Peripheral Processor Routines and Requests

Author: Mark Riordan
Original Content: 60bits.net

PP Code vs. CPU Code

In the early days of CDC operating systems, much of the operating system was coded on PPs. I'm not sure of the reason for this, especially since the PP instruction set was quite awkward. I'm sure that some motivating factors were:

- The desire to keep central memory free for user programs.
- The CPU's inability to do I/O. (But many OS responsibilities have nothing to do with I/O.)
- The fact that there were more PPs than CPUs.

Most CDC machines had only a single CPU, but the 6500 had two. This meant that at MSU, there were only 5 times as many PPs as CPUs, not 10 times as at most other sites. For this and other reasons, Michigan State implemented many of its enhancements in CPU code, and in fact moved many existing OS functions from PP code to CPU code.

For the OS “kernel” itself (though that word was never used), both CPU and PP components were done entirely in assembly language. Furthermore, routines were written to reside at particular memory locations. (There was PP relocatable code, but it was awkward and not used much, and had to follow very strict conventions.) PPs did not have an RA register to bias addresses, and CPU OS code often ran with RA=0. (A number of CPU OS routines did run with RA not equal to 0. They had all sevens for their FL and referenced low core tables by negative addresses, wrapping around memory.)

As I recall, there was a limited amount of overlaying done with CPU OS code. But if you like overlays, PPs are the place for you.

Overlays and PP Memory Layout

There were two basic types of PP programs: those that used the standard “PP Resident” library, and those that did not. PP Resident was a very tightly coded collection of useful utility routines used by nearly all PP programs. A few important PP programs were so space-critical that they were written without the benefit of this library. PP Resident was inexplicably named STL; some people guessed that might stand for SysTem Library. Around 1980 (rather late in the game), STL was recoded by MSU systems programmer Mary Kestenbaum, squeezing out a few more bytes and making the code more elegant. I deeply regret having lost my listing of MCK's STL. I can no longer remember most of the functions PP resident provided. I do remember that it packed a lot of functionality into 700 octal 12-bit bytes, the equivalent of 672 decimal 8-bit bytes.

All PP programs reserved locations 0 - 77B as direct cells. For PP programs that used STL, STL was located in locations 100B - 777B, with the program itself starting at 1000B. Other programs simply started at 100B.

Because memory was so limited in PPs, many PP programs were written using overlays. By convention, overlays were written to load at a multiple of 1000 octal. PP programs were given names with 3 characters, and by convention the first character was a digit representing where the program

Retro1.org - https://codex.retro1.org/
should load (the address divided by 1000B). Main overlays typically loaded at 1000B, so many programs had names that started with 1. For instance, 1AJ (Advance Job) was called when a command in a job was completed and the next control card needed to be read, parsed, and executed. Child overlays loaded at higher locations, so their names started with bigger digits, such as 4.

**Important PP programs**

Notable PP programs included:

**MTR: System monitor**

This was the most important program in the system. In our version of the OS, some of the functionality was moved to a corresponding CPU program called CPUMTR, but MTR was still a very tightly coded program. I can't remember what all of its responsibilities were, but one that I do remember was keeping track of the time. CDC CPUs resembled early PCs in that they did not come with a time-of-day clock, but they did have access to a source of precisely-timed pulses. One hardware channel was reserved for read-only access to a 12-bit counter which constantly incremented. Each time through its loop, MTR would read this special channel and see whether the counter was smaller than last time. If so, the counter must have overflowed. MTR knew how often the counter overflowed and used this information to update a date-and-time data structure in memory. OS requests for the time-of-day were handled by giving them a copy of this data structure, just as PC BIOSes do today. The software time-of-day clock was slow because MTR code assumed that 1024 microseconds equaled one millisecond. However, the time wasn't as slow as it would have otherwise been, because every second, MTR loaded 1MN, which handled all the complexity of updating minutes, hours, day, month, and year. 1MN included some code to help compensate for MTR's inaccuracy.

At MSU, we did have a third-party hardware clock, but it was available only on the CDC 750, not the 6500. Also, in the first few years, the hardware clock was only used to set the time at bootup, similar to the way PCs work. Later, according to Glen Kime's recollection, 1MN was modified to read the hardware clock, to improve accuracy.

