Apr 1st, 8:00 AM

PCOS - An Operating System for Modular Applications

V. P. Tharp

Electronic Engineer Electronic Branch Information Systems Directorate NASA, John F. Kennedy Space Center, Florida

PCOS - AN OPERATING SYSTEM FOR MODULAR APPLICATIONS

V. Phillip Tharp
Electronic Engineer
Electronic Branch
Information Systems Directorate
NASA, John F. Kennedy Space Center, Florida

ABSTRACT

This paper is an introduction to the PCOS operating system for the MC68000 family processors. Topics covered are: development history; development support; rational for development of PCOS and salient characteristics; architecture; and a brief comparison of PCOS to UNIX.

INTRODUCTION

PCOS is a versatile, highly modular, and user configurable, operating system for computers utilizing Motorola 68000 family microprocessors. Its main features are:

- multitasking
- an expandable unified i/o system (interrupt driven)
- inter-process communications (signals, mail boxes, and queues)
- support for interrupt driven scheduling of processes
- support for process level handling of interrupts
- extensive support for modular programing
- Supports a subset of the UNIX C library
- written in assembly language for speed of operation and minimum size

HISTORY

PCOS was developed by NASA to fulfill its internal needs for an operating system for use in control system applications. In the past, when a microprocessor was used to implement a control system, an operating system was not used. A significant part of the development process was spent in developing functions normally provided by an operating system. In addition, since the application was not developed in an environment that encouraged and supported modular programming, modification of the resulting applications software was difficult. New requirements that deviated much from the original requirements tended to force a complete redesign of the software. With the advent of processors of the MC68000 class, and cheap, dense RAM and ROM, an operating system that supported (enforced ?) modular programing was possible and desirable.

In 1983, when development of PCOS began, an operating system existed on the market for the MC6809 microprocessor. OS9 had many of the features considered desirable for NASA's operating system. However, OS9 was written in assembly language and was therefore not available for the MC68000 processor. Microware, the developer of OS9, had plans to port OS9 to the MC68000, but not within the schedule constraints for PCOS. (As happens in NASA, PCOS's development was tied to another project.) The decision was made, with Microware's permission, to use OS9 as the baseline specifications for PCOS.

The first version of PCOS, 1.04, was released for use in the field on September 21, 1984. This version implemented the basic functions provided by OS9 plus an inter-process communications queueing system. On November 15, 1985, version 1.10 was released. This version implemented a mail box system, process level debugger, new utilities written in "C" (with greatly expanded functions), many internal changes for speed and better structure, and bug fixes found during the

year of operational use. Since that time, development has continued up to the present. A version of the PCOS kernel and bios has just been finished, but not yet released, that supports memory protection hardware. Plans call for development of a disk file manager; windowing file manager; uniform graphics interface; X.25 file manager; and support for a tightly coupled multiprocessor environment.

DEVELOPMENT SUPPORT

The current version of PCOS does not provide a disk operating system though one is scheduled for development in 1986. Until a disk operating system is released, applications for PCOS must be developed on a host system. Cross development tools (C compiler, assembler, linker) exist for use under the OS9, UNIX, and VMS operating systems. A programmer's guide is available by contacting the author.

WHY THIS OPERATING SYSTEM?

This is, of course, a key question and a complex one. However, a more general question to ask first is why use any operating system? The answer is that operating systems provide standard services, particularly I/O, that would otherwise have to be implemented for each application. In fact, the earliest operating systems began as nothing more than collections of subroutines that were found to be essentially common from one application to the next.

What then, is so special about PCOS? There is no single feature, taken alone, that would answer that question. In fact, many of the features of PCOS were first introduced in other operating systems, especially UNIX. What PCOS does is combine many previously developed ideas into an operating system that is especially suited as an operating environment for the type of software developed at KSC - control system software. The following sections attempt to exhibit these ideas.

MULTI-TASKING

When an application has many inputs and outputs with little or no synchronization between them, it is best to break the application up into independent pieces. This is called modular programming. Each piece handles a specific interface or function. The pieces must then be scheduled for execution when needed. Some of the pieces will need to communicate with each other. The development and testing of software to provide execution scheduling and communications between the different pieces of an application, is a difficult and time consuming process. A multi-tasking operating system provides these services. Most importantly, it provides these services for the general case and in a standard manner.

In a multi-tasking operating system, each piece becomes a process and shares time on the CPU with other processes based on some type of scheduling criteria. The operating system provides various methods for processes to communicate with each other. A very important advantage is that each process can (and should) be tested separately. Another advantage is that new requirements that are data driven can be implemented as new processes. This is much simpler than attempting to squeeze new code into an existing monolithic program.

