Operating Systems 1/25/18

New instructor for the rest of the semester. Dr Yang

Note: Dr. Yang did not the board at all. His lecture is included, but all figures can be found in ch3.ppt

Programs are static- they are not dynamic. Source code is normally on disc or usb. Even once you compuke your software, it’s still static until its loaded into memory. When you double click on a program or open on the command line. When you allocate an int, the MMU will allocate it off the stack or the heap. However, your program is not the only program on the computer. Now, your operating system can manage more than one program at a time. Now, there’s a long term schedule that will decide when your program will rung (see A2, where we will be writing a simulation of this scheduler). There are CPU and I/O bounded programs. CPU bounded processes, like mathematically intense processes, spend a long time in the CPU and little time waiting for I/O. I/O bounded processes spend more time in the wait/I/O queue. Once your process has received I/O, it goes into the ready queue, waiting for CPU access.

Why do we need to know this? We can increase our program’s efficiency by

Your program must become a process before it can run. The program counter is used to keep track of where the cpu is in your process. if het cpu switches and starts processing a different process.

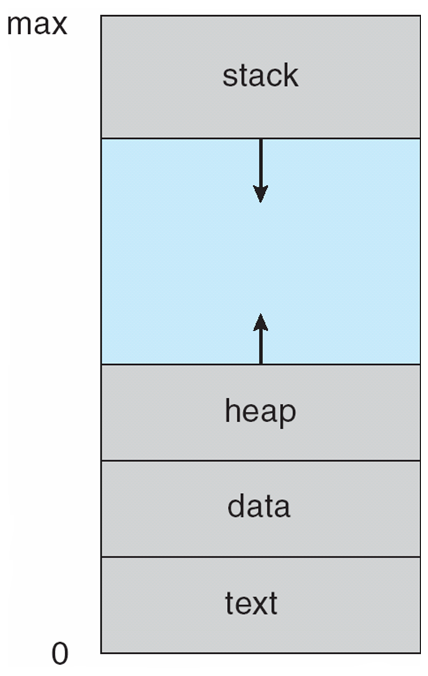
The stack stores your local variables. The heap stores your dynamically allocated variables (ie. malloc in C, new in Java). The data sector holds the global variables.

Figure 1: Visualization of memory from chp3.ppt

The system has no idea how much memory your process needs. And this memory set is specific process.

The operating stem has two goals

1. Increase the throughput of your CPU
2. Reduce the response time for I/O.

The short-term scheduler gives processes a time slice (think 100 milliseconds).

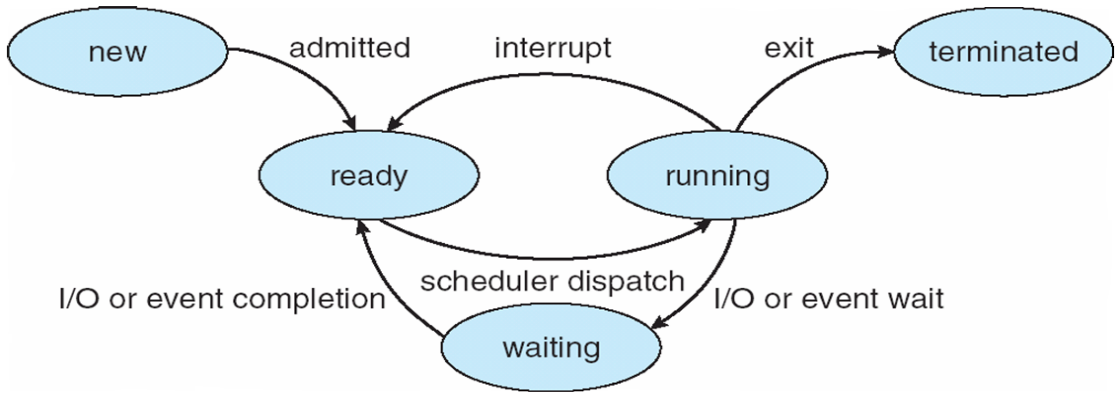
When your program terminates, all of the memory allocated from your program will be deallocated. The operating system does not zero out all of your data- instead, it is considered free, so the operating system will then offer this memory to other processes. All of the memory you fail to free yourself (for example, if you use malloc to allocate a lot of memory and do not call free) will still be freed when your program terminates.

