The most basic problem is one of bus-timing, and a fix is expected soon from Tandy. This is all I can say right now.

The Speech/Sound Cartridge, because it uses the clock signals generated from the 6809E, is driven too fast at the 2MHz speed of L-II to operate correctly. This is also true of several third-party interfaces and ramdisk paks.

Information on hacking the SSC can be had on the electronic forums. Users of other gear should contact their suppliers for updates or patches to their hardware.

Many of us with the original Tandy floppy disk controllers have found that they simply cannot handle the 2Mhz speed. There are two things you can do about this.

You can try replacing the Floppy Disk Controller chip or data separator chips, and hope you bought a faster part than before. Or you can opt for one of the third-party controllers.

Both Disto and J&M controllers seem to work fine so far. The newer, the better, seems to be the rule of thumb.

As far as hard disk set-ups go, the ones at this time that I know will work at 2MHz is the LR Tech from Owlware, FHL's QT CoCo, and perhaps the J&M.
At the last moment before this went to press, several people with Level Two called to ask about some mistakes in the manual. I won't point out the ones like misspellings, just the ones that might confuse you.

SUBJECT: Creating GFX Windows
SECTION: BASIC09 Reference
PAGE: 9-37

Here they tell you how to create a graphics window, but show the "merge sys/stdfonts >/w1" AFTER the wcreate. Nope. All you get is dots on the screen. You must merge stdfonts BEFORE opening any gfx windows, unless you care to do a FONT command to that window after merging. They had it correctly on the page before (9-35) about merging so that you can type later.

SUBJECT: F$FORK, F$LINK, F$LOAD, I$CREATE, I$MAKDIR, I$OPEN
SECTION: OS9 Tech Reference
PAGE: 8-16, 8-23, 8-26, 8-49, 8-56, 8-58

On all of these, after the call X should be pointing to the $0D (carriage return) at the end of the string.

SUBJECT: F$FORK
SECTION: OS9 Tech Reference
PAGE: 8-15

The Y register contains the parameter area size in BYTES, not in pages.

SUBJECT: F$TIME
SECTION: OS9 Tech Reference
PAGE: 8-40

To be exact, on exit X points to the time packet returned to the area at (X) that you had originally passed for the call.

SUBJECT: I$DELETE
SECTION: OS9 Tech Reference
PAGE: 8-50

On return, X should be pointing to the beginning of "MEMO".
INSIDE OS9 LEVEL II
Miscellaneous
Section 3

SUBJECT: F$ALARM
SECTION: OS9 Tech Reference
PAGE: 8-66

F$Alarm is a user call, too. And they left out how to use it. Here's the info:

This call has several variations, which have to do with setting time variables that the Clock module will try to match once a second. You may clear the alarm setting, read it, or set it for one of two exclusive actions.

D = 0000 : clear the setting

X = ptr to 5-byte time packet (YYMMDDDHHMM)
D = 0001 : cause the CC3IO "beep" for 16 seconds after the time packet sent matches system time.

X = ptr to spot for time packet return
D = 0002
X < current alarm setting packet returned
D < current proc id and signal pending

X = ptr to 5-byte time packet (YYMMDDDHHMM)
A = proc id to signal on time match
B = signal to send on time match

SUBJECT: F$DATLOG
SECTION: OS9 Tech Reference
PAGE: 8-78

Actually, not a bad example, but only if you're running on a machine with 4K blocks. On the CoCo-3, Output X = $4329. The actual code just multiplies B*$2000 and adds it to X.

SUBJECT: SS.RDY
SECTION: OS9 Tech Reference
PAGE: 8-113

On devices that support it, the B register will return the number of characters that are ready to be read. Both CC3IO and ACIAPAK support this feature.

SUBJECT: SS.MOUSE
SECTION: OS9 Tech Reference
PAGE: 8-125 on

Somebody forgot the two reserved bytes between Pt.ToTm and Pt.TTTo. As printed, offsets after ToTm are wrong. So insert a "rmb 2 - reserved" after Pt.ToTm. Also ignore the system use note at the end after Pt.Siz.
Also, if you specify screen number zero (Y=0000), then you will return to the normal VDG (32x16) screen. This should be done before a SS.FScrn if you wish to return to a text screen.

This is such a great book that the minor errors can be explained by the authors desire to get the information out to you quickly. You should send them lots of money and good wishes. By the way, this portion of the book is being written very close to April 1st.

PS The word 'them' in the second sentence should be changed to FHL.

PPS Remember it's real close to April 1st.
FONT CONVERSION

This is an RSDOS program from Chris Babcock that converts Graphicom-II font files to the format required by OS9. After conversion, you must copy the file over to an OS9 disk.

You must also specify the group/buffer numbers that you will later use to access the font using the FONT commands. We've been personally using group D0, and buffers 1-8 or so.

10 CLEAR 500,&H7800:POKE&H95C9,&H17:POKE&HFF22,PEEK(&HFF22)OR&H10:CLS:PRINT"Graphicom II Font to OS-9 Font  Copyright 1987 by Chris babcock - Program for Coco 3"
20 DATA 141,83,134,27,141,59,134,43,141,55,182,14,0,141,50,182,14,1,141,45,134,5,141,41,204,0,8,141,46,141,44,204,4,0,141,39,79,16,142,1,0,141,22,49,63,38,250,142,124,0,16,142,3,0,236,129,141,17,49,62,38
30 DATA 248,126,164,45,141,28,38,3,126,206,217,126,207,181,141,18,38,3,126,206,215,126,207,179,141,8,38,3,126,201,86,126,202,4,52,2,182,193,66,129,48,53,130
40 FOR I=&HE04 TO &HE04+103:READ DT:POKE I,DT:NEXT
50 PRINT"What is the filename of the font (Maximum 8 Chars. Ext is "+CHR$(34)+"SET"+CHR$(34)+")":PRINT"Use #:FILENAME if other drive."
60 LINEINPUT ";:F$:PRINT@235,".SET"+CHR$(13):F$:LEFT$(F$,10)+".SET" 
70 PRINT"New filename for the font (Maximum 8 Chars. Ext is "+CHR$(34)+"OS9"+CHR$(34)+")":PRINT"Do NOT enter a drive # now."
80 LINEINPUT ";:G$:PRINT@393,.OS9":G$:LEFT$(G$,8):G$:G$+STRINGS$(8-LEN(G$),32)+"OS9"
90 INPUT"Drive number for OS-9 file";D
100 LOADM "FS"
110 CLS:PRINT"Group number for the OS-9 Font (Give in hexadecimal 00-FF)";LINEINPUT ";:GR$"
120 GR=VAL("&H"+GR$):IF GR<0 OR GR>255 THEN 110
130 PRINT"Buffer/Font number (Hex also)";LINEINPUT ";:BF$"
140 BF=VAL("&H"+BF$):IF BF<0 OR BF>255 THEN PRINT@96,"":GOTO 130
150 POKE&HE0:D:POKE&H95A,D  
160 POKE&HE00,GR:POKE&HE01,BF
170 X=&H94C:FOR I=1 TO 11:POKE X,ASC(MID$(G$,I,1)):X=X+1:NEXT I:POKE&H957,1:POKE&H958,0
180 PRINT"Saving..." 
190 EXEC&D:HE04
200 CLS:PRINT"Use XCOPY or TRSCOPY to move the file over to an OS-9 Level II disk. MERGE the file and type DTSPLAY 1B 3A GROUP BUFFER <cr>" 
210 END
TIPS, GOTCHAS, and LAST MINUTE STUFF

Using L-I VDG Programs

Many of you may want to run programs such as TSEDIT or Steve Bjork's bouncing ball demo within a L-II screen. Fortunately, Microwave provided for this. However, your disk only comes with one VDG-type descriptor, TERM-VDG.

For programs that don't have "/TERM" hard-coded in them, you can set up a window device as a VDG screen using the following method (where wx= any window number):

deiniz wX
xmode /wx type=1 pag=16
shell i=/wX &

This will give you another screen that you can flip to, where you can run TSEDIT or other older programs.

OS9Boots

Under L-I, many of us only loaded drivers and other modules as needed, to save memory. Level Two acts a bit differently, and your methods must change.

You should put ANY and ALL drivers and descriptors that you plan to use, IN your OS9Boot file. If you don't, then each time you load a separate driver, you will take up 8K of your 64K system map... doesn't take more than a couple to really limit the number of tasks or open files that you can have.

When using OS9Gen or Cobbler to make a new boot disk, be sure that you have a CMDS directory with a Shell file and the GrfDrv module. The execution attributes should also be set on these two files. Otherwise, you'll get the dreaded "OS9BOOT FAILED".

Merged Module Files:

If you ident your /DO/CMDS/shell, you'll see that more than one command is included in that file. The reason is that it pays to get as close to an 8K block boundary as possible, so that you use less memory. If you separately loaded each of those commands, each would take an 8K block. Even with 512K, you'd lose memory very quickly.

OS9 will try to fit a block of modules into the upper part of a 64K task map... but remember that the FEXX page and our I/O is from FE00-FFFF in all maps. So the ideal size of a merged file is:

(8K * N) - 512 bytes, where N ranges from 1-7

Actually, N should be kept around 1, if possible. So a Shell file for instance, should ideally be just under $1E00 long. That's (8K * 1)-512 = $2000-$200 = $1E00.

RUNB is 12K, so it takes up 2 blocks, but you still have room for about 5K of things like syscall, inkey, gfx2, etc.
To create a new shell file, for example, you might do:

```
merge shell dir free mdir procs ... etc >newshell
rename shell shell.-old; rename newshell shell
attr shell e pe
```

A "dir e" can tell you the size of merged files or you can print out an Ident of all your commands and use that as a reference to calculate from.

**F$Load from system state:**

Requires an extra parameter if done from a driver or other module that will be in the system map. The U register must point to the process descriptor of the process who's map you want the new module loaded into. Example for loading module file into the system space:

```
leax modnam,pc point to name of module to load
ldd D.SysProc get system proc desc pointer
OS9 "$F$Load load file "modnam" into system map
```

**F$Link from system state:**

Will put the module into the map of the current process (D.Proc). It also gets the name (X points to it) from the D.Proc map. So to link a module into system space, you must "trick" OS9:

```
ldd D.Proc get current process
pshs d save it
ldd D.SysProc get system proc desc
std D.Proc make it current proc temporarily
... (set up link parms)
OS9 F$Link link module(s) into system map
puls d retrieve true user process
std D.Proc and reset as current process
```

**Forking RUNB modules:**

Pete Lyall and I just figured this one out, and even though it's fully explainable, it's still a gotcha...

Let's say that you have a Basic09 I-code (packed) module named "Bob", and it requires 10K of data area. Typing "bob" from the shell command line causes shell to check Bob's header. There it finds that Bob needs 10K and also needs RUNB. So the shell effectively does a 'runb bob #10k'. Fine.

But! If you have the need to fork "RUNB BOB" from within a m/l program and don't know what data size Bob (or any I-code module) needs, you'll probably try just using a F$Fork RUNB with Bob as a parameter - which will fail because RUNB's header only has a default data size required of 4K (possibly 8K for CoCo-3). And 4K isn't enough for Runb to use Bob.

(note: just doing a "runb bob" from the shell cmd line would fail, too)
Moral is that you should either check an I-code’s header yourself, or you could instead do a "FS$Fork Shell bob" and let shell handle everything.

Using L-II Debug on Level Two:

There is no debug included on the L-II disk set. It will be on the Developer’s Pak disk. In the meantime, if you can’t use Modpatch for what you need to do, you can partially patch your current debug to at least let you modify modules in memory.

Debug will link to a module, but does so just to get the module address. It immediately unlinks the same module to keep the system link count correct. Under L-II, this means that the module is mapped into debug’s space, then mapped out right after that.

As debug is now, you CAN use it on any modules that were in your bootfile, but that’s because those cannot be unlinked. To debug other loaded modules, you have to change debug while under Level ONE:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>06CC</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>06CD</td>
<td>3F</td>
<td>12</td>
</tr>
<tr>
<td>06CE</td>
<td>02</td>
<td>12</td>
</tr>
<tr>
<td>06D0</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>06D1</td>
<td>3F</td>
<td>12</td>
</tr>
<tr>
<td>06D2</td>
<td>02</td>
<td>12</td>
</tr>
</tbody>
</table>

Then save it and reverify, of course. The only gotcha now is that since modules are not unlinked at all, then if you try debugging all sorts of modules at one time, you could get an error #207 from the debug map getting filled up. No problem, just Quit and enter Debug again.

Login II Patch

This patch will allow you to use your level I 'LOGIN' command (which currently crashes on a level II system) on a level II system. It corrects the code so that it uses the FS$suser call instead of trying to manipulate the system’s direct page, which is inaccessible under level II for writing (in USER mode). This patch is a joint effort of Kent Meyers and Pete Lyall.

display c
t
* LOGIN2.DBG - A patch script by Pete Lyall
*
* This is a shell procedure to use DEBUG to patch the LOGIN
* command for use on a Level II OS9 system. Note: If you HAVE
* NOT already patched your DEBUG command for use on a level II
* system then either do THAT first, or run this script on a
* LEVEL I system where DEBUG will work.
* *
* -t
tmode .1 -pause
load login
debug
1 login
INSIDE OS9 LEVEL II
Miscellaneous
Section 5

...+52
=49
=20
=32
l login
...+57
=30
l login
...+5a
=31
l login
...+69
=49
=20
=32
l login
...+6e
=30
l login
...+71
=31
l login
...+234
=1f
=02
=10
=3f
=1c
=12
l login
...+49b
=66
=15
=73
q
save login.II login
display c
t
* The patch is completed.
* * Now simply UNLINK LOGIN until it is out of memory
* * The updated LOGIN command has been saved as 'login.ii' in
* * the current directory.
* * To use it, simply copy it to a LEVEL II disk's CMDS
* * directory and rename it to 'login'. Also ensure that all
* * the attributes are set properly for execution.
* * Enjoy!
INSIDE OS9 LEVEL II

Sources
**INSIDE OS9 LEVEL II**

**SOURCES**

**Alarm**

Microwave OS-9 Assembler RS Version 01.00.00  03/30/87 00:15:04  Page 001

Alarm - INSIDE OS9 LEVEL II

00001    nam Alarm
00002    ttl INSIDE OS9 LEVEL II
00003    * alarm - test that sets alarm for next minute.
00004    * causes beep from coco sound output for 15 secs.
00005    * just for fun.
00006    * Copyright 1987 by Kevin Darling

00007
00008  0006    F$Exit    equ 6
00009  0015    F$Time    equ $15
00010  001E    F$Alarm    equ $1E
00011
00012  0054    D.Time    equ $54
00013  0057    D.Min    equ $57
00014
00015  0000    87CD0026    mod len,name,$11,$81,entry,msize
00016
00017  D 0000    time    rmb 10
00018  D 000A    rmb 200
00019  D 00D2    msize    equ .
00020
00021  000D    416C6172    name fcs "Alarm"
00022  0012    01    fcb 1
00023  0013    entry
00024  0013    30C4    leax time,u
00025  0015    103F15    OS9 F$Time
00026  0018    6C1D    inc D.Time-D.Min,x next minute (bad on 59)
00027  001A    CC0001    ldd #$0001
00028  001D    103F1E    OS9 F$Alarm    set alarm time
00029  0020    103F06    OS9 F$Exit
00030
00031  0023    A9F133    emod
00032  0026    len    equ *
00033  end

00000    error(s)
00000    warning(s)
$0026    00038 program bytes generated
$00D2    00210 data bytes allocated
$00CA    00202 bytes used for symbols

0057    E D.MIN    0054    E D.TIME    0013    L ENTRY    001E    E F$ALARM    0006    E F$EXIT
0015    E F$TIME    0026    E LEN    00D2    E M$SIZE    000D    L NAME    0000    D TIME
DMEM - dmem <block> <offset> [<length>] ! dump
             dmem -<proc#> <offset> [<length>] ! dump

Dmem writes up to $1000 bytes to standard out, that it has copied over for you from other maps. If no length is given, it defaults to 256 ($0100) bytes. Examples using data above:

    dmem  4 0 ! dump : dumps first 256 bytes of GrfDrv
    dmem  2 lCA lAE ! dump : dumps CC3Go
    dmem  0 0 1000 >/dl/file : file contains lower sysmem vars

    dmem -3 0 20 ! dump : dump first 32 shell data bytes
    dmem -3 E000 5FA ! dump : another way of dumping Shell
    dmem -1 0 1000 >/dl/file : file contains lower sysmem vars

Good use of PROC, PMAP, MDIF, and DMEM depends on the data you get from each. Open a graphics window and recheck the MMAP. Kill a Shell, and notice the status and signal codes. Look up the status bits in your old DEFS file, signal from Error codes. Watch how modules get mapped in using PMAP and MDIF.