PCOS is a multi-tasking operating system. It is technically called a hybrid. This term implies that PCOS supports two types of scheduling criteria - round robin and event driven. Round robin scheduling implies that all active processes share the CPU based on their priority. Those processes with higher priority execute more often than lower priority processes. Event driven scheduling implies that a process get control of the CPU based on the occurrence of an event, usually an interrupt. This dual nature of PCOS allows processes that need to execute in real-time to do so while allowing non real-time process to time-share the CPU.

INTER-PROCESS COMMUNICATIONS

Once an application has been broken up into processes, methods must be provided for the processes to communicate with each other. Since many pieces of an application can be data driven, inter-process communication implies inter-process synchronization. PCOS provides three type of inter-process communications; signals, queues, and mailboxes.

Signals are an asynchronous control mechanism used for inter-process communication and control. A signal behaves like a software interrupt in that it can cause a process to suspend a program, execute a specific routine, and afterward return to the interrupted program. Signals can be sent from one process to another process or they can be sent from PCOS routines to a process. Status information can be conveyed by the signal in the form of a word value. Some of the signal "codes" have pre-defined meanings, but all the rest are user-defined.

Inter-process communication queues are provided under PCOS to allow processes to exchange data in fixed size blocks. A queue is a software implementation of a FIFO. Each queue is divided into elements. IO to a queue is done one element at a time. If a queue is full, and a process wants to place an element on a queue, the process can be suspended from execution until the queue has room for the element. In a like manor, a process wanting to take an element from a queue can be suspended until an element is placed on the queue. Queues serve as IO and synchronization between data driven processes. The number of queues, number of elements in each queue and size of the elements for each queue are determined by values in the INIT module (see architecture below). Queues provide a very low overhead means of inter-process IO.

Mailboxes are queues that hold one element of one word each. They are primarily used for process synchronization. The number of mailboxes in a PCOS system is specified in the INIT module. Mail boxes are numbered from 0 to N-1, where N is the number of mailboxes. The value stored in a mailbox may range from 1 to $FFFF (hex). A value of 0 is not allowed.

MEMORY MODULES

PCOS uses the concept of the memory module. Memory modules attempt to support modular programming by providing a standard way of storing and accessing code and global data. Memory modules consist of a header, body, and CRC check code. They are required to be position independent and in the case of code, not modify themselves. A name, type, and revision is assigned to each memory module. Reference to a memory module is by its name and type. PCOS provides the necessary mechanisms for accessing and managing memory modules. A properly written code memory module is reentrant and can be used by multiple processes at the same time.

One very important aspect of memory modules is their support of configuration control. PCOS uses the memory module's revision information to distinguish between a current memory module and an outdated one. The CRC information is used to insure that the memory module is good.

IO SYSTEM

PCOS provides a unified IO system. At the process level all devices appear the same. IO is performed in terms of streams of bytes. PCOS makes no assumptions at the process interface as to the type of IO device. This type of system makes processes more testable. A process designed to read values from an A to D converter can be tested by reading values from a terminal keyboard without changing the process. (Its IO is redirected when the process is started.)

PCOS also provides access contention so that multiple processes can use the same IO device concurrently.

New components can be added to the IO system to support virtually any type of device. This means that most device control software can be implemented as extensions to the IO system further supporting modular programming.

EXTENDIBILITY

PCOS is highly modular and as such, supports extensions to itself. As mentioned above, the IO system can be extended. New system calls can be added or existing system calls replaced by extensions to the PCOS kernel. Even the kernel itself can be replaced. PCOS is not meant to be a fixed system. It main purpose is to set standards for implementing operating system services and applications in a modular fashion.

SUPPORT

Control system software tends to break operating systems. This type of software puts strain on three areas that are usually weak in an operating system, interrupts, inter-process communications, and process synchronization. The reason is simple, these areas represent the most difficult parts of an operating system to get working right. The result is that control system software designers need close support from the operating system vendors. At the least, access to full source code is mandatory. Access to real experts on the operating system is almost as important, for if this expertise is not available, the control system designers have to become experts on the operating system. Usually this happens in one of two ways. Either they spend the time up front and learn the operating system, or, as is usually the case, they learn it the hard way - when it breaks.

Since PCOS was developed by NASA, access to source code is readily available. As long as the development team is intact, technical support is available. In addition, since this is a non-commercial environment, the development group has nothing to hide, so the PCOS documentation should be more informative. Eventually the documentation and the source code should be sufficient for most support requirements.

ARCHITECTURE

PCOS is composed of memory modules, each of which provides a specific function. These memory modules can be divided into functional groups.

The first group consists of the SYSTEM DEBUGGER, KERNEL, INIT and CLOCK modules. This group is the minimum configuration required for a PCOS system. The SYSTEM DEBUGGER module determines the location and size of RAM and ROM in the system and passes this information to the kernel at startup time. The SYSTEM DEBUGGER module also provides a system level debugger for debugging PCOS itself. The kernel provides basic system services such as multi-tasking, memory management, maintenance and control of memory modules, interprocess communication, and access to functions provided by other PCOS modules. The CLOCK module is a software handler for the specific real-time clock hardware. INIT is an initialization table used by the kernel during system startup to specify data structure sizes, the name of the initial process to run, whether the I/O system is being used, and other information. Note that this group does not include the unified I/O system although the kernel does contain the high level interface and control routines. A PCOS system built from this group alone would require the application processes to provide their own I/O.

