Comp 204: Computer Systems and Their Implementation

Lecture 4: Processes(3)

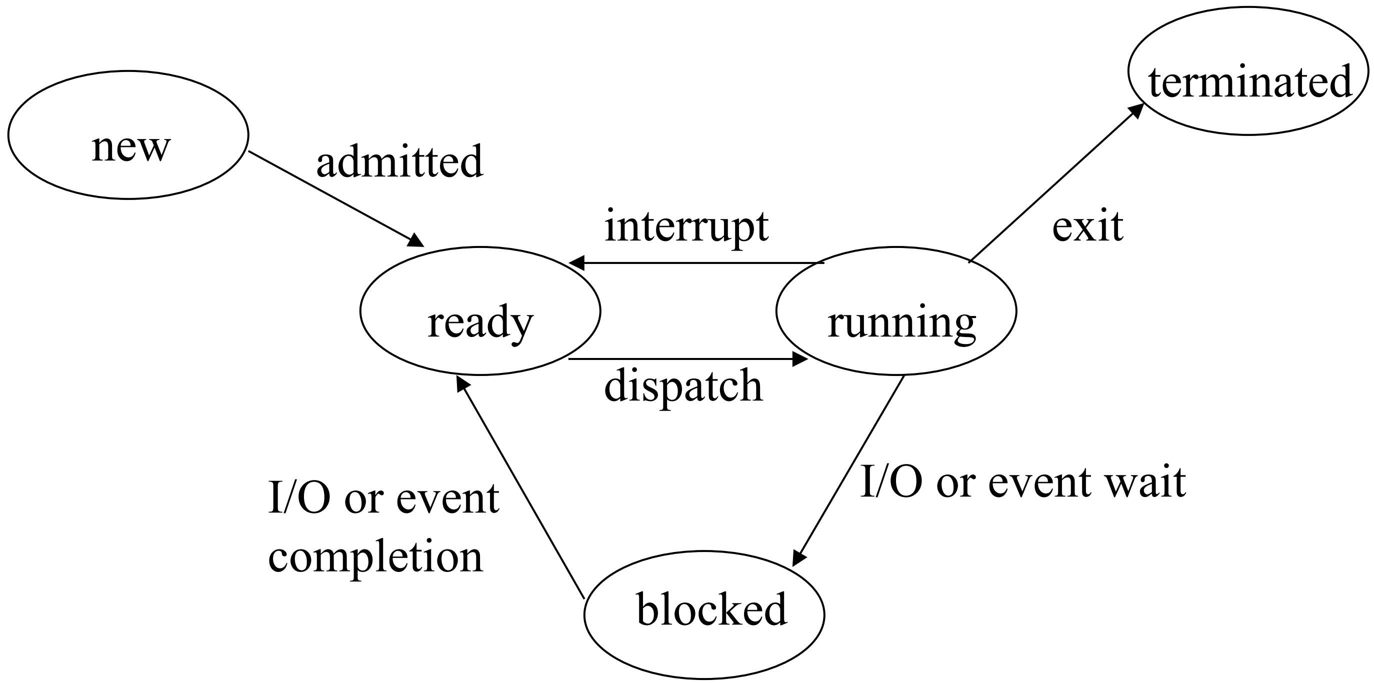
Today

- Process states
- Context switch
- Inter-process communication
  - Signals
  - Pipes

Process States

- Running
  - on a uniprocessor machine, only one process can be executing at any time
  - may be interrupted at end of time-slice if no I/O requests or system calls performed
- Ready
  - refers to a process that is able to run, but does not currently have the CPU
- Waiting(Blocked)
  - refers to a process that is unable to continue, even if granted the CPU

State Changes



Question

- If a process executes a *fork()* system call, which of the following are true?

  a) The parent process is moved to the *blocked* state
  b) The child process is placed in the *running* state
  c) The parent process is moved to the *ready* state
  d) The child process is placed in the *blocked* state
  e) The parent process is moved to the *terminated* state

Answer: c
The parent process will be moved back to the *ready* state (depending on the scheduling policy), and once the child has been admitted, will also be placed in the *ready* state

Question

- A running process makes a system call to read data from a file. Which process state should it enter next?

a) New
b) Ready
c) Running
d) Blocked
e) Terminated

Answer: d
Blocked; it may take some time before the file system can read the file (e.g. on a networked file store), so the process is blocked until the data is available.