Figure out system data use by knocking out the blocks you know are in other use, with PMAP and MMAP.
IN Inside Os9 Level II
Sources

DMem

Microware Os-9 Assembler RS Version 01.00.00 03/30/87 00:15:20 Page 001
DMem - Inside Os9 Level II

00002

* DMem - display block/mem offset
00004

* "Dmem blk offset [len]! dump"
00005

* "Dmem #id offset [len]! dump"
00006

* 08feb87 - change page offset to byte or id.
00008

* 22jan87 - version 1
00009

* Copyright 1987 by Kevin Darling
00011

00012

00000 87CD0136

00013

0000D 444D65ED

00014

0011 02

00015

00016

F$Exit equ $06
00017

F$GPrDsc equ $18
00018

F$CpyMem equ $1B
00019

I$Write equ $8A
00020

I$Writeln equ $8C
00021

00022

1000

00023

00024

D 0000

acc  rmb 2
00025

D 0002

input  rmb 1
00026

D 0003

offset  rmb 2
00027

D 0005

dlen  rmb 2
00028

D 0007

id  rmb 1
00029

D 0008

prcdsc  rmb 512
00030

D 0208

buffer  rmb buffsiz
00031

D 1208

stack  rmb 200
00032

D 12D0

msize  equ
00033

00034

0048

dat  equ prcdsc+$40
00035

00036

0012

hexin
00037

0012 0F00

clr  acc
00038

0014 0F01

clr  acc+1
00039

0016

hex01
00040

0016 A680

lda ,x+
00041

0018 8120

cmpa #$20
00042

001A 272A

beq hexrts
00043

001C 810D

cmpa #$0D
00044

001E 2726

beq hexrts
00045

0020 8030

suba #$30
00046

0022 810A

cmpa #$10
00047

0024 2504

bcs hex2 0-9
00048

0026 8407

anda #$7

A-F
00049

0028 8B09

adda #$9

Page 6-2-2
INSIDE O69 LEVEL II

SOURCES

DMem

00049  0028 8B09  adda  #9
00050  002A       hex2
00051  002A  48  asla
00052  002B  48  asla
00053  002C  48  asla
00054  002D  48  asla
00055  002E  9702  sta  input
00056  0030  DC00  ldd  acc  get accumulator
00057  0032  0902  rol  input
00058  0034  59  rolb
00059  0035  49  rolq
00060  0036  0902  rol  input
00061  0038  59  rolb
00062  0039  49  role
00063  003A  0902  rol  input
00064  003C  59  rolb
00065  003D  49  rola
00066  003E  0902  rol  input
00067  0040  59  rolb
00068  0041  49  role
00069  0042  D000  std  acc
00070  0044  20D0  bra  hex01
00071  0046       hexrts
00072  0046  301F  leax  -1,x
00073  0048  DC00  ldd  acc
00074  004A  39  rts
00075   004B       entry
00077  004E  1700DA  lbsr  skipspc  skip leading
00078  004E  102700C7  lbeq  badnum  ..was <cr>
00079  0052  812D  cmpa  "-"  else is it #id ?
00080  0054  2617  bne  entry0  ..no
00081
00082  0056  3001  leax  1,x  yes, skip '-'
00083  0058  8DB8  bsr  hexin  get id number
00084  005A  1F98  tfr  b,a
00085  005C  3410  pshs  x
00086  005E  30C90008  leax  >produc,v
00087  0062  103F18  OS9  $GPrDSc  get that proc desc
00088  W  0065  10250053  lbc6  error
00089  0069  3510  puls  x
00090  006B  2006  bra  entry1
00091
00092  006D       entry0
00093  006D  8DA3  bsr  hexin  get block #
00094  006F  0F48  clr  dat  set in fake datimg
00095  0071  D749  std  dat+1
00096
00097  0073       entry1
00098  0073  1700B2  lbsr  skipspc  get offset
00099  0076  1027009F  lbeq  badnum
00100  W  007A  17FF95  lbsr  hexin
00101  007D  DD03  std  offset
00102

Page 6-2-3
INSIDE OS9 LEVEL II

SOURCES

DMem

00103  007F 1700A6  lbsr skips pc  get possible length
00104  00B2 270E  beg entry2
00105  W 00B4 17FF8B  lbsr hexin
00106  00B7 10B31000  cmpd #$1000
00107  00B8 2308  bls entry3
00108  00B8 CC1000  ldd #$1000
00109  00B9 2003  bra entry3
00110  0092  entry2
00111  0092 CC0100  ldd #$0100
00112  0095  entry3
00113  0095 DD05  std dlen
00114
00115  0097 30C90048  leax >dat,u
00116  009B 1F10  tfr x,d  D=dat image ptr
00117  009D 1090E5  ldy dlen  Y=count
00118  00A0 9E03  ldx offset  X=offset within dat image
00119  00A2 3440  pshs u
00120  00A4 33C90208  leau buffer,u
00121  00A6 103F1E  OS9 F$CopyMem
00122  00A7 3540  puls u
00123  00A9 250D  bcs error
00124
00125  00AF 1090E5  ldy dlen
00126  00B0 30C90208  leax buffer,u  point within buffer
00127  00B6 8601  lda #1
00128  00B8 103F8A  OS9 I$Write
00129  00BB 5F  bye
00130  00BC 5F  clr b
00131  00BC 5F  error
00132  00BC 103F06  OS9 F$Exit
00133
00134  00BF 5573653A  fcc "Use: DMem <block> <offset> [<length>]!"
00136  00EB 0A  fcb $0A
00137  00EC 206F723A  fcc " or: DMem -<id> <offset> [<length>]!"
00138  0118 0D  fcb $0D
00139  005A  helplen  equ *-help
00140  0119  badnum
00141  0119 308DFFA2  leax help,pc
00142  011D 108E005A  ldy #helplen
00143  0121 8602  lda #2
00144  0123 103F08C  OS9 I$Write
00145  0126 2093  bra bye
00146  0128  skips pc
00148  0128 A680  lda ,x+
00149  012A 8120  cmpa #$20
00150  012C 27FA  beq skips pc
00151  012E 301F  leax -1,x
00152  0130 810D  cmpa #$0D
00153  0132 39  rts
INSIDE OS9 LEVEL II

SOURCES

DMem

00154
00155 0133 979412       emod
00156 0136       len    equ    *
00157       end

$0136 00310 program bytes generated
$12D0 04816 data bytes allocated
$0223 00547 bytes used for symbols

0000 D ACC       0119 L BADNUM    0208 D BUFFER    1000 S BUFFSIZ    00BB L BYE
0048 E DAT       0005 D DLEN      004B L ENTRY      006D L ENTRY0     0073 L ENTRY1

0092 L ENTRY2    0095 L ENTRY3    00BC L ERROR      001B E F$CPYMEM    0006 E F$EXIT
0018 E F$GPRDSC  00BF L HELP      005A E HELPLEN     0016 L HEX01      002A L HEX2
0012 L HEXIN     0046 L HEXRST     008A E I$WRITE     008C E I$WRITLN     0007 D ID
0002 D INPUT     0136 F LEN        12D0 E MSIZE      000D L NAME       0003 D OFFSET
0008 D PRCDSC    0128 L SKIPSPE    1208 D STACK