**DSD: Operator console**

"Dynamic System Display" was the routine that ran the operator's console. It was heavily overlayed. For a description of using DSD, see [Console Commands](#).

**DUD: User control of console**

"Dynamic User Display" was a program similar to DSD that allowed user programs to display arbitrary text or graphics on the operator console. DUD was not part of the operating system; it had to be loaded specially into the OS (and never during production hours). Because in normal usage, DSD was essential to being able to run the system, DUD, which replaced DSD while it was running, had to implement some of DSD's functionality.

The only times I ever knew DUD to be used was when Pete Poorman and I would play Northwestern University's CHESS 3.0 late at night. CHESS 3.0 was one of the best chess-playing programs in its day,
though by the time Pete and I were playing it, it was no longer the most recent version. CHESS 3.0 displayed a pretty convincing image of a chess board on the console, and accepted commands from the keyboard. There was also a timesharing version, usable via teletype.

One of the few times I've been on television was when I showed CHESS 3.0 to a chess camp for small kids. A local TV station was on hand to gather footage for a color story on the camp. I'm sure most of those 10-year-olds could have whupped me in a regular game, but in the loud computer room, surrounded by massive boxes of computer equipment and important-looking adults, the kids turned into morons. I had to pretty much play the game myself.

A few years later, I purchased a IBM PC chess program written by David Slate, one of the authors of CHESS 3.0. I didn't play it a lot, because it required you to boot the computer with the chess floppy. In other words, the game was an OS, not an application. I guess that wasn't too unusual in the early days of PC games.

1SP: Disk I/O

The above PP routines were unusual in that they hogged a PP; once loaded, they stayed running in that PP forever. Most PP programs were transient. They were loaded into a PP, did their work in typically a fraction of a second, and then the PP was marked as available for loading another PP program. PP resident (STL) was responsible for the loading.

There was one important PP routine that was a hybrid: 1SP, the Stack Processor. 1SP did disk I/O; the significance of the words “Stack Processor” are lost on me. Responsive disk I/O was very important to system performance, of course, so the system made sure that a copy of 1SP was always loaded into at least one PP, even if there were no outstanding disk I/O requests. In fact, since there were multiple disk controllers and disk units, the system could do true simultaneous disk I/O, and therefore tried to keep multiple copies of 1SP loaded to allow this to happen. I believe that the system dynamically adjusted the number of copies of 1SP in PPs. If there was a lot of disk I/O on multiple units for a while, more copies of 1SP would be loaded. However, you wouldn't want to tie up too many PPs with idle copies of 1SP, so the number would be allowed to dwindle when the I/O load lessened. In later versions of the OS on the PP-rich 750, the system kept one copy of 1SP loaded for each disk controller.

Most PP routines were stored on disk, but the master copy of 1SP was kept in either central memory or ECS (I can't remember which). This not only made this important routine available more quickly, but also avoided the chicken-and-egg problem that would otherwise result.

PP requests

CDC operating systems implemented an unusual system call mechanism. System requests—referred to as PP requests even if no PP program was involved—were made by placing a specially-formatted word at address 1 of a program's field length (i.e., RA+1). This location was scanned periodically by MTR (or CPUMTR). When the system noticed that a job's RA+1 was non-zero, it would zero the location and start servicing the request. By convention (though I never understood why), applications would loop, waiting for RA+1 to zero both before and after issuing a request. It certainly was necessary for an application to ensure that RA+1 was zero before issuing a request, lest a previously-issued but as yet unserviced request be overwritten. But this could have been done by consistently checking either
before or after each request. Some applications executed a Program Stop (PS) instruction after placing an RA+1 request. Need Help: RRM said the bit about PS, but I don't recall this. I guess after servicing a request MTR would have had to check to see whether a job was sitting on PS and if so, increment P. I don't recall this.

In the early days, a significant amount of the system's CPU time (I seem to recall 5-10%) was spent by applications looping, waiting for the system to notice their RA+1 requests. An optional instruction, the Central Processor Exchange Jump, was available to allow an application to transfer control to the OS and have it notice the request. This XJ instruction was kind of like a software interrupt. At MSU, we had this instruction added to our CPU around 1975.