Process Descriptors

- For each process, the OS kernel maintainers a descriptor or Process Control Block (PCB)
- PCB contains info like
  - unique process ID
  - user ID of process owner
  - process state
  - position in memory
  - accounting stats. (time used etc.)
  - resources allocated (open files, devices, etc.)
  - register values

Context Switch

- When a process is interrupted
  - all current state information (including program counter and other registers) is saved into PCB
  - PCB is put into a queue
    - may have several, e.g. for different devices
  - the kernel may do some of its own work
    - e.g. handling a system call
  - the PCB of a process from the ready queue is selected, and its context restored
- Whole context switch is an expensive overhead
  - hardware support may help
    - e.g. multiple register sets

PCBs and Queuing

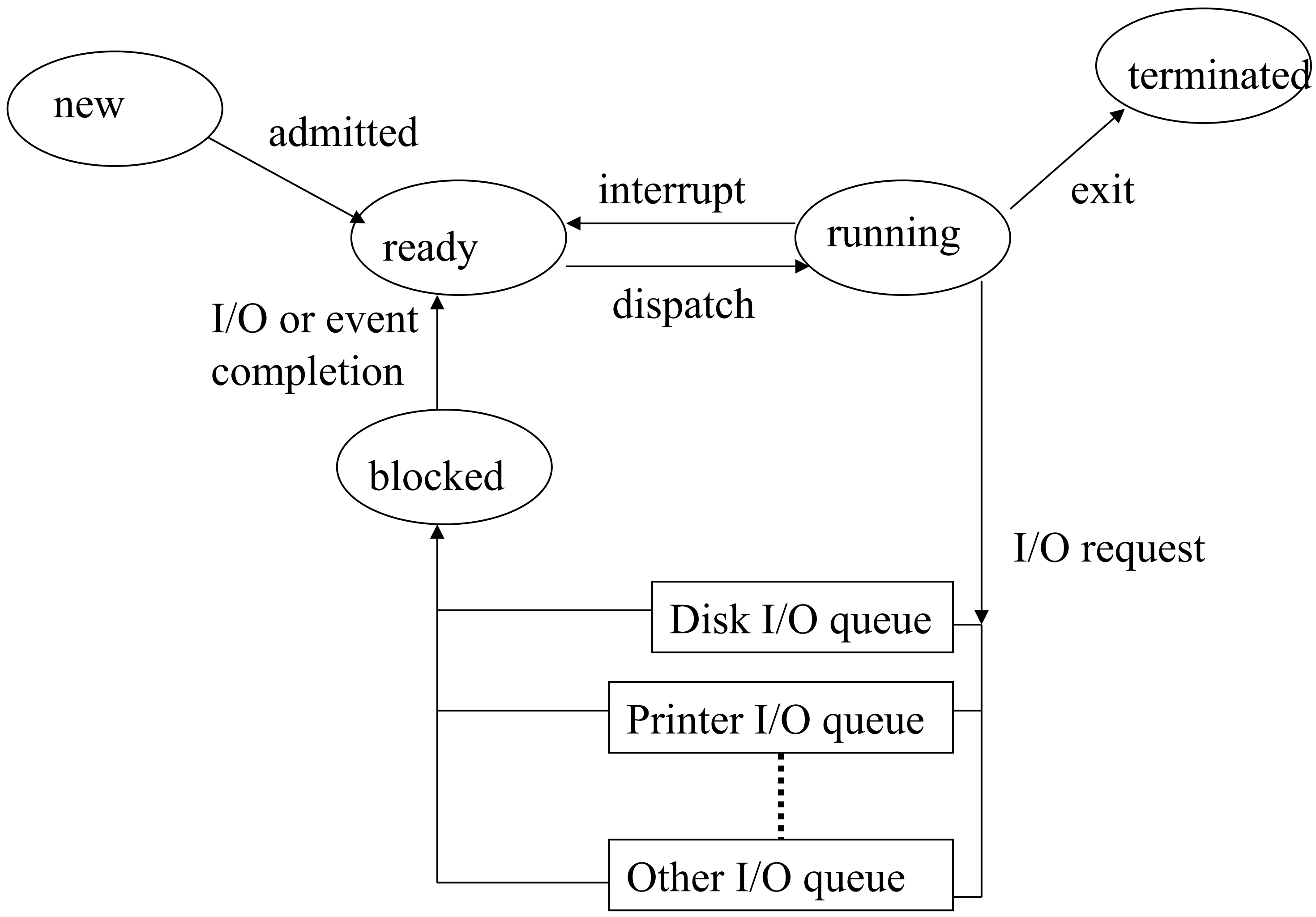
- The PCB of each process is updated as the process progresses from the start to the end of its execution

