CSE451 Processes
Spring 2001

Gary Kimura
Lecture #4
April 2, 2001

Today

- Quick review (our road map)
- Processes the basic computational “object” of the OS
- Conceptually
  - What formally makes up a process?
  - What does a process do (more importantly what does an OS do to a process)
- Nuts and bolts
  - The usual round of data structures and code behind the concept

But First, Your Job for Week #2

- Readings in Silberschatz
  - Chapter 4 (Monday and Wednesday lecture),
  - Chapter 5 (Friday lecture)
- Homework #2
  - Out: Today Monday April 2, 2001
  - Due: Next Monday April 9, 2001
  - Silberschatz questions 4.6, 4.7, and 5.2

What have we looked at so far

- What is an OS when it is all put together and what does it do?
- What tools are available to build the OS? (i.e., the hardware)
- What are the big pieces we need build and how do we stick them together? (i.e., how do we organize the insides)

Process Management

- Process management is the task of shepherding programs through the OS gauntlet.
- In process management there are several issues we need to deal with:
  - What is the basic entity involved (i.e., the units of execution)
  - How are those entities represented in the OS
  - How is work scheduled in the CPU
  - What are possible execution states, and how does the system move from one to another
- The term *process*, not surprisingly is the name we use to encapsulate this basic entity.

The Process

- Basic idea behind the *process* is that
  - A process is the unit of execution, it represents a running program in the system
  - It is also the unit of scheduling and resource allocation in the OS. Meaning the OS typically allocates resources to a process
  - It is the dynamic (active) execution context (as opposed to a program, which is static)
- A process is sometimes called a *job* or a *task* or a *sequential process.*

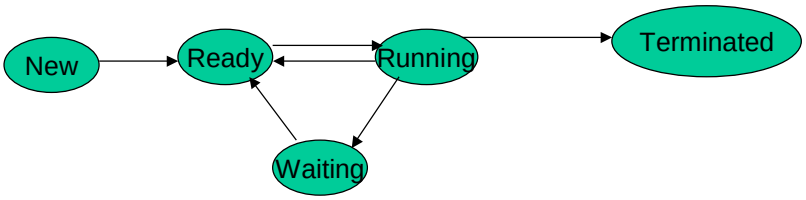
What's in a Process?

- A process consists of at least:
  - The code for the running program
  - The data for the running program (both static and dynamic)
  - An execution stack tracing the state of procedure calls made
  - The Program Counter, indicating the next instruction to execute
  - A set of general-purpose registers with current values
  - A set of operating system resources (open files, connections to other programs, etc.)
- The process contains all the state for a program in execution.

Process State

- There may be several processes running the same program (e.g., an editor), but each is a distinct process with its own representation.
- Each process has an *execution state* that indicates what it is currently doing, e.g.,:
  - Ready: waiting to be assigned to the CPU
  - Running: executing instructions on the CPU
  - Waiting: waiting for an event, e.g., I/O completion
- As a program executes, it moves from state to state

Process State Changing



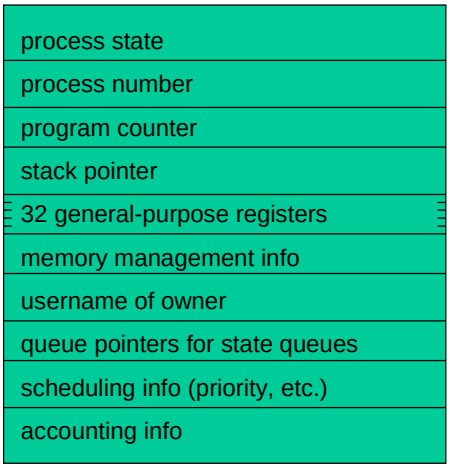
Processes move from state to state as a result of actions they perform (e.g., system calls), OS actions (rescheduling), and external actions (interrupts)

Process Data Structures

- At any time, there are many processes in the system, each in its particular state.
- The OS must have data structures representing each process. This structure is often called a *Process Control Block* or PCB for short.
- The PCB contains all of the information about a process.
- The PCB is where the OS keeps all of a process' hardware execution state (PC, SP, registers) when the process is not running. For example, when an interrupt occurs the OS stores the interrupted process' state into its PCB

PCB

The PCB contains lots of information, e.g.:



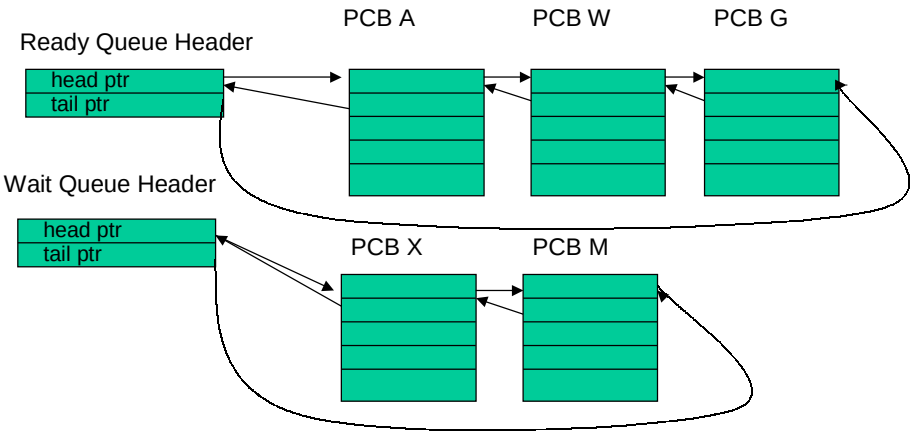
PCBs and Hardware State

- When a process is running its Program Counter, stack pointer, registers, etc., are loaded on the CPU (i.e., the processor hardware registers contain the current values)
- When the OS stops running a process, it saves the current values of those registers into the PCB for that process.
- When the OS is ready to start executing a new process, it loads the hardware registers from the values stored in that process' PCB.
- The process of switching the CPU from one process to another is called a *context switch*. Timesharing systems may do 100s or 1000s of context switches a second!

State Queues

- The OS maintains a collection of queues that represent the state of all processes in the system.
- There is typically one queue for each state, e.g., ready, waiting for I/O, etc.
- Each PCB is queued onto a state queue according to its current state.
- As a process changes state, its PCB is unlinked from one queue and linked onto another.

State Queues (Continued)



There may be many wait queues, one for each type of wait (specific device, timer, message,…).

PCBs and State Queues

- PCBs are data structures, dynamically allocated in OS memory.
- When a process is created, a PCB is allocated to it, initialized, and placed on the correct queue.
- As the process computes, its PCB moves from queue to queue.
- When the process is terminated, its PCB is deallocated.

Creating a Process

- One process can create other processes to do its work. These are *child* processes and the creator is the *parent*.
- In some systems, the parent defines (or donates) resources and privileges for its children.
- When a child is created, the parent may either wait for it to finish its task, or continue in parallel.
- In Unix, subprocesses are created by a call to Fork; the child is identical to the parent, except for a return code from Fork. The child often begins by executing a new (and different) program within itself, via a call to Exec. (Fork and Exec are Unix System Calls.)
- In bootstrapping a system there is usual a starting process

Cooperating Processes

- Processes can be independent or they can be cooperating to accomplish a single job.
- Cooperating processes can be used:
  - to gain speedup by overlapping activities or performing work in parallel
  - to better structure an application as a small set of cooperating processes
  - to share information between jobs
- Sometimes processes are structured as a pipeline where each produces work for the next stage that consumes it, and so on.

Next time

- Processes are nice, but threads are dandy