Processes

H. Levy

Process Managment

- **Process management deals with several issues:**
  - what are the units of execution?
  - how are those units of execution represented in the os?
  - how is work scheduled on the CPU?
  - what are possible execution states, and how does the system move from one to another?

Process

- **Basic idea is the *process*:**
  - process is the unit of of execution
  - the unit of scheduling
  - 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*.**
- **A sequential process is a program in execution; it defines the sequential, instruction at a time, execution of a program.**

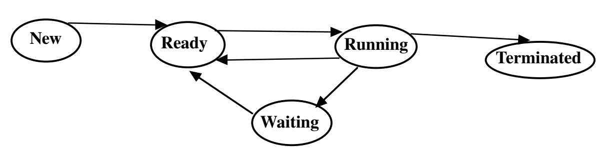
What's in a process?

- **A process consists of at least:**
  - the code for the running program
  - the data for the running program
  - an execution stack showing the state of calls made
  - the Program Counter, indicating the next instruction
  - 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's currently doing, e.g.:**
  - ready : waiting for 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), operating system actions (rescheduling), and external actions (interrupts).**

Process Data Structures

- At any time, there are many processes in the system, each in it's particular state.
- The OS must have data structures representing each process: this data structure is called the <u>PCB:</u>

  Process Control Block.
- The PCB contains all of the info about a process.
- The PCB is where the OS keeps all of a process' execution state (e.g., PC, registers) when it is not running.

PCB

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

process state (running, waiting, etc.)
process number
Program Counter
Stack Pointer
General Purpose Registers
memory management info
username of owner
list of open files
queue pointers for state queues
scheduling info (priority, etc.)
accounting info, resource usage
I/O states: I/O in progress
...other stuff...

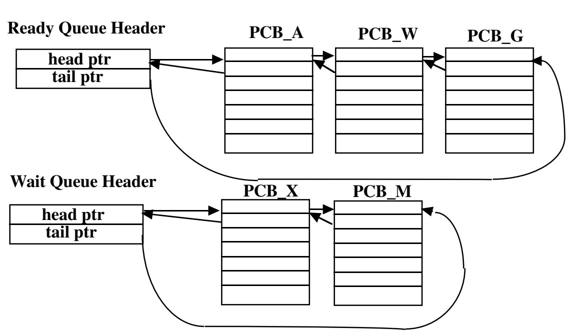
PCBs and Hardware State

- When a process is running, its Program Counter, stack pointer, registers, etc., are loaded on the CPU (i.e., the hardware registers contain the current values).
- When the OS stops a running process, it saves the current values of those registers into the PCB for that process.
- When the OS is ready to start executing a waiting 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 <u>queues</u> 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, ...
- Each PCB is queued onto a state queue according to its current state.
- As a process changes state, its PCB is unliked from one queue and linked onto another.

State Queues



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

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 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 <u>identica</u>l 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.)**

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 set of small cooperating processes
  - to share information between jobs
- **Sometimes processes are structured as a pipline where each produces work for the next stage that consumes it, and so on.**