Page 6-2-5
MMAP - Show memory block map, display mfree.
U = used, M = loaded module, . = no RAM, else FREE.
Of course, add at least one free block, since
MMAP's using one for data! This is my 128K map:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
# = = = = = = = = = = = = = = = =
0 U U U U M U M U M _ _ _ _ _ U .
1 . . . . . . . . . . . . . . .
2 . . . . . . . . . . . . . . .
3 . . . . . . . . . . . . . . .
```

Number of Free Blocks: 5
Ram Free in KBytes: 40
INSIDE OS9 LEVEL II
SOURCES
MMap

Microwave OS-9 Assembler RS Version 01.00.00 03/30/87 00:15:48 Page 001
MMap - INSIDE OS9 LEVEL II

00001 nam MMap
00002 ttl INSIDE OS9 LEVEL II
00003 * mmap - memory blockmap for cc3
00004 * 01 feb 87
00005 * Copyright 1987 by Kevin Darling
00006
00007 0006 F$Exit equa 6
00008 0019 F$GBlkMpequ $19
00009 008A I$Write equa $8A
00010 008C I$Writelequ $8C
00011
00012 0000 B7CD01El mod len,name,$ll,$81,entry,msize
00013 000D 4D4D61F0 name fcs "MMap"
00014 0011 03 fcb 3
00015
00016 0400 buffsiz set 1024
00017 D 0000 leadflag rmb 1
00018 D 0001 number rmb 3
00019 D 0004 free rmb 1
00020 D 0005 row rmb 1
00021 D 0006 spc rmb 1
00022 D 0007 out rmb 3
00023 D 000A mapsiz rmb 2
00024 D 000C blksz rmb 2
00025 D 000E blknnum rmb 1
00026 D 000F buffer rmb buffsiz
00027 D 040F stack rmb 200
00028 D 04D7 msize equa .
00029
00030 0012 header
00031 0012 20202020 fcc " 0 1 2 3 4 5 6 7 8 9 A B C D E F"
00032 0035 0D fcb $0D
00033 0024 hdrlenequ *-header
00034 0036 hdr2
00035 0036 20232020 fcc "# == == == == == == == == == == ==
00036 0059 0D fcb $0D
00037 0024 hdrlen2 equa *-hdr2
00038
00039 005A entry
00040 005A 1700EF lbrt crtn
00041 005D 308DPPB1 leax header,pc
00042 0061 8601 lda $1
00043 0063 10BE0024 ldy #hdrlen
00044 0067 103F8C OS9 I$Writele
00045 006A 308DFPC8 leax hdr2,pc
00046 006E 10BE0024 ldy #hdrlen2
00047 0072 103F8A OS9 I$Write
00048 0075 304F leax buffer,ul get block map
00049 0077 103F19 OS9 F$GBlkMpe
00050 007A 1025009B lbcse error
INSIDE OS9 LEVEL II
SOURCES

MMap

00051 007E 0F0E      clr blknun
00052 0080 0F04      clr free
00053 * std blksz
00054 * sty mapsiz
00055
00056 0082 304F      leax buffer,u
00057 0084 8630      lda #$30
00058 0086 9705      sta row
00059 0088 8640      lda #$40
00060 008A 3402      pshs a                         save count
00061 008C          loop
00062 008C A6E4      lda ,s
00063 008E 850F      bita #$0F
00064 0090 261F      bne loop2
00065
00066 0092 3410      pshs x
00067 0094 1700B5     lbrs crtn
00068 0097 3046      leax spc,u
00069 0099 10800004   ldy #4
00070 009D 9605      lda row
00071 009F 9707      sta out
00072 00A1 0C05      inc row
00073 00A3 CC2020    ldd #$2020
00074 00A6 9706      sta spc
00075 00A8 DD08      std out+1
00076 00AA 8601      lda #1
00077 00AC 103F8A     os9 IWrite
00078 00AF 3510      puls x
00079
00080 00B1          loop2
00081 00B1 F680      ldb ,x+                 get next block
00082 00B3 270A      beq freeram
00083 00B5 2B12      bmi notram
00084 00B7 C502      bitb #2
00085 00B9 260A      bne module
00086 00BB C655      ldb #$U                      ram-in-use
00087 00BD 200C      bra put
00088 00BF          freeram
00089 00BF C65F      ldb #'_                  not used
00090 00C1 0C04      inc free
00091 00C3 2006      bra put
00092 00C5          module
00093 00C5 C64D      ldb #'_M                module
00094 00C7 2002      bra put
00095 00C9          notram
00096 00C9 C62E      ldb #'.               not ram
00097 00CB          put
00098 00CB D707      stb out
00099 00CD C620      ldb #$20
00100 00CF D708      stb out+1
00101 00D1 3410      pshs x
00102 00D3 3047      leax out,u
00103 00D5 10800002   ldy #2
00104 00D9 8601      lda #1

Page 6-3-3
INSIDE OS9 LEVEL II
SOURCES

MMAP

00105 00DE 103F8A
00106 00DE 3510
00107 00E0 6AE4
00108 W 00E2 1026FFA6
00109 00E6 3502
00110
00111 00E8 8D62
00112 00EA 8D60
00113 00EC 308D002C
00114 00F0 108E0018
00115 00F4 8601
00116 00F6 103F8A
00117 00F9 D604
00118 00FB 4F
00119 W 00FC 170071
00120 00FF 8D4B
00121
00122 0101 308D002F
00123 0105 108E0018
00124 0109 8601
00125 010B 103F8A
00126 010E D604
00127 0110 8608
00128 0112 3D
00129 W 0113 17005A
00130 0116 8D34
00131 0118
00132 0118 5F
00133 0119
00134 0119 103F06
00135
00136 011C 204E756D
00137 0018
00138 0134 20202020
00139 0018
00140
00141 014C
00142 014C 3412
00143 014E 860D
00144 0150 9707
00145 0152 3047
00146 0154 108E0001
00147 0158 8601
00148 015A 103F8C
00149 015D 3592
00150
00151 015F
00152 015F 9707
00153 0161 341C
00154 0163 3047
00155 0165 108E0001
00156 0169 8601
00157 016B 103F8A
00158 016E 3590

OS9 ISWrite
puls x
dec ,s
lbne loop
puls a
bsr crtn
leax freemsg,pc
ldy #freelen
lda #1
OS9 ISWrite
ldb free
cir
lbr outdec
bsr crtn
leax rammsg,pc
ldy #ramlen
lda #1
OS9 ISWrite
ldb free
mul
lbr outdec
bsr crtn
bye
clrb
error
OS9 F$Exit
freemsg fcc " Number of Free Blocks: "
freelem equ -$freemsg
rammsg fcc " Ram Free in KEbytes: "
ramlen equ -$rammsg
crtn
pshs a,x
lda #80D
sta out
leax out,u
ldy #1
lda #1
OS9 ISWrite
puls a,x,pc
sta out
pshs x
leax out,u
ldy #1
lda #1
OS9 ISWrite
puls x,pc
INSIDE 089 LEVEL II

SOURCES

MMap

00159
00160  0170
00161  0170  3041
00162  0172  0F00
00163  0174  6F84
00164  0176  6F01
00165  0178  6F02
00166  017A
00167  017A  6C84
00168  017C  830064
00169  017F  24F9
00170  0181  C30064
00171  0184
00172  0184  6C01
00173  0186  83000A
00174  0189  24F9
00175  018B  C3000A
00176  018E  5C
00177  018F  E702
00178  0191  8D08
00179  0193  8D06
00180
00181  0195
00182  0195  A680
00183  0197  8B2F
00184  0199  20C4
00185
00186  019B
00187  019F  0D00
00188  019D  26F6
00189  019F  E684
00190  01A1  0C00
00191  01A3  5A
00192  01A4  26EF
00193  01A6  0F00
00194  01A8  8620
00195  01AA  3001
00196  01AC  20B1
00197
00198  01AE  42D247
00199  01B1
00200

outdec equ  *  D=number
leax number,u
clr leadflag
clr ,x
clr 1,x
clr 2,x

hundred
inc ,x
subd #100
bcc hundred
adda #100
ten
inc 1,x
subd #10
bcc ten
adda #10
incb
stb 2,x
bsr printled
bsr printled
printnum
lda ,x+
adda #$30-1  make ascii
bra print

printled
tst leadflag  print leading zero?
bne printnum  ..yes
ldb ,x  is it zero?
inc leadflag
decb
bne printnum  ..no, print zero's
clr leadflag  else suppress
lda #$20
leax 1,x
bra print

emod
len equ  *

end
INSIDE OS9 LEVEL II

SOURCES

MMap

00000 error(s)
00003 warning(s)
$01B1 00433 program bytes generated
$04D7 01239 data bytes allocated
$02B9 00697 bytes used for symbols

000E D BLKNUM 000C D BLKSIZ 000F D BUFFER 0400 S BUFSIZ 0118 L BYE
014C L CHTN 005A L ENTRY 0119 L ERROR 0006 E F$EXIT 0019 E F$GBLKM
0004 D FREE 0018 E FREELEN 011C L FREEMSG 00BF L FREEERAM 0036 L HDR2
0024 E HDRLEN 0024 E HDRLEN2 0012 L HEADER 017A L HUNDRED 008A E I$WRITE
008C E I$WRITLN 0000 D LEADFLAG 01E1 E LEN 008C L LOOP 00B1 L LOOP2
000A D MAPSIZ 00C5 L MODULE 04D7 E MSIZE 000D L NAME 00C9 L NOTRAM
0001 D NUMBER 0007 D OUT 0170 E OUTDEC 015F L PRINT 019B L PRINTLED
6195 L PRINTNUM 00CB L PUT 0018 E RAMLEN 0134 L RAMMSG 0005 D ROW
0006 D SPC 040F D STACK 0184 L TEN
PMAP - Process DAT Image Maps. The best. Shows blocks in use by processes. Lower is data, top is modules.
Example: block 09 is mapped into $6000-7FFF in the system dat map. Note that Shell in block 06 (see DIRM above) is simply mapped into both procs 2 and 3 at $E000-FE7F along with any other modules in that block.

<table>
<thead>
<tr>
<th>ID</th>
<th>01</th>
<th>23</th>
<th>45</th>
<th>67</th>
<th>89</th>
<th>AB</th>
<th>CD</th>
<th>EF</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td>09</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>3F SYSTEM</td>
</tr>
<tr>
<td>2</td>
<td>05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>06 Shell</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>06 Shell</td>
</tr>
<tr>
<td>4</td>
<td>0A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>08 PMap</td>
</tr>
</tbody>
</table>
INSIDE OS9 LEVEL II

SOURCES

PMap

Microware OS-9 Assembler RS Version 01.00.00 03/30/87 00:16:17  Page 001

PMap - INSIDE OS9 LEVEL II

00001 nam PMap
00002 ttl INSIDE OS9 LEVEL II
00003 * PMap - CC3 proc dating display
00004 * 08 feb 87 : derived from my Proc cmd.
00005 * Copyright 1987 by Kevin Darling
00006
00007 0088 D.PthDBT equ $0088
00008 00C3 PD.DEV equ $03
00009 0004 V$DESC equ $04
00010
00011 0006 P$Exit equ 6
00012 0018 P$GPrDsc equ $18
00013 001B P$CpyMem equ $1B
00014 008A I$Write equ $8A
00015 008C I$WriteLn equ $8C
00016
00017 0004 M$Name equ 4
00018
00019 0000 P$ID equ 0
00020 0001 P$PID equ 1
00021 0004 P$SP equ 4
00022 0006 P$Task equ 6
00023 0007 P$PagCnt equ 7
00024 0008 P$User equ 8
00025 000A P$Prior equ $0A
00026 000B P$Age equ $0B
00027 000C P$State equ $0C
00028 0010 P$IOQN equ $10
00029 0011 P$PModul equ $11
00030 0019 P$Signal equ $19
00031 0030 P$Path equ $30
00032 0040 P$DATImg equ $40
00033
00034 0000 87CD01F8 mod len,mname,$11,$81,entry,msize
00035 00D 504D61F0 mname fcs "PMap"
00036 0011 01 fcb 1
00037
00038 0200 buffsz set 512
00039
00040 D 0000 umem rmb 2
00041 D 0002 sysimg rmb 2 pointer to sysprc datimage
00042 D 0004 dating rmb 2 dating for copymem
00043 D 0006 lineptr rmb 2
00044 D 0008 number rmb 3
00045 D 000B leaflag rmb 1
00046 D 000C path rmb 2
00047 D 000E pid rmb 1
00048 D 000F hdr rmb 12 header
00049 D 001B out rmb 80
00050 D 006B buffer rmb buffsz each proc desc
00051 D 026B stack rmb 200

Page 6-4-2
00052 D 0333 msize equ
00053
00054 0012 header
00055 0012 20494420 fcc " ID 01 23 45 67 89 AB CD EF
Program
00056 003E 0D fcb $0D
00057 002D hdrlen equ *-header
00058 003F header2
00059 003F 2D2D2D2D fcc "----- -- -- -- -- -- -- -- "
00060 006B hdcr
00061 006B 0D fcb $0D
00062 002D hdrlen2 equ *-header2
00063
00064 006C entry
00065 006C DF00 stu unem
00066 006E 8601 lda #1
00067 0070 0F0E clr pid
00068 0072 308DFFF5 leax hdcr,pc
00069 0076 108E0001 ldy #1
00070 007A 103F8C OS9 I$Write
00071 W 007D 10250034 lbc error
00072 0081 308DF8D leax header,pc
00073 0085 108E0202 ldy #hdrlen
00074 0089 103F8C OS9 I$Write
00075 W 008C 10250025 lbc error
00076 0090 308DFFAB leax header2,pc
00077 0094 108E002D ldy #hdrlen2
00078 0098 103F8C OS9 I$Write
00079
00080 009B main
00081 009B DE00 ldu unem
00082 009D 30C81E leax out,u
00083 00A0 9F06 stx lineptr
00084 00A2 0C0E inc pid next process
00085 00A4 270E beq bye ..>255 = exit
00086 00A6 960E lda pid proc id
00087 00A8 30C86B leax buffer,u destination buff
00088 00AB 103F18 OS9 P$GetDesc get proc desc
00089 00AE 25EB bcs main ..loop if not one
00090 00B0 8D0C bsr output report proc data
00091 00B2 20E7 bra main ..loop.
00092
00093 00B4 bye
00094 00B4 5F clrb
00095 00B5 error
00096 00B5 103F06 OS9 P$Exit
00097
00098 00B8 53595354 sysnam fcs "SYSTEM"
00099 0006 syslen equ *-sysnam
00100
00101 00BE output
00102 00BE A684 lda P$ID,x process id
00103 00C0 1700E6 lbsr outdecl
00104 00C3 1700C1 lbsr space
INSIDE OS9 LEVEL II
SOURCES
PMap

00105 00C6 1700BE  lbsr  space
00106 00C9 1700BB  lbsr  space
00107 00CC 1700B8  lbsr  space
00108
00109  * Print Process DATImage:
00110  * X=proc desc
00111 00CF 3410  pshs  x
00112 00D1 308840  leax  P$DATImg,x
00113 00D4 C608  ldb  #8
00114 00D6 3404  pshs  b
00115 00D8  printk
00116 00D8 EC81  ldd ,x++
00117 00DA 4D  tst
00118 00DB 2710  beq  printk2
00119 00DD 109E06  ldy  lineptr
00120 00E0 CC2E2E  ldd  "..
00121 00E3 FDA1  std  ,y++
00122 00E5 109F06  sty  lineptr
00123 00E8 17009C  lbsr  space
00124 00EB 2005  bra  printk3
00125 00ED  printk2
00126 00ED 1F98  tfr  b,a
00127 00EF 170093  lbsr  outhexl
00128 00F2  printk3
00129 00F2 6AE4  dec  ,s
00130 00F4 26E2  bne  printk
00131 00F6 3514  puls  b,x
00132
00133  * Print Primary Module Name:
00134 00F8 17008C  lbsr  space
00135 00FB 318840  leax  P$DATImg,x
00136 00FE 1F20  tfr  y,d  D=dat image in proc desc
00137 0100 DD04  std  dating
00138 0102 AE8811  ldx  P$PModul,x  X=offset in map
00139 0105 2614  bne  doname
00140 0107 308DFPAD  leax  >sysnam,pc
00141 0109 109E06  ldy  lineptr
00142 010E C606  ldb  #syslen
00143 0110  copy
00144 0110 A680  lda  ,x+
00145 0112 A7A0  sta  ,y+
00146 0114 5A  decb
00147 0115 26F9  bne  copy
00148 0117 8D43  bsr  name2
00149 0119 2002  bra  printklin
00150 011B  doname
00151 011B 8D19  bsr  printknam
00152
00153 011D  printklin
00154 011D 9E06  ldx  lineptr  now print whole line:
00155 011F 860D  lda  #$0D
00156 0121 A784  sta  ,x
00157 0123 DE00  ldu  unem
00158 0125 30C81E  leax  out,u
00159  0128 108E0050  ldy  #80
00160  012C  8601  lda  #1
00161  012E  103F8C  OS9  I$WriteLn
00162  W  0131  1025FF80  lbcS  error
00163  0135  39  rts
00164
00165  * Find and Print a Module Name:
00166  * X=mod offset, U=data area, datimg=ptr
00167  0136  printnam
00168  0136  3440  pshs  u
00169  0138  334F  leau  hdr,u  destination
00170  013A  DC04  ldd  datimg  proc  datimg  ptr
00171  013C  108E000A  ldy  #10  Y=length
00172  0140  103F1B  OS9  P$CpyMem  get  header
00173  0143  1025FF6F  lbcS  error
00174
00175  0147  EC44  ldd  M$Name,u  get  name  offset  from  header
00176  0149  DE06  ldu  lineptr  move  name  to  output  line
00177  014B  308B  leax  d,x  X=offset  in  map  to  name
00178  014D  DC04  ldd  datimg
00179  014F  108E0028  ldy  #40  max  char  len
00180  0153  103F1B  OS9  P$CpyMem  get  name
00181  0156  3540  puls  u
00182  0158  1025FF59  lbcS  error
00183
00184  015C  name2
00185  015C  3410  pshs  x
00186  015E  9E06  ldx  lineptr
00187  0160  5F  clrB  b  is  length
00188  0161  name3
00189  0161  5C  incB
00190  0162  A680  lda  ,x+
00191  0164  2AFB  bpl  name3
00192  0166  C128  cmpB  #40
00193  0168  2411  bcc  name5
00194
00195  016A  847F  anda  #$7F  fix  it  up,  then
00196  016C  A71F  sta  -1,x
00197  016E  C109  cmpB  #9
00198  0170  2409  bcc  name5
00199  0172  8620  lda  #$20
00200  0174  name4
00201  0174  A780  sta  ,x+
00202  0176  5C  incB
00203  0177  C109  cmpB  #9
00204  0179  25F9  bcs  name4
00205  017B  name5
00206  017B  9F06  stx  lineptr
00207  017D  3590  puls  x,pc
00208
00209  *------------------------------------------------------------------------
00210
00211  017F  outhex2
00212  017F  3404  pshs  b

Page 6-4-5
INSIDE OS9 LEVEL II
SOURCES
PMap

00213 0181 8D08   bsr  hexl
00214 0183 3502   puls a
00215 0185       outhex