Figure 2: A visualization of program states from chp3.ppt

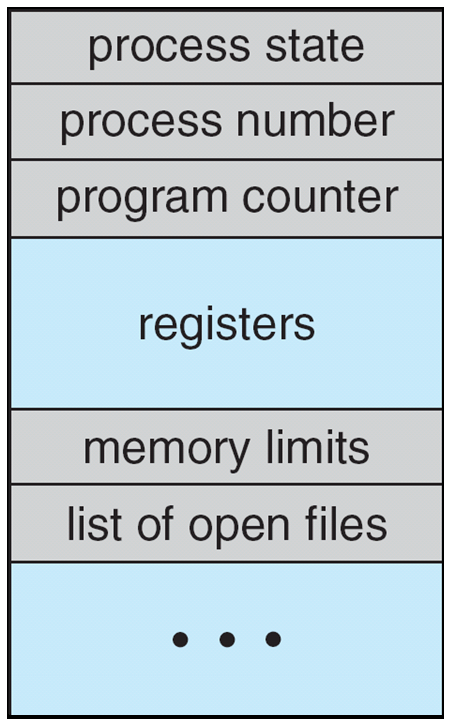


Figure 3: the PCB

Your process also uses a PCB (Process Control Block). Details can be found in chp3.ppt.

When you fork your process, your process is split into two processes running the same code. You can differentiate between parent and child processes using the pid.

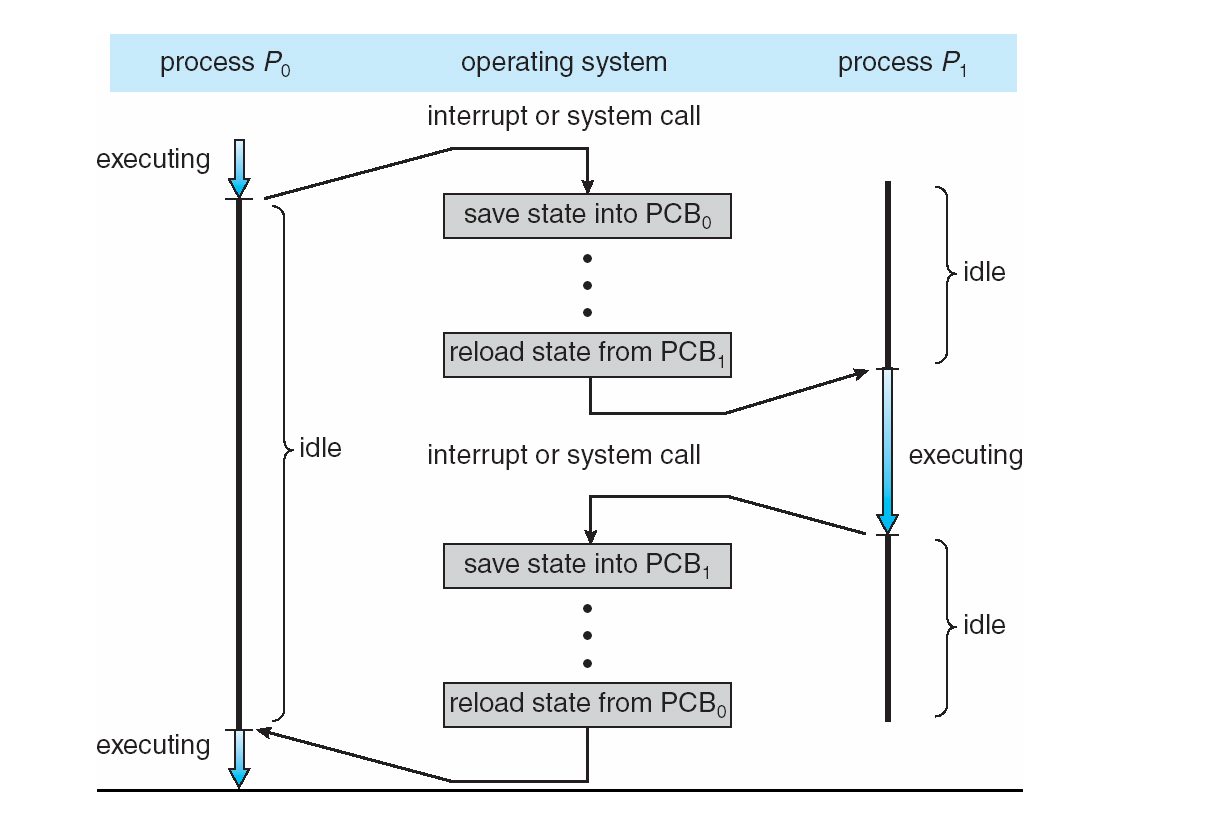
To switch between processes (called a **context switch**) is an expensive operation.

Figure 4: Context switching

Each process has each own individual memory. Threads share memory.

