Notes and Background on Operating System for the 68000 CPU

Brent Seidel
Phoenix, AZ

February 24, 2024

Note that this is a draft version and not the final version for publication.

Contents

1 Overview

This is a collection of notes on a simple multi-tasking operating system for the 68000 CPU, temporarily names os68k. The main goals are:

1. To be able to blink lights in interesting pattern in the Pi-Mainframe (`https://github.com/BrentSeidel/Pi-Mainframe`) project,
2. To learn something about operating system design, and
3. To actually have a somewhat useful operating system.

Note that all statements are subject to change as the project develops.

This is based on the 68000 simulator part of the Sim-CPU (`https://github.com/BrentSeidel/Sim-CPU`) project. It is targeted towards a system with 16Mbytes of memory and no MMU. The memory is divided into 16 one megabyte sections, the first (lowest) section is for the operating system and each task gets one section. The gives a system that supports up to 15 user tasks plus the operating system.

Simulated devices provided by Sim-CPU that are currently used are the clock to provide a periodic interrupt for tasking and the serial-telnet port for console I/O. Simulated disks are not yet supported.

The operating system also includes a library with utility routines that can be used by the user programs.

2 Kernel

The kernel is composed of several sections. The first two are at locations that are fixed by the simulated CPU and hardware:

1. The CPU vector table starts at address 0 and contains 256 long word entries. This occupies 1 kilobyte of space.

2. The I/O port section runs from the end of the vector table at an address of 400_{16} and runs to 1000_{16}.

The remaining sections are arbitrarily arranged and will probably change with development.

1. HW_SECT contains code for interfacing with the hardware devices and related interrupt service routines.

2. OS_SECT contains the main operating system code including initialization, the clock interrupt, context save/restore, scheduling, exceptions, and system calls.

3. OS_DATA contains the operating system data tables and messages. The data tables currently include the task control blocks and the console device blocks.

4. LIB_SECT contains library routines for use by the operating system and user programs. Note that routines that use system calls shouldn't be used by the operating system, at least not yet.

5. LIB_DATA contains library data. Note that since library routines may be in use by multiple tasks simultaneously, this should be constant data only. Any variable data should be allocated on the stack.

6. OS_STACK is space for the operating system stack.

7. USR_STACK is space for the operating system user stack. The is used by the null task that runs when no other task can be run. Its stack needs are minimal.

2.1 Hardware Abstraction Layer

The currently supported hardware includes a clock that provides periodic interrupts and a terminal interface that can be accessed externally by telnet (or gtelnet).

2.1.1 Clock

The clock has a settable rate for the periodic interrupts. The rate is a byte size port which allows any value 0-255. The multiplier in the simulation is 100mS, thus to get a 1 second interrupt, the rate would be set to 10. Setting the rate to one gives a 100mS interrupt (or 10 times a second). This is a reasonable rate for multitasking. As some point, the simulation may get adjusted to allow higher rates. There is a tradeoff as higher interrupt rates give more overhead. This will require some experimentation.

2.1.2 Terminal Interfaces

Multiple terminal interfaces can be supported. Right now, each interface has its own interrupt vector, but it should be possible to have a single vector where the service routine determines which interfaces are ready.

2.2 Scheduling

Currently, a simple round-robin scheduler is used.

2.3 System Calls

Currently, all system calls are handled by `TRAP #0`. The system call number and any parameters are pushed onto the stack prior to the call. The following system calls are currently defined:

0 SYS_EXIT - Exit program

1 SYS_PUTS - Send a string to the console

2 SYS_GETC - Get a character from the console

3 SYS_PUTC - Send a character to the console

16 SYS_SLEEP - Suspend current task for a number of clock ticks

64 SYS_SHUTDOWN - Shuts the system down

3 Library

The library starts with a table of addresses of the various library routines. This allows the user programs to find the desired routine by looking in a fixed address. The code for a library call looks something like (Other registers besides %A0 can be used, but %A7 is the stack pointer and %A6 is often used as a frame pointer. The macros use %A0):

```
...   Put stuff on the stack
      move.l #LIBTBL,%A0
      move.l LIB_GETSTR(%A0),%A0
      jsr (%A0)
...   Cleanup the stack
```

Since the library routines can be preempted, the library must contain only reentrant code.

4 User Space

Each task is allocated 1 megabyte of space starting on a megabyte boundary. The initial PC is the start of the space and the initial SP is the end of the space. The user program can use this space as it sees fit.