00216 0185 8D04   bsr  hexl
00217 0187       space
00218 0187 8620   lda  #$20
00219 0189 2014   bra  print
00220
00221 018B       hexl
00222 018E 1F09   tfr  a,b
00223 018D 44     lrsa
00224 018E 44     lrsa
00225 018F 44     lrsa
00226 0190 44     lrsa
00227 0191 8D02   bsr  outhex
00228 0193 1F98   tfr  b,a
00229 0195       outhex
00230 0195 640F   anda  #$0F
00231 0197 810A   cmpa  #$0A    0-9
00232 0199 2502   bcs  outdig
00233 019B 8D07   adda  #$07    A-F
00234 019D       outdig
00235 019D 8B30   adda  #$30    make ASCII
00236 019F       print
00237 019F 341C   pshs x
00238 01A1 5D06   ldx  lineptr
00239 01A3 A780   sta ,x
00240 01A5 9F06   stx  lineptr
00241 01A7 3590   puls x,pc
00242
*-----------------------------------------------
00243 01A9       outdecl equ * A=number
00244 01A9 1F09   tfr  a,b
00245 01AB 4F     clra
00246 01AC       outdec equ * D=number
00247 01AC 0F0B   clrz leadflag
00248 01AE 3410   pshs x
00249 01B0 9E00   ldx  umem
00250 01E2 3008   leax number,x
00251 01E4 6F84   clrz ,x
00252 01E6 6F01   clrz 1,x
00253 01E8 6F02   clrz 2,x
00254 01EA       hundred
00255 01EA 6C84   inc  ,x
00256 01EC 830064  subd  #100
00257 01BF 24F9   bcc  hundred
00258 01C1 C30064  addd  #100
00259 01C4       ten
00260 01C4 6C01   inc  1,x
00261 01CE 83000A  subd  #10
00262 01C9 24F9   bcc  ten
00263 01CB C3000A  addd  #10
00264 01CE 5C     incb
00265 01CF E702   stb  2,x

Page 6-4-6
INSIDE OS9 LEVEL II
SOURCES
PMap

00267
00268  01D18D0F
00269  01D38D0D
00270  01D58D05
00271  W 01D717FFAD
00272  01DA3590
00273
00274  01DC
00275  01DC A680
00276  01DE8B2F
00277  01E020BD
00278
00279  01E2
00280  01E2 0D0B
00281  01E4 26F6
00282  01E6 E684
00283  01E8 0C0B
00284  01EA 5A
00285  01EB 26EF
00286  01ED 0F0B
00287  01EF 8620
00288  01F1 3001
00289  01F3 20AA
00290
00291  01F5 474519
00292  01F8
00293
00000 error(s)
00004 warning(s)
01F8 00504 program bytes generated
0333 00819 data bytes allocated
0499 01177 bytes used for symbols

006B D BUFFER 0200 S BUFSIZ 00B4 L BYE 0110 L COPY 0088 E D.PTHDBT
0004 D DATIMG 011B L DONAHE 006C L ENTRY 00B5 L ERROR 001B E $CPFMEM
0006 E $EXIT 0018 E $GPRDSC 000F D HDR 006B L HDRC 002D E HDRLEN
002D E HDRLEN2 0012 L HEADER 003F L HEADER2 018B L HEX1 018A L HUNDRED
008A E $WRITE 008C E $WRITL 000B D LEADFLAG 01F8 E LEN 0006 D LINEPR
0004 E $NAME 009B L MAIN 00D L MNAME 0333 E MSIZE 015C L NAME2
0161 L NAME3 0174 L NAME4 017E L NAME5 0008 D NUMBER 001B D OUT
01AC E OUTDEC 01A9 E OUTDEC1 019D L OUTDIG 0195 L OUTHEX 0185 L OUTHEX1
017F L OUTHEX2 00BE L OUTPUT 000B E $PAGE 0040 E $DATIMG 0000 E $ID
0010 E $IOQ 0007 E $PAGCNT 0030 E $PATH 0001 E $PID 0011 E $PMODUL
000A E $PRIOR 0019 E $SIGNL 0004 E $SP 000C E $STATE 0006 E $TASK
0008 E $USER 000C D PATH 0003 E PD_DEV 000E D PID 019F L PRINT
01E2 L PRINTED 01D L PRINTLIN 0136 L PRINTNAM 01DC L PRINTNUM 00D8 L PRIMG
00ED L PRIMG2 00F2 L PRIMG3 0187 L SPACE 026B D STACK 0002 D SYSIMG
0006 E SYSL 00B8 L SYSTAL 01C4 L TEN 0000 D UMEM 0004 E $DESC
PROC - Like procs, but shows standard in/out devices:
St = status byte, Sig = pending signal in hex and dec.

Example:

OS9: dirm >/w7 & (setpr 2 255; proc >/d1/test)

<table>
<thead>
<tr>
<th>ID</th>
<th>Prnt</th>
<th>User</th>
<th>Pty</th>
<th>Age</th>
<th>St</th>
<th>Sig</th>
<th>Module</th>
<th>Std in/out</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>Shell &lt;Term &gt;Term</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>128</td>
<td>128</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>Shell &lt;W1 &gt;W1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>128</td>
<td>128</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>DirM &lt;Term &gt;W7</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>128</td>
<td>130</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>Shell &lt;Term &gt;Term</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>128</td>
<td>129</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>Proc &lt;Term &gt;D1</td>
</tr>
</tbody>
</table>
PROC - INSIDE OS9 LEVEL II

* Proc - L-II Proc for coco 3

* 06 Feb 87: add std out also
* 03 Feb 87: add path name display
* 01 Feb 87: working version
* Copyright 1987 by Kevin Darling

D.PthDBT equ $0088
PD.DEV equ $03
V$DESC equ $04
M$Name equ 4
PSID equ 0
PS$ID equ 1
PS$SP equ 4
PS$Task equ 6
PS$PgCnt equ 7
PS$User equ 8
PS$Prior equ $0A
PS$Age equ $0B
PS$State equ $0C
PS$IOQN equ $10
PS$Modul equ $11
PS$Signal equ $19
PS$Path equ $30
PS$DATimg equ $40
mod len,mname,$11,$81,entry,msize
mname fcs "Proc"
fcb 9
buffsiz set 512

# Pointer registers

umem rmb 2
sysimg rmb 2

datimg rmb 2
dateptr rmb 2
number rmb 3
leadflag rmb 1
path rmb 2
pid rmb 1
INSIDE OS9 LEVEL II
SOURCES
Proc

00051 D 000F namlen rmb 1
00052 D 0010 hdr rmb 64 header
00053 D 0050 out rmb 80
00054 D 00A0 buffer rmb buffsiz each proc desc
00055 D 02A0 sysprc rmb buffsiz sys proc desc
00056 D 04A0 stack rmb 200
00057 D 0568 msize equ .
00058
00059 0012 header fcc " ID Prnt User Pty Age St Sig .. Module
00060 0012 20494420 fcb $0D
00061 0048 0D hdrlen equ *-header
00062 0037 header2 fcc "--- --- ----- --- --- -- --- -----
00063 0049 2D2D2D20 hdrcr fcb $0D
00064 0049 07F header2 equ *-header2
00065 0067 037 entry
00066 0070 080 DFOO stu unem
00067 0071 082 8601 lda #1
00068 0072 084 970E sta pid
00069 0073 086 308DF5F5 leax hdrcr,pc
00070 0074 08A 108E0001 ldy #1
00071 0075 08E 103F08C OS9 I$Writeln
00072 0076 W 091 10250045 lbc$ error
00073 0077 095 308DF79 leax header,pc
00074 0078 099 108E0037 ldy #hdrlen
00075 0079 09D 103F08C OS9 I$Writeln
00076 W 090 10250036 lbc$ error
00077 0081 0A4 308DFFA1 leax header2,pc
00078 0082 0A8 108E0037 ldy #hdrlen2
00079 0083 0AC 103F08C OS9 I$Writeln
00080 0084 0AF 8601 lda #1
00081 0085 0BL 30C902A0 leax \sysprc,u get system proc desc
00082 0086 0B5 103F18 OS9 F$GPDesc
00083 0087 0B8 2520 bcs error
00084 0088 0EA 308840 leax F$DATImg,x just for it's dating
00085 0089 0ED 9F02 stx Sysimg
00086 008A main
00087 0090 0BF DE00 ldu unem
00088 0093 0C1 30C850 leax out,u
00089 0094 0C4 9F06 stx lineptr
00090 0095 0C6 0C0E inc pid next process
00091 0096 0C8 270F beq bye ..>255 = exit
00092 0097 0CA 960E lda pid proc id
00093 0098 0CC 30C900A0 leax buffer,u destination buff
00094 0099 0DD 103F18 OS9 F$GPDesc get proc desc
00095 009A 0D3 25EA bcs main ..loop if not one
00096 009B 0D5 8D06 bsr output report proc data
00097 009C 0D7 20E6 bra main ..loop.
00098

Page 6-5-3
INSIDE OS9 LEVEL II

SOURCES

Proc

0C105 00D9   bye
0C106 00D9 5F  clrb
0C107 00DA   error
0C108 00DA 103F06  OS9F$Exit
0C109
0C110 00DD   output
0C111 00DD A684  lda P$ID,x  process id
0C112 00DF 17015D  lshr outdec1
0C113 00E2 A601  lda P$PID,x  parent's id
0C114 00E4 170158  lshr outdec1
0C115 00E7 170133  lshr space
0C116 00EA EC08  ldd P$User,x  user id
0C117 00EC 170153  lshr outdec
0C118 00EF 17012B  lshr space
0C119 00F2 A60A  lda P$Prior,x  priority
0C120 00F4 170148  lshr outdec1
0C121 00F7 A60B  lda P$Age,x
0C122 00F9 170143  lshr outdec1
0C123  * lda P$Task,x task number
0C124  * lshr outhex1
0C125 00FC 17011E  lshr space
0C126 00FF A60C  lda P$State,x  state
0C127 0101 170117  lshr outhex1
0C128 0104 A6819  lda P$Signal,x signal
0C129 0107 170135  lshr outdec1
0C130 010A A6819  lda P$Signal,x signal in hex
0C131 010D 17010B  lshr outhex1
0C132
0C133 0110 17010A  lshr space
0C134 0113 EC830  ldd P$Path,x  save proc stdin path #
0C135 0116 DD0C  std path  and path1 stdout
0C136
0C137  * Print Primary Module Name:
0C138  * X=proc desc
0C139 0118 318840  lea y P$DAT1mg,x
0C140 011B 1P20  tfr y,d  D=dat image in proc desc
0C141 011D DD04  std naming
0C142 011F AE811  ldx P$PModul,x  X=offset in map
0C143 0122 C609  ldb #9
0C144 0124 D70F  stb namlen
0C145 0126 1700A2  lshr printnam
0C146
0C147  * Print Std Input Device:
0C148 0129 863C  lda #'<
0C149 012B 8D21  bsr device
0C150 012D    stdout
0C151 012D 960D  lda path+1
0C152 012F 970C  sta path
0C153 0131 863F  lda #'>
0C154 0133 8D19  bsr device
0C155
0C156 0135  println
0C157 0135 9E06  ldx lineptr  now print whole line:
0C158 0137 860D  lda #$8D

Page 6-5-4
INSIDE OS9 LEVEL II

SOURCES

Proc

00159 0139 A784  sta ,x
00160 013B DE00  ldu unmem
00161 013D 30C850  leax out,u
00162 0140 108E0050  ldy #80
00163 0144 8601  lda #1
00164 0146 103F8C  OS9 $Writeln
00165 W 0149 1025FF8D  lbc error
00166 014D 39  rts

00167

00168 014E  device
00169 014E DE00  ldu unmem
00170 0150 1700E2  lbrs print ("< >")
00171 0153 960C  ldi path
00172 0155 2610  bne device2
00173 0157 8620  ldi #$20
00174 0159 C605  ldb #5
00175 015E 109E06  ldy lineptr
00176 015E device0

00177 015E A7A0  sta ,y+
00178 0160 5A  decb
00179 0161 26FB  bne device0
00180 0163 109F06  sty lineptr
00181 0166 39  rts

00182

00183 0167  device2

00184 0167 33C810  leau hdr,u get path table offset:
00185 016A DC02  ldd sysimg in system map
00186 016C 800088  ldx #$D.PthDBT from direct page ptr
00187 016F 108E0002  ldy #2
00188 0173 103F1B  OS9 $CpyMem
00189 0176 1025FF60  lbc error

00190

00191 017A 9E10  ldx hdr get path descriptor table:
00192 017C 108E0040  ldy #64
00193 0180 DC02  ldd sysimg
00194 0182 103F1B  OS9 $CpyMem (X was D.PthDBT ptr)
00195 0185 1025FF51  lbc error

00196

00197 0189 D60C  ldb path point to path block:
00198 018B 54  lsrb four paths / sub-block
00199 018C 54  lsrb
00200 018D A6C5  lda b,u A=msb block address
00201 018F 3402  pshs a
00202 0191 D60C  ldb path then point to path within
00203 0193 C403  andb #$3
00204 0195 8640  lda #$40
00205 0197 3D  mul
00206 0198 3502  puls a D=path descriptor address

00207

00208 019A CB03  addb #$PD.DEV and get device table ptr
00209 019C 1001  tfr d,x
00210 019E DC02  ldd sysimg
00211 01A0 108E0002  ldy #2
00212 01A4 103F1B  OS9 $CpyMem
INSIDE OS9 LEVEL II

00213  01A7 1025FF2F    lbc$ error
00214
00215  01AB 9E10    ldb #V$DESC but we want it's desc ptr
00216  01AD C604    ldx hdr X=device table entry sys addr
00217  01AF 3A    abx
00218  01B0 DC02    ldd sysimg
00219  01B2 108E0002    ldy #2
00220  01B6 103F1B    OS9 F$CpyMem
00221  01B9 1025FF1D    lbc$ error
00222
00223  01BD 9E10    ldx hdr then get desc address:
00224  01BF DE00    ldu umem
00225  01C1 DC02    ldd sysimg
00226  01C3 DD04    std datimg
00227  01C5 C605    ldb #5
00228  01C7 D70F    stb namlen
00229  01C9 2000    bra printnam and get device name
00230
00231  * Find and Print a Module Name:
00232  * X=mod offset, U=data area, datimg=ptr
00233  01CB printnam
00234  01CB 3440    pshs u
00235  01CD 33C810    leau hdr,u destination
00236  01D0 DC04    ldd datimg proc datimg ptr
00237  01D2 108E000A    ldy #10 Y=length
00238  01D6 103F1B    OS9 F$CpyMem get header
00239  01D9 1025FFED    lbc$ error
00240
00241  01DD EC44    ldd #S>Name,u get name offset from header
00242  01DF DE06    ldu lineptr move name to output line
00243  01E1 308B    leax d,x X=offset in map to name
00244  01E3 DC04    ldd datimg
00245  01E5 108E0028    ldy #40 max char len
00246  01E9 103F1B    OS9 F$CpyMem get name
00247  01EC 3540    puls u
00248  01EE 1025FFEB    lbc$ error
00249
00250  01F2 3410    pshs x
00251  01F4 9B06    ldx lineptr
00252  01F6 5F    clrb B is length
00253  01F7 name3
00254  01F7 5C    incb
00255  01F8 A680    lda ,x+ fix it up, then
00256  01FA 2AFB    bpl name3
00257  01FC C128    cmpb #40
00258  01FE 2411    bcc name5
00259
00260  0200 847F    anda #$7F
00261  0202 A71F    sta -1,x
00262  0204 D10F    cmpb namlen
00263  0206 2409    bcc name5
00264  0208 8620    lda #$20
00265  020A name4

Page 6-5-6
INSIDE OS9 LEVEL II
SOURCES

Proc

00266 020A A780    sta   ,x+
00267 020C  5C     incb
00268 020D D10F    cmpb namlen
00269 020F  25F9    bcs name4
00270 0211        name5
00271 0211  9F06    stx  lineptr
00272 0213  3590    puls x,pc
00273
00274
00275
00276 0215        outhex2
00277 0215  3404    pshs  b
00278 0217  8D08    bsr  hex1
00279 0219  3502    puls  a
00280 021B        outhex1
00281 021B  8D04    bsr  hex1
00282 021D        space
00283 021D  8620    lda  #$20
00284 021F  2014    bra  print
00285
00286 0221        hex1
00287 0221  1F89    tfr  a,b
00288 0223  44     lsra
00289 0224  44     lsra
00290 0225  44     lsra
00291 0226  44     lsra
00292 0227  8D02    bsr  outhex
00293 0229  1F98    tfr  b,a
00294 022B        outhex
00295 022B  840F    anda  #$0F
00296 022D  810A    cmpa  #$0A    0-9
00297 022F  2502    bcs  outdig
00298 0231  8B07    adda  #$07    A-F
00299 0233        outdig
00300 0233  8B30    adda  #$30    make ASCII
00301 0235        print
00302 0235  3410    pshs  x
00303 0237  9E06    ldx  lineptr  ++++
00304 0239  A780    sta   ,x+
00305 023B  9F06    stx  lineptr
00306 023D  3590    puls x,pc
00307
00308
00309 023F        outdecl equ  *   A=number
00310 023F  1F89    tfr  a,b
00311 0241  4F     clra
00312 0242        outdec equ  *   D=number
00313 0242  0F0B    clra ledflag
00314 0244  3410    pshs  x
00315 0246  9E00    ldx  umem
00316 0248  3008    leax number,x
00317 024A  6F84    clra  ,x
00318 024C  6F01    clra  1,x
00319 024E  6F02    clra  2,x

Page 6-5-7
INSIDE OS9 LEVEL II

SOURCES

Proc

00320 0250 hundred
00321 0250 6C84 inc ,x
00322 0252 830064 subd #100
00323 0255 24F9 bcc hundred
00324 0257 C30064 addd #100
00325 025A ten
00326 025A 6C01 inc 1,x
00327 025C 83000A subd #10
00328 025F 24F9 bcc ten
00329 0261 C3000A addd #10
00330 0264 5C incb
00331 0265 E702 stb 2,x
00332
00333 0267 8D0F bsr printled
00334 0269 8D0D bsr printled
00335 026B 8D05 bsr printnum
00336 026D 17FFAD lbsr space
00337 0270 3590 puls x,pc
00338
00339 0272 printnum
00340 0272 A680 lda ,x+
00341 0274 8B2F adda #$30-1 make ascii
00342 0276 20BD bra print
00343
00344 0278 printled
00345 0278 0D0B tst leadflag print leading zero?
00346 027A 26F6 bne printnum .yes
00347 027C E604 ldb ,x is it zero?
00348 027E 0C0B inc leadflag
00349 0280 5A decb
00350 0281 26EF bne printnum .no, print zero's
00351 0283 0F0B clr leadflag else surpress
00352 0285 8620 lda #$20
00353 0287 3001 leax l,x
00354 0289 20AA bra print
00355
00356 028B 015EAF emod
00357 028E len equ *
00358

Page 6-5-8
SMAP - Show system page memory map. As above, except in pages.
Important info adding drivers, starting many procs, etc.