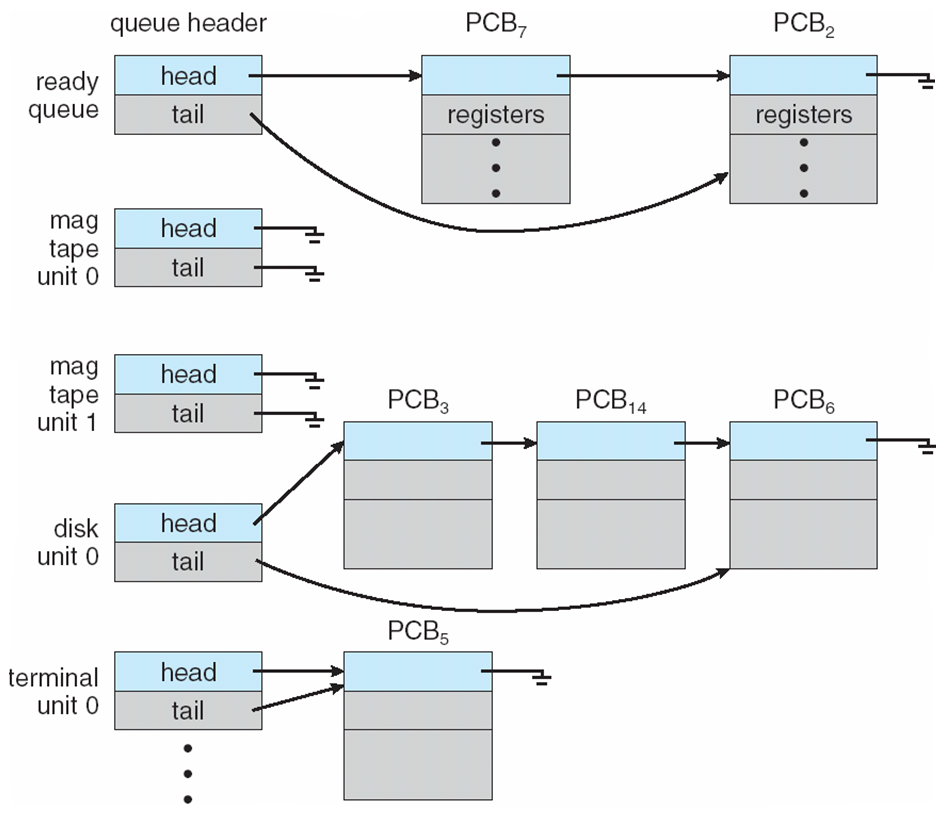
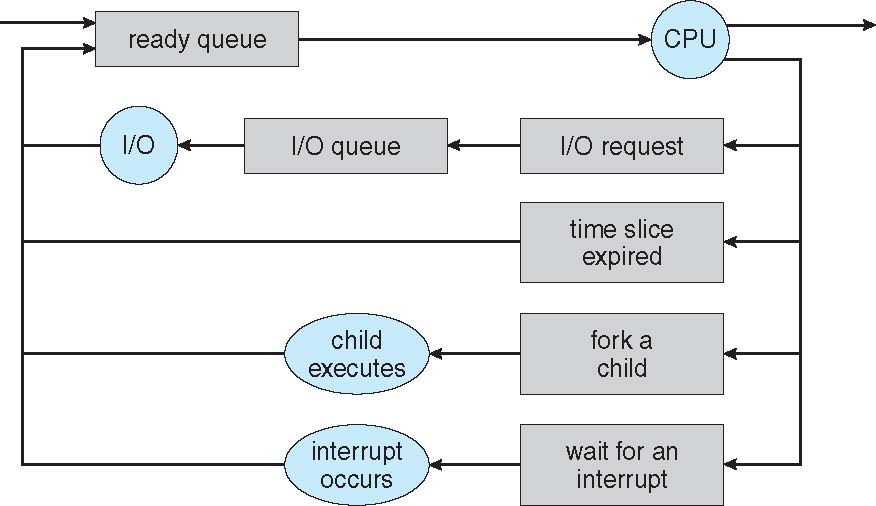
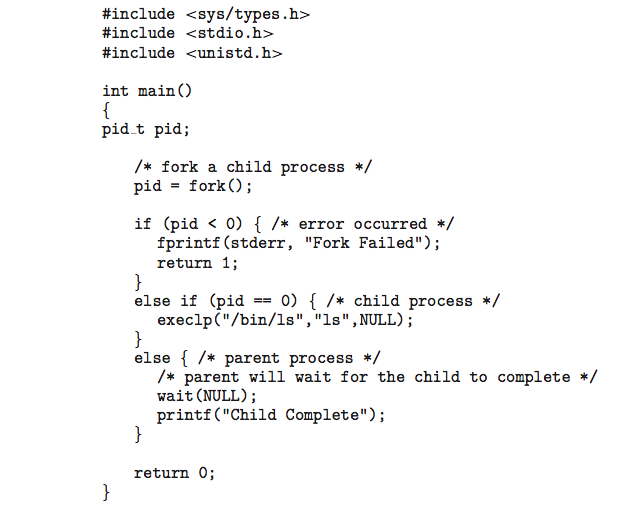


Figure 5: One visualization of the ready queue

The ready queue is (in some implementations) a linked list.



Our next program will be similar to this:[[1]](#footnote-1)



If the value of pid = 0, you are in the child process. If the pid is not 0 (and not negative), then you are the parent. We will be talking about inter-process communication.

1. This section is all review of fork + exec from Systems Programming Concepts. If you are comfortable with that, feel free to skip this. [↑](#footnote-ref-1)