- Queues use PCBs to track the processes' progress through the system. The PCBs are linked to form queues:
  - 'Ready queue' linking the PCBs for every 'ready' process
  - 'New queue' linking the PCBs for processes just entering the system

PCBs and Queuing

- Processes that are ‘blocked’ are linked together by ‘reason for waiting’
  - PCBs for these processes are linked into several queues
    - e.g. those waiting for I/O on a specific disk drive are linked together, those waiting for a printer are linked in a different queue
- All queues need to be effectively managed in an order that is determined by the process scheduling policies and algorithms

Queuing


new
admitted
terminated
interrupt
running
exit
ready
dispatch
I/O or event
completion
blocked
I/O request
Disk I/O queue
Printer I/O queue
Other I/O queue

Inter-Process Communication

- Inter-Process Communication (IPC) mechanisms allow processes to talk to each other
- IPC useful when processes working together (cooperating processes)
  - synchronisation and/or passing data
- UNIX examples:
  - signals
  - pipes
  - sockets

Signals

- A process can usually be terminated by typing CTRL-C
  - Actually sends a signal to process
  - Process responds by aborting
- Signals can be sent from one process to another
  - signal() system call
- Signals can be sent from the command line using kill command
  - Format: kill -<signal> <pid>
  - e.g. kill -9 12345 sends signal 9 (kill signal) to process 12345

Responding to Signals

- A receiving process can respond to a signal in three ways:
  – Perform default action (e.g. abort)
  – Ignore the signal
  – 'Catch' the signal; i.e. execute a designated procedure

- The 'kill' signal (signal 9) cannot be caught or ignored
  – Guaranteed way to stop process

Example kill signals

1 HUP (hang up)
2 INT (interrupt)
3 QUIT (quit)
6 ABRT (abort)
9 KILL (non-catchable, non-ignorable kill)
14 ALRM (alarm clock)
15 TERM (software termination signal)

Pipes

- The command ‘wc –l file’ counts the number of lines in file
- If we just type ‘wc –l’ we don’t get an error
- Instead, data is read from standard input (keyboard by default)
  - Similarly for output files and standard output (screen)
- The pipe symbol ‘|’ attaches the standard output of one program to the standard input of another, e.g. who | wc -l

Common wc flags

-l number of lines
-w number of words
-c number of characters

By default, all three stats are displayed. Flags state what stats appear…

Question

- If the UNIX command ‘head file’ outputs the first 10 lines of file, the command ‘tail –n file’ outputs the last n lines of file, and the command ‘wc –w file’ counts the number of words in file, what will the following output?

  head file | tail -1 | wc –w

  a) The number of words in the tenth line from the end of file
  b) The first 10 lines of file, then the last line of file, then the number of words in file
  c) The number of words in line 10 of file only
  d) The number of words in line 10, then line 9, then line 8, etc.
  e) The number of words in the first ten lines plus the last line of file

Answer: c
Only line 10 will be passed to wc -w

Sockets

- A socket is a communication endpoint of a channel between two processes
- Unlike pipes, the processes
  - do not need to have the same lineage
  - do not need to be on same machine
  - do not even need to be on same local-area network
- When two processes communicate using sockets, one is designated the server and the other the client
  - in some cases, doesn't matter which is which
- More usually, a daemon server process offers a service to many clients

Client-Server Examples

- The following are examples of common servers:
  - Web server: accessed by client's web browser
  - Mail server: retrieving and sending emails to clients
  - File server: holding documents to be accessed by clients
  - Database server: providing database services to clients, e.g. customer database, stock database…
  - etc