```
address: 0 1 2 3 4 5 6 7 8 9 A B C D E F
%  == == == == == == == == == == == == == == == ==
0  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
1  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
2  --- --- --- --- --- --- --- --- --- --- ---
3  --- --- --- --- --- --- --- --- --- --- ---
4  --- --- --- --- --- --- --- --- --- --- ---
5  --- --- --- --- --- --- --- --- --- --- ---
6  --- --- --- --- --- --- --- --- --- --- ---
7  --- --- --- --- --- --- --- --- --- --- ---
8  --- --- --- --- --- --- --- --- --- --- ---
9  --- --- --- --- --- --- --- --- --- --- ---
A  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
B  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
C  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
D  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
E  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
F  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
```

Number of Free Pages: 98
Ram Free in KBytes: 24
Inside OS9 Level II
Sources
SMap

Microwave OS-9 Assembler RS Version 01.00.00  03/30/87 00:17:48  Page 001
SMap - Inside OS9 Level II

00001  nam       SMap
00002  ttl       Inside OS9 Level II
00003  * SMap - System Memory Blockmap For CC3
00004  * 08 Feb 87
00005  * Copyright 1987 By Kevin Darling

00006
00007  004E     D.SysMem equ $004E system mem map
00008
00009  0066     FS Exit equ 6
00010  001B     F$CopyMem equ $1F
00011  008A     I$Write equ $8A
00012  008C     I$WriteLn equ $8C
00013
00014  0000     87CD01D5  mod len, name, $11, $81, entry, msize
00015  000D     534D61F0  name fcs "SMap"
00016  0011     01  fcb 1
00017
00018  0100     buffsz set 256
00019
00020  D 0000     leadflag rmb 1
00021  D 0001     number rmb 3
00022  D 0004     free rmb 1
00023  D 0005     row rmb 1
00024  D 0006     spc rmb 1
00025  D 0007     out rmb 3
00026  D 000A     mapsiz rmb 2
00027  D 000C     blksz rmb 2
00028  D 000E     blknum rmb 1
00029  D 000F     buffer rmb buffsz
00030  D 010F     stack rmb 200
00031  D 01D7     msize equ .
00032
00033  0012     header
00034  0012  20202020  fcc " 0 1 2 3 4 5 6 7 8 9 A B C D E F"
00035  0035     OD  fcb $0D
00036  0024     hdrlen equ *-header
00037  0036     hdr2
00038  0036  20232020  fcc " 0 1 2 3 4 5 6 7 8 9 A B C D E F"
00039  0059     OD  fcb $0D
00040  0024     hdrlen2 equ *-hdr2
00041
00042  005A  00000000  dating fdb 0,0
00043
00044  005E     entry
00045  005E  17010F  lbsr crtn
00046  0061  308DDFAAD  leax header, pc
00047  0065  8601  lda #1
00048  0067  108E0024  ldy #hdrlen
00049  006E  103F8C  OS9 IS$Write
00050  006E  308DF0C4  leax hdr2, pc
00051  0072  108E0024  ldy #hdrlen2
INSIDE OS9 LEVEL II

SOURCES

SMap

00052 0076 103F8A
00053
00054  * Get SysMap Ptr:
00055 0079 308DFPDD
00056 007D 1F10
00057 007F 8E004E
00058 0082 108E0002
00059 0086 3440
00060 0088 334F
00061 008A 103F1B
00062 008D 3540
00063 008F 102500AC
00064
00065  * Get SysMap:
00066 0093 AE4F
00067 0095 108E0100
00068 0099 3440
00069 009B 334F
00070 009D 103F1B
00071 00A0 3540
00072 00A2 10250099
00073
00074 00A6 0F0E
00075 00A8 0F04
00076  * std blksiz
00077  * sty mapsiz
00078 00AA 304F
00079 00AC 8630
00080 00AE 9705
00081 00B0 6FE2
00082 00B2
00083 00B2 A6E4
00084 00B4 850F
00085 00F6 2627
00086
00087 00B8 3410
00088 00BA 1700B3
00089 00BD 3046
00090 00BF 108E0004
00091 00C3 9605
00092 00C5 813A
00093 00C7 2604
00094 00C9 8641
00095 00CB 9705
00096 00CD
00097 00CD 9707
00098 00CF 0C05
00099 00D1 CC2020
00100 00D4 9706
00101 00D6 DD08
00102 00D8 8601
00103 00DA 103F8A
00104 00DD 3510

OS9  I$Write
leax datimg,pc
tfr x,d
ldx #D.SysMem
ldy #2
pshs u
leau buffer,u
OS9 F$CpyMem
puls u
lbc Error

get map address
ldy #buffsiz
pshs u
leau buffer,u
OS9 F$CpyMem
puls u
lbc Error
clr blknum
clr free
leax buffer,u
lda #$30
sta row
clr ,s
save count
ldc ,s
bne loop2
pshs x
lbcr crtn
leax spc,u
ldy #4
lda row
cmpa #$3A
bne oknum
lda #$41
sta row
oknum
sta out
inc row
ldd #$2020
sta spc
std out+1
lda #1
OS9 I$Write
puls x

Page 6-6-3
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00106</td>
<td>00DF</td>
<td></td>
<td>loop2</td>
<td></td>
</tr>
<tr>
<td>00107</td>
<td>00DF E680</td>
<td>ldb ,x+</td>
<td>get next block</td>
<td></td>
</tr>
<tr>
<td>00108</td>
<td>00E1 270A</td>
<td>beq freeram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00109</td>
<td>00E3 2B04</td>
<td>bmi notram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110</td>
<td>00E5 C655</td>
<td>ldb #'U</td>
<td>ram-in-use</td>
<td></td>
</tr>
<tr>
<td>00111</td>
<td>00E7 2008</td>
<td>bra put</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00112</td>
<td>00E9</td>
<td></td>
<td>notram</td>
<td></td>
</tr>
<tr>
<td>00113</td>
<td>00E9 C62E</td>
<td>ldb #'.</td>
<td>not RAM</td>
<td></td>
</tr>
<tr>
<td>00114</td>
<td>00EB 2004</td>
<td>bra put</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00115</td>
<td>00ED</td>
<td></td>
<td>freeram</td>
<td></td>
</tr>
<tr>
<td>00116</td>
<td>00ED C65F</td>
<td>ldb #'.</td>
<td>not used</td>
<td></td>
</tr>
<tr>
<td>00117</td>
<td>00EF 0C04</td>
<td>inc free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00118</td>
<td>00F1</td>
<td></td>
<td>put</td>
<td></td>
</tr>
<tr>
<td>00119</td>
<td>00F1 D707</td>
<td>stb out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00120</td>
<td>00F3 C620</td>
<td>ldb #$20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00121</td>
<td>00F5 D708</td>
<td>stb out+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00122</td>
<td>00F7 3410</td>
<td>pshs x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00123</td>
<td>00F9 3047</td>
<td>leax out,u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00124</td>
<td>00FE 108E0002</td>
<td>ldy #2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00125</td>
<td>00FF 8601</td>
<td>lda #1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00126</td>
<td>0101 103F8A</td>
<td>OS9 InWrite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00127</td>
<td>0104 3510</td>
<td>puls x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00128</td>
<td>0106 6AE4</td>
<td>dec ,s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00129</td>
<td>W 0108 1022FPA6</td>
<td>lbhi loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00130</td>
<td>010C 3502</td>
<td>puls a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00131</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00132</td>
<td>010E 8D60</td>
<td>bsr crtn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00133</td>
<td>0110 8D5E</td>
<td>bsr crtn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00134</td>
<td>0112 308D0002C</td>
<td>leax freemsg,pc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00135</td>
<td>0116 108E0017</td>
<td>ldy #freelen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00136</td>
<td>011A 8601</td>
<td>lda #1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00137</td>
<td>011C 103F8A</td>
<td>OS9 InWrite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00138</td>
<td>011F D604</td>
<td>ldb free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00139</td>
<td>0121 4F</td>
<td>clra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00140</td>
<td>W 0122 17006F</td>
<td>lbsr outdec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00141</td>
<td>0125 8D49</td>
<td>bsr crtn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00142</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00143</td>
<td>0127 308D002E</td>
<td>leax *rammsg,pc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00144</td>
<td>012E 108E0017</td>
<td>ldy #ramlen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00145</td>
<td>012F 8601</td>
<td>lda #1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00146</td>
<td>0131 103F8A</td>
<td>OS9 InWrite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00147</td>
<td>0134 D604</td>
<td>ldb free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00148</td>
<td>0136 4F</td>
<td>clra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00149</td>
<td>0137 54</td>
<td>lsrb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00150</td>
<td>0138 54</td>
<td>lsrb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00151</td>
<td>W 0139 170058</td>
<td>lbsr outdec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00152</td>
<td>013C 8D32</td>
<td>bsr crtn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00153</td>
<td>013F</td>
<td>bye</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00154</td>
<td>013E 5F</td>
<td>clrb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00155</td>
<td>013F</td>
<td>error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00156</td>
<td>013F 102F06</td>
<td>OS9 F$Exit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00157</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00158</td>
<td>0142 204E756D</td>
<td>freemsg fcc</td>
<td>&quot; Number of Free Pages: &quot;</td>
<td></td>
</tr>
<tr>
<td>00159</td>
<td>0017</td>
<td>freelen equ</td>
<td>*-freemsg</td>
<td></td>
</tr>
</tbody>
</table>

Page 6-6-4
sources

rammsg fcc " Ram Free in KBytes: "
ramlen equ *-rammsg

crtn
pshs a,x
lda #$0D
sta out
lea out,u
ldy #1
lda #1
OS9 I$Write
puls a,x,pc

print
sta out
pshs x
lea out,u
ldy #1
lda #1
OS9 I$Write
puls x,pc

outdec equ * D=number
lea number,u
clr leadflag
clr ,x
clr 1,x
clr 2,x
hundred
inc ,x
subd #$100
bcc hundred
addd #$100

194 3041
ten
inc 1,x
subd #$10
bcc ten
addd #$10

incb
stb 2,x
bsr printed
bsr printed

printnum
lda ,x+
addd #$30-1 make ascii
bra print

printled
tst leadflag print leading zero?
bne printnum ..yes
ldb ,x is it zero?
inc leadflag
deck
INSIDE OS9 LEVEL II

SOURCES
SMap

00214 01C8 26EF       bne printnum ..no, print zero's
00215 01CA 0F00       clr leadflag else surprise
00216 01CC 8620       lda #$20
00217 01CE 3001       leax l,x
00218 01D0 20B1       bra print
00219
00220 01D2 1F9F9F     emod
00221 01D5            len equ *
00222            end

00000 error(s)
00003 warning(s)
$01D5 00469 program bytes generated
$01D7 00471 data bytes allocated
$02D7 00727 bytes used for symbols

000E D BLKNUM 000C D BLKSIZE 000F D BUFFER 0100 S BUFSIZE 013E L BYE
0170 L CRTN 004E E D.SYSEM 005A L DATING 005E L ENTRY 013F L ERROR
001B E F$COPYEM 0006 E F$EXIT 0004 D FREE 0017 E FREELEN 0142 L FREEMSG
00ED L FREERAM 0036 L HDR2 0024 E HDRLEN 0024 E HDRLEN2 0012 L HEADER
019E L HUNDRED 008A E I$WRITE 008C E I$WRITNL 0000 D LEADFLAG 01D5 E LEN
00B2 L LOOP 00DF L LOOP2 00A0 D MAPSIZE 01D7 E MSIZE 000D L NAME
0029 L NOTPARM 0001 D NUMBER 00CD L OKNUM 0007 D OUT 0194 E OUTDEC
0183 L PRINT 01BF L PRINTLED 01F9 L PRINTNUM 00F1 L PUT 0017 E RAMLEN
0159 L RAMMSG 0005 D ROW 0006 D SPC 010F D STACK 01A8 I TEN
INSIDE OS9 LEVEL II

Reference
SYSTEM MEMORY MAP:
RAM    00000 - 7FFFF 512K bytes
ROM    78000 - 7FEFF when enabled
I/O    XFF00 - XFFFF I/O space and GIME regs

64K PROCESS MAP:
RAM    0000 - FEFF (possible vector page FEXX)
I/O    FF00 - FFFF (appears in all pages)

Note: the Vector Page RAM at 7FE00 - 7FEFF, when enabled, will appear instead
of the RAM or ROM at XFE00 - XFEFF. (see FF90 Bit 3)
XFF00-0X  PIA0   (not fully decoded)
XFF10-1F  reserved
XFF20-2X  PIA1   (not fully decoded)
XFF30-3F  reserved
XFF40-5F  SCS    (see note on FF90 Bit 2)
XFF60-7F  undecoded (for current peripherals)
XFF80-8F  reserved

FF90 INITIALIZATION REGISTER 0
Bit 7  -  CoCo Bit  1= Color Computer 1/2 Compatible
Bit 6  -  1= MMU enabled
Bit 5  -  1= GIME IRQ output enabled to CPU
Bit 4  -  1= GIME FIRQ "   "
Bit 3  -  1= Vector page RAM at FEXX enabled
Bit 2  -  1= Standard SCS