The next group of modules implements the unified I/O system. The I/O system is structured in a hierarchy consisting of 3 types of memory modules. Each type of memory module represents one level in the hierarchy. File managers are the highest level and process I/O requests for similar classes of devices. As an example, the Sequential Character File Manager (SCFMAN) will handle almost any device that operates on characters in a character-serial fashion, such as a terminal or a printer. Device drivers are the next level. They handle low level I/O functions such as read and write, init and terminate for a specific kind of device or interface chip. A device driver is usually designed to work with only one file manager. Device descriptors are the lowest level. These modules are small tables that associate specific I/O ports with their logical names and the port's device driver and file manager. The name of the device is the name of the device descriptor. They also contain the physical address of the port and initialization data. By use of device descriptors, only one copy of each driver is required for each specific type of I/O device regardless of how many devices of that type the system uses. The user can add additional file managers, device drivers, and device descriptors to handle classes of devices not covered by the modules provided with PCOS.

The utilities package is the next logical group of PCOS modules. Each utility is an independent program that runs as a process under PCOS just like any application process and provides some service to an operator. The utilities are all designed to work with a standard ASCII terminal. Included are:

Table 3-1: LIST OF UTILITIES

Shell	-	a command interrupter that provides a man machine I/F to system services such as the ability to start and stop processes
Procs	-	a program for statusing the other processes that have been started
Mdir	-	displays the module directory much like a directory of a disk

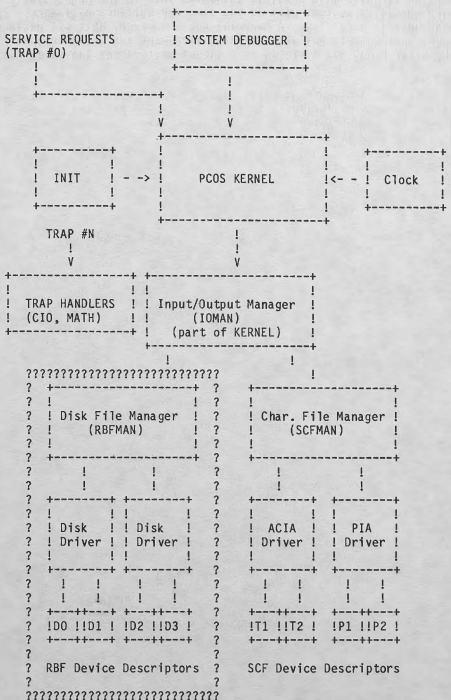
Mfree - displays the current amount of free RAM
Setime - sets the system real-time clock
Date - displays system date and time in gregorian and julian formats
Tmode - allows customization of the way the operator's terminal works such as page pause
Debug - provides symbolic debugging capability for individual processes (breakpoints, trace, single step, disassembly)

Note that the utilities are not required to run PCOS.

The last group of memory modules consists of PCOS extensions. These memory modules provide additional system calls or functions to PCOS. One type of extension modules provided for use with programs written in C are trap handlers. These modules make available to all C programs the functions provided by the C library. Since the code size for some of these functions is quite large, having only one copy of the library in the system results in considerable savings in storage. For example, any program wanting to make use of the C function "printf" would call the "CIO" trap handler. Another type of extension module adds new system calls to the kernel. The name of the module, if it exists, is specified in the init module.

The following is a block diagram of PCOS:

Table 3-2: PCOS COMPONENT MODULE ORGANIZATION



PCOS VERSES UNIX

Standardization is a very important issue in software design today. Just as the development of 7400 series integrated circuits made possible a quantum leap in the productivity of logic designers, standard operating systems will do the same for software designers. There is no standard yet though UNIX is very close to becoming one. However, UNIX was designed as a software development environment, not as a target environment and was developed roughly 15 years ago. PCOS, on the other hand, was designed as a target environment and makes use of current technology.

Many of PCOS' features were derived from UNIX. Most of PCOS' system calls have close relatives in UNIX. The entire standard C library is implemented under PCOS. The primary difference between UNIX and PCOS is in modularity. In most implementations of UNIX, the entire operating system is one monolithic program. PCOS, on the other hand, is constructed of memory modules. In addition, UNIX is not designed to be placed in ROM; will not operate without a disk; and is not designed for real-time processing.

REFERENCES

PCOS is the product of the efforts of Priscilla Stanley, Lee Hornyak, and Phil Tharp of NASA, Kennedy Space Center.