Bit 1  -  ROM mapping  0 X - 16K internal, 16K external
        "   "  1 0 - 32K internal
        "   "  1 1 - 32K external

CoCo bit set = MMU disabled, Video address from SAM, RGB/Comp Palettes = CC2.
Interrupt bits 5 and/or 4 must be set for FIRQ/IRQ FF92-3 to pass to CPU.
Access and moves throughout mem are usually done from constant RAM at FEXX.
If Bit2=0, then XFF50-5F is SCS, and XFF40-4F will be internal to CoCo.

FF91 INITIALIZATION REGISTER 1
Bit 5  -  TINS    Timer INput Clock Select: 0= 70 nsec, 1= 63 usec
Bit 0  -  TR      MMU Task Register Select (0/1 - see FFA0-AF)

7-1-1
INSIDE OS9 LEVEL II
Reference
Section 1

FF92  IRQENR  Interrupt Request Enable Register (IRQ)
FF93  FIRQENR  Fast Interrupt Request Enable Reg (FIRQ)
(Note that the equivalent interrupt output enable bit must be set in FF90.)
Both registers use the following bits to enable/disable device interrupts:
Bit 5 - TMR  Timer
Bit 4 - HBORD  Horizontal border
Bit 3 - VBORD  Vertical border
Bit 2 - E12  Serial data input
Bit 1 - E11  Keyboard
Bit 0 - E10  Cartridge (CART)

I have no idea if both IRQ & FIRQ can be enabled for a device at same time.

FF94  Timer MSB  Write here to start timer.
FF95  Timer LSB
Load starts timer countdown. Interrupts at zero, reloads count & continues.
Must turn timer interrupt enable off/on again to reset timer IRQ/FIRQ.

FF96  reserved
FF97  reserved

FF98  Alpha/graphics Video modes, and lines per row.
Bit 7 = BP  0 is alphanumeriic, 1= bit plane (graphics)
Bit 6 = na ...
Bit 5 = BPI 1= color burst phase change
Bit 4 = MCH  MOnoCHrome bit (composite video output) (1=mono)
Bit 3 = H50  50hz vs 60hz bit
Bit 2 = LPR2  Number of lines/char row:
Bit 1 = LPR1  (Bits 2-1-0 below:)
Bit 0 = LPR0
000 - 1 line/char row  100 - 9 lines/char row
001 - 2  101 - 10
010 - 3  110 - 11 (??)
011 - 8  111 - 12 (??)

FF99  VIDEO RESOLUTION REGISTER
Bit 7 - na ...
(bits 6-5):
Bit 6 - LPF1  Lines Per Field: 00= 192 lines  10= 210 lines
Bit 5 - LPF0  "  "  01= 200 lines  11= 225 lines
Bit 4 - HR2  Horizontal Resolution
Bit 3 - HR1  "  "
Bit 2 - HR0  "  " (see below for HR, CRES bits)
Bit 1 - CRES1  Color RESolution bits
Bit 0 - CRES0  "  "

7-1-2
TEXT MODES:

Text: CoCo Bit= 0 and FF98 bit7=0. CRES0 = 1 for: attribute bytes are used.

<table>
<thead>
<tr>
<th>HR2</th>
<th>HR1</th>
<th>HR0</th>
<th>(HR1 = don't care for text)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>&quot;</td>
<td>0</td>
<td>X 1</td>
</tr>
<tr>
<td>32</td>
<td>&quot;</td>
<td>0</td>
<td>X 0</td>
</tr>
</tbody>
</table>

GRAPHICS MODES:

<table>
<thead>
<tr>
<th>X</th>
<th>Colors</th>
<th>HR2</th>
<th>HR1</th>
<th>HR0</th>
<th>CRES1</th>
<th>CRES0</th>
<th>Bytes/line</th>
</tr>
</thead>
<tbody>
<tr>
<td>640</td>
<td>4</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>160</td>
</tr>
<tr>
<td>640</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>512</td>
<td>4</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>128</td>
</tr>
<tr>
<td>512</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>320</td>
<td>16</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>160</td>
</tr>
<tr>
<td>320</td>
<td>4</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>320</td>
<td>2</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>256</td>
<td>4</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>160</td>
<td>16</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>40</td>
</tr>
</tbody>
</table>

Old SAM modes work if CC Bit set. HR and CRES are Don't Care in SAM mode. Note the correspondence of HR2 HR0 to the text mode's bytes/line. -Kev

FF9A Border Palette Register (XX00 0000 = CoCo 1/2 compatible)
FF9B Reserved

FF9C Vertical Fine Scroll Register
FF9D Screen Start Address Register 1 (bits 18-11)
FF9E Screen Start Address Register 0 (bits 10-3)
FF9F Horizontal Offset Register

Bit 7 - horizontal offset enable bit (128 char width always)
Bit 6 - X6 ... offset count (0-127)
Bit 5 - X5 for column scan start.
Bit 4 - X4
Bit 3 - X3
Bit 2 - X2
Bit 1 - X1
Bit 0 - X0

If Bit 7 set & in Text mode, then there are 128 chars (only 80 seen)/line. This allows an offset to be specified into a virtual 128 char/line screen, useful for horizontal hardware scrolling on wide text or spreadsheets.
INSIDE 0S9 LEVEL II
Reference
Section 1

FFA0-AF  MEMORY MANAGEMENT UNIT (MMU)
FFA0-A7  Task #0 Map Set (8K block numbers in the 64K map)
FFA8-AF  Task #1 Map Set (Task map in use chosen by FF91 Bit 0)

Each register has 6 bits into which is stored the block number 0-63 ($00-$3F) of the
Physical 8K RAM block (out of 512K) that you wish to appear at the CPU Logical address
corresponding to that register.

Also can be shown this way: the 6 register bits, when the Logical Address in the range of
that register, will become the new Physical RAM address bits:

18 17 16 15 14 13

MMU Register:  CPU:
Task0  Task1  Logical Address / Block#  
FFA0  FFA8  0000 - 1FFF  0  The 6-Bit Physical Block Number
FFA1  FFA9  2000 - 3FFF  1  placed in a MMU register will
FFA2  FFAA  4000 - 5FFF  2  become the A13-A18 lines when
FFA3  FFAB  6000 - 7FFF  3  the corresponding Logical Add
FFA4  FFAC  8000 - 9FFF  4  is accessed by the CPU.
FFA5  FFAD  A000 - BFFF  5
FFA6  FFAE  C000 - DFFF  6
FFA7  FFAF  E000 - FFFF  7

Ex: You wish to access Physical RAM address $35001. That Address is:

A-18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
      ...3... ...5... ...0... ...0... ...0... ...1...  
      0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Taking address bits 18-13, we have: 0 1 1 0 1 0 0, or $1A, or 26. This is the physical RAM
block number, out of the 64 (0-63) available in a 512K machine.

Now, let's say you'd like to have that block appear to the CPU at Logical Block 0 (0000-
1FFF in the CPU's 64K memory map).

You would store the Physical Block Number ($1A) in either of the two Task Map registers
that are used for Logical Block 0 (FFA0 or FFA8). Unless your program doing this is in the
Vector RAM at FEXX (set FF90 Bit 3, so ALWAYS there), you would want to use your current
Task Map Register Set. If the TR bit at FF91 was 0, then you'd use MMU register FFA0 for the
$1A data byte.

To find the address within the block, use Address Bits 12-0 plus the Logical base address
(which in this case is $0000):

Now you could read/write address $1001, which would actually be $35001.

7-1-4
FFB0-BF  COLOR PALETTE REGISTERS  (6 bits each)
  FFB0  -  palette 0
  FFB1  -  palette 1  The pixel or text attribute bits in video memory
  ...  form the address of a color palette (0-15).
  FFBF  -  palette 15  It is the color info in that palette which is seen.

Reg bits-  5  4  3  2  1  0
CMP ... I1 I0 P3 P2 P1 P0  Intensity and Phase (16 colors x 4 shades)
RGB ... R1 G1 B1 R0 G0 B0  Red Green Blue  (64 RGB combo's)

When CoCo Bit is set, and palette registers preloaded with certain default values (ask, if
you need these), both the RGB and CMP outputs appear the same color, supposedly.

40/80 Column Text Screen Bytes are Even-char, Odd-attribute, in memory.
Characters selected from 128 ASCII.  NO text graphics-chars.

  Char Attributes- 8 bits... F U T T TB B B
                   Flashing, Underline, Text foregrnd, Backgrnd colors 0-7.
------------------------------------------------------------------------

FFC0-DF  SAM : same as before (mostly compatible Write-Only Switches)
  FFD8  =  CPU .895 MHz  (no address-dependent speed)
  FFD9  =  1.79 MHz
  FFDE  =  Map RAM/ROM
  FFDF  =  all RAM
------------------------------------------------------------------------
ADDENDUM

This is an addendum to the GIME information.
Thanks to Greg Law and his friend Dennis Weldy for much register info.

GIME Register Corrections:

$FF91 - Bit 5, Timer Input Select. Looks like 0=slower speed, instead. Haven't had time to put a scope on it to check actual clocks, yet. Not sure.

$FF92-3 - Interrupt Request Regs: You can also read these regs to see if there is a LOW on an interrupt input pin. If you have both the IRQ and FIRQ for the same device enabled, you read a Set bit on both regs if that input is low.

For example, if you set $FF02=0 and $FF92=2, then as long as a key is held down, you will read back Bit 1 as Set.

The keyboard interrupt input is generated by simply AND'ing all the matrix pins read back at $FF00. Therefore, you could select the key columns you wished to get by setting the appropriate bits at $FF02 to zero. Pressing the key drops the associated $FF00 line to zero, causing the AND output to go low to the GIME. Setting $FF02 to all Ones would mean only the Joystick Fire buttons would generate interrupts.

$FF94-95 - Storing a $00 at $FF94 seems to stop the timer. Also, apparently each time it passes thru zero, the $FF92/93 bit is set without having to re-enable that Int Request.

$FF98 - Bit 5 is the artifact color shift bit. Change it to flip Pmode 4 colors. A One is what is put there if you hold down the F1 key on reset. POKE &HFF98, &H13 from Basic if your colors artifact the wrong way for you.

$FF9F - Horz Offset Reg. If you set Bit 7 and you're in Gfx mode, you can scroll across a 128 byte picture. To use this, of course, you'd have to write your own gfx routines. On my machine, tho, an offset of more than about 5 crashes.

$FFB0-BF - As I originally had, and we all know by now, FFB0-B7 are used for the text mode char background colors, and FFB8-BF for char foreground colors, in addition to their other gfx use.

CoCo-3 Internal Tidbits:

The 68B09E address lines finally have pullup resistors on them. Probably put in for the 2MHz mode, they also help cure a little-known CoCo phantom: since during disk access, the Halt line tri-states the address, data, and R/W lines, some old CoCo's would float those lines right into writing junk in memory. Now $FFFF would be presented to the system bus instead.

Since the GIME catches the old VDG mode info formerly written to the PIA at $FF22, those four now-unconnected lines (PB4-7 on the 6821) might have some use for us.
INSIDE 0S9 LEVEL II
Reference
Section 1

Also, Pin 10 of the RGB connector is tied to PB3 on the same PIA. Shades of
the Atari ST. Could possibly be used to detect type of monitor attached, if we
like.

Data read back from RAM must go thru a buffer, the GIME, and another buffer.
Amazing that it works at 2 MHz.

In case you didn't catch the hint from GIME.TXT on FF90 Bit 2, the option of
an internal SCS select opens up the possibility of a CoCo-4 with a built-in
disk controller.

GIME PINS:

<table>
<thead>
<tr>
<th>61 63 65 67 01 03 05 07 09</th>
<th>09 ------- 01 68 ------- 61</th>
</tr>
</thead>
<tbody>
<tr>
<td>60 62 64 66 68 02 04 06 08</td>
<td>10</td>
</tr>
<tr>
<td>58 59</td>
<td>1</td>
</tr>
<tr>
<td>56 57</td>
<td>1</td>
</tr>
<tr>
<td>54 55</td>
<td>1</td>
</tr>
<tr>
<td>52 53</td>
<td>1</td>
</tr>
<tr>
<td>Bottom</td>
<td>Top</td>
</tr>
<tr>
<td>50 51</td>
<td>1</td>
</tr>
<tr>
<td>48 49</td>
<td>1</td>
</tr>
<tr>
<td>46 47</td>
<td>1</td>
</tr>
<tr>
<td>44 45 42 40 38 36 34 32 30</td>
<td>26</td>
</tr>
<tr>
<td>28 26</td>
<td>26</td>
</tr>
<tr>
<td>43 41 39 37 35 33 31 29 27</td>
<td>27</td>
</tr>
</tbody>
</table>

| 01 - GND        | 18 - D6   | 35 - +5 Volts | 52 - A13 |
| 02 - XTAL      | 19 - D7   | 36 - Z3      | 53 - A14 |
| 03 - XTAL      | 20 - FIRQ* | 37 - Z4    | 54 - A15 |
| 04 - RAS*      | 21 - IRQ*  | 38 - test (+5)| 55 - VSNC* |
| 05 - CAS*      | 22 - CART* Int in | 39 - Z5 | 56 - HSYNC* |
| 06 - E         | 23 - Keybd* Int in | 40 - Z6 | 57 - D7 (RAM) |
| 07 - Q         | 24 - RS232* Int in | 41 - Z7 | 58 - D6 |
| 08 - R/W*      | 25 - A0 (fm CPU) | 42 - Z8 | 59 - D5 |
| 09 - RESET*    | 26 - A1   | 43 - A4 (fm CPU) | 60 - D4 |
| 10 - WEn* 0    | 27 - A2   | 44 - A5      | 61 - D3 |
| 11 - WEn* 1    | 28 - A3   | 45 - A6      | 62 - D2 |
| 12 - D0 (CPU)  | 29 - S2   | 46 - A7      | 63 - D1 |
| 13 - D1        | 30 - S1   | 47 - A8      | 64 - D0 |
| 14 - D2        | 31 - S0   | 48 - A9      | 65 - Comp Vid |
| 15 - D3        | 32 - Z0 (RAM) | 49 - A10 | 66 - Blue |
| 16 - D4        | 33 - Z1   | 50 - A11     | 67 - Green |
| 17 - D5        | 34 - Z2   | 51 - A12     | 68 - Red  |

Notes: WEnx = Write Enables for Banks 0 and 1 RAM
S2-0 = (address select code -> 74LS138):

| 000 -0- ROM 010 -2- FF0X, FF2X | 100 -4- int SCS | 110 -6- norm SCS |
| 001 -1- CTS 011 -3- FF1X, FF3X | 101 -5- n/a | 111 -7- ??ram?? |

7-1-7
CONNECTORS:
(CN5,6 - top to bottom, CN2 - left to right)

CN6 - Gnd, +5, D1, D0, D2, D3, D6, D7, D5, D4, WEn1, Gnd
CN5 - Gnd, D2, D3, D1, WEn0, D0, CAS, D6, D5, D4, D7, Gnd
CN2 - Gnd, RAS, Z0, Z1, Z2, Z3, Z6, Z5, Z4, Z7, Z8, Gnd

Tho as far as the CN's go, even if I have messed up all but the CAS, RAS, WEn's, and +5, you could connect the extra RAM Dx and Zx pins in parallel to each bank in any order. Most RAM's don't care.

CN6 and CN5 data lines go to separate 256K banks, of course.

General Info:

Data is written to the RAM by byte thru IC10 or IC11, selected by WEn 0 or 1.
(write enable 0 = even addresses, write enable 1 = odd addresses)
Two bank RAM data is read back to the GIME thru IC12 & IC13, byte at a time.
The CPU can then get it from the GIME by byte.
IC 10, 11, 12 = 74LS244 buffer. IC13 = 74LS374 latch clocked by CAS* rise.
RAM Read --> IC12 --> GIME enabled by CAS low. (read first)
RAM Read --> IC13 --> GIME enabled by CAS hi. (latched & read)

Test Points:
TP 2 = E    TP 4 = RAS    TP 6 = Comp Video    TP 9 = Green
TP 3 = Q    TP 5 = CAS    TP 8 = Red        TP10 = Blue
--- IRQ POLLING TABLE ---

A list of 9-byte entries, one for each device controller / driver that has used the FS$Irq call. When an IRQ comes, IOMAN uses this list to find the device that is requesting service.

IOMAN then JSR's to the driver's interrupt routine, which is expected to clear the IRQ, and do whatever I/O is required. The driver normally will wake up V.WAKE, the process that was using the device. (The driver had put the process to sleep.)

--- DEVICE TABLE ---

When a device is first called upon, IOMAN inserts quick reference info about the device in the table, and calls the device's INIT subroutine that first time only.

Table used by IOMAN for making path desc's & calling the device's file mgr; by file mgr to call device's driver.

--- MODULE DIRECTORY ---

Table of modules in memory, at 00A00-00FFF. Contains info on their physical address, and used by OS9 for quick lookup of module names. Also used to keep track of the number of users.

--- PATH DESCRIPTORS ---

Each open path has a Path Descriptor, which is shared by all processes that got the path desc by ISDup'ing a path, or by having the path passed to it by the FSFork call, which dup's the first 3 standard path's of the parent to the child.

The desc block number is NOT the number you use in a program to access the path. The block number is stored in the process desc I/O path table in the order in which the paths are opened (they take the first empty spot found in the proc path table).

Your number is simply an index into the path desc I/O table in your process descriptor, which is then used by IOMAN to get the real path desc block number.

The base address of all path desc's is in D.PthDBT.

---
Entry Format IRQ POLLING TABLE

Q$POLL 00-01 Polling address (status byte)
Q$FLIP 02 Flip byte for negative logic
Q$MASK 03 Mask byte for IRQ bit
Q$SRV 04-05 Service routine
Q$STAT 06-07 Static storage address
Q$PRTY 08 Priority of device
POLL_SZ . Size of each entry

Entry Format DEVICE TABLE

V$DRIV 00-01 Driver module
V$STAT 02-03 Driver static storage
V$DESC 04-05 Descriptor module
V$FMGR 06-07 File manager module
V$USR 08 Device user count
DEV_SZ . Size of each entry

Entry Format MODULE DIRECTORY

MD$MPDAT 00-01 Module's block(s) DAT Image Pointer
MD$MBSZ 02-03 Memory Block Size
MD$MPTR 04-05 Offset pointer in block to module
MD$LNK 06-07 Module Link Count

Block Format PROC/PATH DESCRIPTORS

Descriptors (process/path) are allocated in 64-byte blocks, out of 256-byte pages. The very first block is dedicated as pointers to this and any other pages needed to hold the max # of descriptors in use.

00-3F MSB's of pages allocated to this type of descriptor
40-7F Descriptor #1
80-BF Descriptor #2
C0-FF Descriptor #3

Therefore, byte $01 in the first page above points to the next page of four 64-byte blocks:

00-3F Descriptor #4
40-7F Descriptor #5
80-BF Descriptor #6
C0-FF Descriptor #7

The descriptor # is used as the proc ID / path pointer by the system. If the descriptor is not in use (killed/closed), the first byte of the block is cleared as a flag, else it is equal to the descriptor number itself.
MODULE TYPES

$10 Prgrm Program module  $C0 Systm System module
$20 Sbrtn Subroutine mod  $D0 F1Mgr File manager
$30 Multi Multi-module  $EO Drivr Device driver
$40 Data Data module  $F0 Devic Device descriptor

UNIVERSAL MODULE HEADER

M$ID  00-01 Sync bytes ($87CD)
M$Size  02-03 Module size
M$Name  04-05 Offset from start to module name
M$Type  06 Type / language nibbles
M$Revs  07 Attributes / revision nibbles
M$Parity  08 Header parity
...... Rest of header, program, and CRC value.

INIT MODULE

Maxmem  00-08 Universal header
Maxmem  09-0B Top of free memory
PollCnt  OC IRQ polling table max entry count
DevCnt  0D Device table max entry count
InitStr  0E-0F Startup module name offset ('CC3GO')
SysStr  10-11 Default device name offset ('/D0')
StdStr  12-13 Standard I/O pathlist ('/TERM')
BootStr  14-15 Bootstrap module name ('Boot')
ProtFlag  16 Write-protect enable flag
...... Name strings

PROGRAM MODULE

00-08 Universal header
M$Exec  09-0A Execution entry offset
M$Mem  0B-0C Data memory size required
...... Program

SUBROUTINE MODULE

00-08 Universal header
M$Exec  09-0A Subroutine entry point (may be elsewhere)
M$Mem  0B-0C Stack space required (optional for pgm use)
...... Subroutine(s)
FILE MANAGER

00-08 Universal header
M$Exec 09-0A Offset to Execution Entries Table
Name string, etc.

Execution Entries Table (all LBRA xxxx)

FMCREA 00-02 Create new file
FمونNB 03-05 Open file
FMMDIR 06-08 Make directory
FMCDIR 09-0B Change directory
FMDLET 0C-0E Delete file
FMSEEK 0F-11 Seek position in file
FMREAD 12-14 Read from file
FMWRIT 15-17 Write to file
FMRLLN 18-1A Read line with editing
FMWRLLN 1B-1D Write line with editing
FMGSTA 1E-20 Get file status
FMSTTA 21-23 Set file status
FMCLSC 24-26 Close file
File manager program

DEVICE DRIVER

00-08 Universal header
M$Exec 09-0A Offset to Execution Entries Table
M$Mem 0B-0C Static storage required
M$Mode 0D Driver mode capabilities
Name string, etc.

Execution Entries Table (all LBRA xxxx)

D$INIT 00-02 Initialize device
D$READ 03-05 Read from device
D$WRIT 06-08 Write to device
D$GSTA 09-0B Get device status
D$PSTA 0C-0E Put device status
D$TERM 0F-11 Terminate device
Device driver program
## DEVICE DESCRIPTOR

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00-08</td>
<td>Universal header</td>
</tr>
<tr>
<td>09-0A</td>
<td>File manager name offset for this device</td>
</tr>
<tr>
<td>0B-0C</td>
<td>Driver name offset</td>
</tr>
<tr>
<td>0D</td>
<td>Device capabilities</td>
</tr>
<tr>
<td>0E-10</td>
<td>Device extended address</td>
</tr>
<tr>
<td>11</td>
<td>Number of options in initialization table</td>
</tr>
<tr>
<td>12</td>
<td>Device type 0=SCF 1=RBF 2=PIPE 4=NFM</td>
</tr>
<tr>
<td>13-</td>
<td>Initialization table (copied to path desc)</td>
</tr>
<tr>
<td>..</td>
<td>Name strings</td>
</tr>
</tbody>
</table>
VIDEO DISPLAY CODES

All codes are hex (natch) and are sent to the desired device window.
(see also pages 20 on, in September 86 RAINBOW for examples)
Parameters with H** L** parts are the High (msb) and Low (lsb) bytes.
Device windows are the /Wx's, overlay windows go within device windows.
Visible screens will change to the one containing the current active window.
(each displayable screen can have several windows in it)

-----------------------------------------------

DWSSET 1B 20 STY CPX CPY SXX SZZ PRN PRN (PRN)
Device Window Set - set up a device window (/Wx)

DWEND 1B 24
Device Window End

SELECT 1B 21
Select Active Window - send this code to the device window whose screen you
wish to become visible and the new active keyboard user.

OWSET 1B 22 SVS CPX CPY SXX SZZ PRN PRN
Overlay Window Set - set up an overlay window within a device window

OWEND 1B 23
Overlay Window End

CAREA 1B 25 CPX CPY SXX SZZ
Change Window Area - changes active window portion

Notes:
/Wx - up to 31 windows, plus /W and /TERM
CPX CPY - starting char col & row
SXX SZZ - size in rows & cols
PRN - palette register number (00-0F)
SVS - save switch (0=no, 1=yes) to save data under OW
STY - window screen type
 0 = current type: allows multiple windows in a screen
 1 = 40x24 text
 2 = 80x24 text
 5 = 640x192 two color gfx
 6 = 320x192 four color
 7 = 640x192 four color
 8 = 320x192 sixteen color

-----------------------------------------------

DEFGPBUF 1B 29 GRP BFN HBL LBL
Define Get/Put Buffer - preset a buffer size

KILBUF 1B 2A GRP BFN
Kill Buffer - return buffer to free mem

GPLD 1B 2B GRP BFN STY HXS LSX HSY LSY HBL LBL DATA...
Get/Put Buffer Load

GETBLK 1B 2C GRP BFN HBX LBX HBY LBY HXS LSX HSY LSY
Get Graphics Block
PUTBLK  1B 2D GRP BFN HBX LBX HBY LBY
Put Graphics Block

Notes:
GRP    - Get/Put Buffer Group Number 00-FE
BFN    - Get/Put Buffer Number 01-FF (within Group)
HBL/LBL - 16 bit length
-SX -SY - size X Y
-BX -BY - buffer X Y

Get/Put Groups and their Buffer subsets are used to store screen data, fonts, and pattern ram info.
Certain Group numbers are pre-defined as reserved, or as fonts, patterns, etc. Within those Groups, specific Buffer numbers are set aside.
For your own use, you should do an F$ID call to get your process id, kill the group, then open it for your use. This keeps things separated.

The standard Groups and Buffers within those groups:
C8 - fonts
  01 - 8x8 font
  02 - 6x8 font
  03 - 8x8 gfx

C9 - clipboards

CA - pointers
  01 - arrow
  02 - pencil
  03 - large cross-hair
  04 - wait
  05 - stop!
  06 - text()
  07 - small cross-hair

CB - patterns (2 color)
  01 - dot
CC - patterns (4 color)
  02 - vertical lines
CD - patterns (16 color)
  03 - horz lines
  04 - cross-hatch
  05 - left slant
  06 - right slant
  07 - small dot
  08 - big dot

PSET  1B 2E GRP BFN Pattern Set - select buffer as pattern ram array
LSET  1B 2F LCD Logic Set - select mode for pattern display
  0 = store data on screen as is
  1 = AND pattern data w/screen data
  2 = OR
  3 = XOR

7-3-2
INSIDE OS9 LEVEL II
Reference
Section 3

DEFCLR 1B 30  Default Color Reset
PALETTE 1B 31 PRN CTN  Set Palette Reg
FCOLOR 1B 32 PRN  Foreground Color - use palette # PRN
BCOLOR 1B 33 PRN  Background Color - use palette # PRN
BORDER 1B 34 PRN  Border Color - use palette # PRN

Notes:
CTN  color (00-3F RRRGGGBBBL xslated by monitor type)

SCALESW 1B 35 BSW  Scaling - 01 = drawing is relative to window size
DWPRTSW 1B 36 BSW  Window Protect Switch (boundary detection)
GCSET 1B 39 GRP BFN  Set source of Graphics Cursor data
FONT 1B 3A GRP BFN  Select Font - previously loaded into buffer
BCHRSW 1B 3B BSW  Block Char - draw char font as full char block
TCHRMSW 1B 3C BSW  Transparent Char - draw char dots only
BOLDSW 1B 3D BSW  Boldface Char
PROPSW 1B 3F BSW  Proportional

Notes:
BSW  option switch (00=off, 01=on)

CURSOR 1B 40 HBX LBX HBY LBY  RCURSOR 1B 41 (Relative Coords)
POINT 1B 42 HBX LBX HBY LBY  RPOINT 1B 43 - use relative coords
LINE 1B 44 HBX LBX HBY LBY  RLINE 1B 45 HBXo LBXo HBYo LBYo
LINEM 1B 46 HBX LBX HBY LBY  RLINEM 1B 47 for these cmds
BOX 1B 48 HBX LBX HBY LBY  RBOX 1B 49 ...
BAR 1B 4A HBX LBX HBY LBY  RBAR 1B 4B ...
PUTGC 1B 4E HBX LBX HBY LBY
FPILL 1B 4F
CIRCLE 1B 50 HBR LBR
ELIPSE 1B 51 HBRx LBRx HBRy LBRy
ARC 1B 52 HBRx LBRx HBRy LBRy HX01 LX01 HY01 LY01 HX02 LX02 HY02 LY02
RARC 1B 53 HBRxo " " etc

Other Terminal Codes:

HOME 01  ERASEEOS 0B
GO XY 02  CLSHOME 0C
ERASELINE 03  RETURN <CR> 0D
ERASEEOL 04  REVERSE 1F 20
CURSOROFF 05 20  REVERSEOFF 1F 21
CURSORON 05 21  UNDERLINEON 1F 22
RIGHT 06  UNDERLINEOFF 1F 23
BELL 07  BLINKON 1F 24
LEFT 08  BLINKOFF 1F 25
UP 09  INSLINE 1F 30
DOWN 0A  DBLLINE 1F 31

7-3-3
## Keyboard Definitions with Hex Values

<table>
<thead>
<tr>
<th>Norm</th>
<th>Shift</th>
<th>Ctrl</th>
<th>Norm</th>
<th>Shift</th>
<th>Ctrl</th>
<th>Norm</th>
<th>Shift</th>
<th>Ctrl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>0</td>
<td>40</td>
<td>60</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>C</td>
<td>A</td>
<td>41</td>
<td>a</td>
<td>61</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>E</td>
<td>C</td>
<td>43</td>
<td>c</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>25</td>
<td>G</td>
<td>D</td>
<td>44</td>
<td>d</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5</td>
<td>45</td>
<td>E</td>
<td>E</td>
<td>46</td>
<td>e</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>F</td>
<td>F</td>
<td>46</td>
<td>f</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>G</td>
<td>G</td>
<td>47</td>
<td>g</td>
<td>67</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>8</td>
<td>58</td>
<td>H</td>
<td>H</td>
<td>48</td>
<td>h</td>
<td>68</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>9</td>
<td>0</td>
<td>I</td>
<td>I</td>
<td>49</td>
<td>i</td>
<td>69</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>J</td>
<td>J</td>
<td>4A</td>
<td>j</td>
<td>6A</td>
</tr>
<tr>
<td>;</td>
<td>2</td>
<td>A</td>
<td>0</td>
<td>K</td>
<td>K</td>
<td>4B</td>
<td>k</td>
<td>6B</td>
</tr>
<tr>
<td>,</td>
<td>2</td>
<td>C</td>
<td>1</td>
<td>L</td>
<td>L</td>
<td>4C</td>
<td>l</td>
<td>6C</td>
</tr>
<tr>
<td>.</td>
<td>2</td>
<td>E</td>
<td>0</td>
<td>M</td>
<td>M</td>
<td>4D</td>
<td>m</td>
<td>6D</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>F</td>
<td>0</td>
<td>N</td>
<td>N</td>
<td>4E</td>
<td>n</td>
<td>6E</td>
</tr>
<tr>
<td>&lt;</td>
<td>0</td>
<td>CLR</td>
<td>0</td>
<td>O</td>
<td>O</td>
<td>4F</td>
<td>o</td>
<td>6F</td>
</tr>
<tr>
<td>&gt;</td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<CLR><0> = shift u/l case
<control>-;

The only new key code generated is the 7F rubout key.
System Error Codes

001 01 Exit
002 02 Keyboard abort
003 03 Keyboard interrupt
200 E$PathFul C8 Path Table full
201 E$PathNum C9 Bad Path Number
202 E$Poll CA Polling Table Full
203 E$Mode CB Bad Mode
204 E$DevOvf CC Device Table Overflow
205 E$SID ID CD Bad Module ID
206 E$DirFul CE Module Directory Full
207 E$MemFul CF Process Memory Full
208 E$UnkSvc D0 Unknown Service Code
209 E$ModBusy D1 Module Busy
210 E$HPAddr D2 Bad Page Address
211 E$EOF D3 End of File
212 D4 Attempt to return memory not assigned
213 E$Nes D5 Non-Existing Segment
214 E$FNA D6 File Not Accessible
215 E$BadPath D7 Bad Path Name
216 E$PNNF D8 Path Name Not Found
217 E$SLF D9 Segment List Full
218 E$CEF DA Creating Existing File
219 E$IBA DB Illegal Block Address
220 E$HangUp DC Carrier lost
221 E$MNF DD Module Not Found
222 E$E SP DF Deleting Stack Pointer memory
223 E$IPrcID E0 Illegal Process ID
224 E$NoChild E2 No Children
225 E$ISwi E3 Illegal SWI code
226 E$PrAbt E4 Process Aborted
227 E$PrcFul E5 Process Table Full
228 E$UnkFork E6 Illegal Fork Parameter
229 E$KnownMod E7 Known Module
230 E$MCRC E8 Bad Module CRC
231 E$USigB E9 Unprocessed Signal Pending
232 E$NEMod EA Non Existing Module
233 E$NName EB Bad Name
234 E$INHP EC Bad module header parity
235 E$NoRam ED No Ram Available
236 E$PrcID EE Bad Process ID
237 E$NoTask EF No available Task number
240 E$Unit FO Illegal Unit (drive)
241 E$Sec F1 Bad SECTOR number
242 E$WP F2 Write Protect
243 E$CRC F3 Bad Check Sum
244 E$Read F4 Read Error
245 E$Write F5 Write Error
246 E$NotRdy F6 Device Not Ready
247 E$Seek F7 Seek Error
248 E$Full F8 Media Full
249 E$BTyp F9 Bad Type (incompatible) media
250 E$DevBusy FA Device Busy
251 E$IDC FB Disk ID Change
252 E$Lock FC Record is busy (locked out)
253 E$Share FD Non-sharable file busy
254 E$Deadlk FE I/O Deadlock error

7-5-1
INDEX

A Closer Look 4-1-3
A Small Problem 5-1-1
ADDENDUM 7-1-6
AREAS OF INTEREST 4-1-7
An Analogy 1-3-3
BUGS- Hardware 5-3-2
BUGS- Manual 5-3-3
BUGS- Software 5-3-1
Basic Interrupt Handling 2-4-1
CC310 4-1-2
CC310 4-1-3
CHARACTERS FONT 4-3-1
Clock INIT & Operation 2-4-3
CoCo-3 Internal Tidbits 7-1-6
Comparison with Other I/O Devices 4-1-1
Connectors 7-1-8
Dat 1-5-1
Dat Images & Tasks 1-4-1
Each User Task Map 1-5-2
Example Maps 1-5-6
Example One 1-5-6
Example Two 1-5-7
F$Link from system state 5-5-2
F$Load from system state 5-5-2
FONT CONVERSION 5-4-1
Forking RUNB modules 5-5-2
GENERAL DRIVER NOTES 3-5-1
GIME Register Corrections 7-1-6
GRFDRV 4-1-3
GRFDRV 4-1-5
GRFINT 4-1-4
GRFINT/WINDINT 4-1-3
General Information 7-1-8
General Notes viruses end up as irq 2-4-5
Gime Dat 1-3-1
I/O 1-2-3
ICMAN INIT 2-4-2
IOMAN IRQ Poll Synopsis 2-4-3
IRQ Handling 2-4-1
IRQ's On Level Two 3-5-2
IRQ's- Clock & Devices 2-4-1
IRQ- Related DP.Vars & System Tables 2-4-5
Information 5-1-1
Killing Window Processes 5-1-2
L-II Direct Page Variable Map $00XX 2-1-2
L-II Process Descriptor Variables 2-1-1
Level II Task Switching 2-4-2
Level Two Device Addresses 3-5-1
Level Two In More Detail 1-5-1
Level Two vs One: General 1-5-1
Level-I CoCo 1/2 2-4-1
Level-I CoCo 3 2-4-1
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical vs Physical Addresses</td>
<td>1-3-1</td>
</tr>
<tr>
<td>Login II Patch</td>
<td>5-5-3</td>
</tr>
<tr>
<td>Making Windows</td>
<td>5-2-1</td>
</tr>
<tr>
<td>Memory Mgmt</td>
<td>1-3-1</td>
</tr>
<tr>
<td>Merged Module Files</td>
<td>5-5-1</td>
</tr>
<tr>
<td>Misc.</td>
<td>5-1-3</td>
</tr>
<tr>
<td>Misc. Window Tips</td>
<td>4-1-6</td>
</tr>
<tr>
<td>Multi-tasking Principles</td>
<td>1-2-1</td>
</tr>
<tr>
<td>Multi-tasking Principles</td>
<td>1-2-2</td>
</tr>
<tr>
<td>Multi-tasking Principles</td>
<td>1-2-3</td>
</tr>
<tr>
<td>Multi-tasking Principles</td>
<td>1-2-3</td>
</tr>
<tr>
<td>Now I Have It, What's Its Use?</td>
<td>1-3-4</td>
</tr>
<tr>
<td>OS9 Boots</td>
<td>5-5-1</td>
</tr>
<tr>
<td>OS9 System Calls</td>
<td>2-2-1</td>
</tr>
<tr>
<td>OS9 Vector Initialization</td>
<td>2-4-2</td>
</tr>
<tr>
<td>Ok, What About OS9 Level-II</td>
<td>1-3-5</td>
</tr>
<tr>
<td>Other L-II Driver Changes</td>
<td>3-5-2</td>
</tr>
<tr>
<td>Other System Ram Usage</td>
<td>2-1-4</td>
</tr>
<tr>
<td>Overview of OS9</td>
<td>1-1-2</td>
</tr>
<tr>
<td>Process Information</td>
<td>1-1-2</td>
</tr>
<tr>
<td>Process quenes</td>
<td>1-2-2</td>
</tr>
<tr>
<td>Program Modules</td>
<td>1-1-2</td>
</tr>
<tr>
<td>RBF Things</td>
<td>3-5-2</td>
</tr>
<tr>
<td>SCF Special Chars.</td>
<td>3-5-1</td>
</tr>
<tr>
<td>SHELL</td>
<td>5-1-1</td>
</tr>
<tr>
<td>Screen Memory</td>
<td>4-1-5</td>
</tr>
<tr>
<td>Signals</td>
<td>1-2-4</td>
</tr>
<tr>
<td>Simple System Memory Map</td>
<td>1-1-2</td>
</tr>
<tr>
<td>States</td>
<td>1-2-2</td>
</tr>
<tr>
<td>Swi's</td>
<td>1-5-1</td>
</tr>
<tr>
<td>Switching Between Maps</td>
<td>1-5-5</td>
</tr>
<tr>
<td>System Memory Allocation</td>
<td>1-5-4</td>
</tr>
<tr>
<td>THE WINDOW DRIVERS</td>
<td>4-1-1</td>
</tr>
<tr>
<td>TIPS, GOTCHAS &amp; LAST MINUTE STUFF</td>
<td>5-5-1</td>
</tr>
<tr>
<td>The CoCo-2 Boards</td>
<td>1-3-1</td>
</tr>
<tr>
<td>The CoCo-3 Dat</td>
<td>1-3-2</td>
</tr>
<tr>
<td>The Main Players</td>
<td>1-1-3</td>
</tr>
<tr>
<td>The System Map</td>
<td>1-5-2</td>
</tr>
<tr>
<td>The System Task Map</td>
<td>1-5-2</td>
</tr>
<tr>
<td>Too Much To Say</td>
<td>1-3-5</td>
</tr>
<tr>
<td>Transfer To System State-Level I/II</td>
<td>2-4-2</td>
</tr>
<tr>
<td>Universal System Tables</td>
<td>1-1-2</td>
</tr>
<tr>
<td>User Maps Module &amp; Data Areas</td>
<td>1-5-4</td>
</tr>
<tr>
<td>Using L-I Debug On Level Two</td>
<td>5-5-3</td>
</tr>
<tr>
<td>Using L-I VDG Programs</td>
<td>5-5-1</td>
</tr>
<tr>
<td>Using Rogue to Make a System Disk</td>
<td>5-2-1</td>
</tr>
<tr>
<td>VDGINT</td>
<td>4-1-2</td>
</tr>
<tr>
<td>VIDEO DISPLAY CODES</td>
<td>7-3-1</td>
</tr>
</tbody>
</table>
Congratulations on your purchase of the first of what we hope will be many books on the most powerful operating system available, OS-9!

This copy of 'INSIDE OS9 LEVEL II' will be upgraded as time goes by and as more information is collected.

To get your discounted copy of the next edition of 'INSIDE OS9 LEVEL II' fill out the coupon below and mail it in.

---

**COUPON**

This coupon entitles the bearer to edition 2 of 'INSIDE OS9 LEVEL II' when it becomes available for 50% of the cover price of this copy. The price of $20.00 includes COD UPS charges. This coupon is good for shipment in the USA only.

Name__________________________________________
Address(no PO BOX)_________________________________
City________________________State________ZIP__________
Date________________________

Mail to:
Frank Hogg Laboratory, Inc.
770 James Street
Syracuse, New York 13203
DON'T LET YOUR FINGERS DO THE WALKING!

Typing in the long listings from INSIDE OS9 LEVEL II can be a real pain. By ordering INSIDE OS9 LEVEL II DISK you can eliminate that hardship. The disk includes the programs in the book plus many other nifty programs by the author.

The INSIDE OS9 LEVEL II DISK is just $20.00 and contains all the programs in the book plus others by the author.

You can use your Visa, MasterCard, American Express or Diners Club card. or you can enclose payment and mail your order to:

Frank Hogg Laboratory, Inc.
770 James Street
Syracuse, New York 13203
315/474-7856

YES! Send me the INSIDE OS9 LEVEL II DISK

Name________________________________________
Address_____________________________________
City________________________________________ State__________ ZIP______

Payment Enclosed ____ Charge my Visa account____ Charge my MasterCard account____ Charge My American Express account(5% surcharge)_____ Charge my Diners Club account(5% surcharge)_____

Account No._________________________ Card Expires________________________

Signature________________________________________ Interbank No.______________

Non-US orders add $2.00 US to cover additional shipping. NY State residents add 7% sales tax.
Frank Hogg Laboratory is the largest supplier of software for OS9. We have also been supporting OS9 longer than any other company. If you would like to get on our mailing list to receive free newsletters and catalogs please fill out the coupon below and mail it in.

Thank You

MAILING LIST COUPON

YES! Please put me on your mailing list to receive free newsletters and other information about OS9.

My interest is:____________________________________

Name____________________________________________

Address___________________________________________

City_____________State_______ZIP_______________

Date______________

Mail to:
Frank Hogg Laboratory, Inc.
770 James Street
Syracuse, New York 13203