“OS Structures” lecture review

- There are three main categories of tasks a modern OS has to accomplish
  - process management
  - memory management
  - disk/file management
- OS is a big and complex program; traditional monolithic kernel design approach yields OSes that are fast but hard to develop, modify and debug. Other approaches have been suggested:
  - layering
  - microkernel
  - network computers

Lecture 4: Process Management

- looking up processes in Unix
- process creation and termination
- process state
- process control block
- process scheduling
  - 2 - state model
  - 5 - state model

What is a process?

- A *process* (sometimes called a *task,* or a *job*) is, informally, a program in execution
- “Process” is not the same as “program”
  - We distinguish between a passive program stored on disk, and an actively executing process
    - Multiple people can run the same program; each running copy corresponds to a distinct process
  - The program is only part of a process; the process also contains the execution state

Listing Unix Processes

```
prompt%  ps -efl
UID     PID  PPID   C   STIME    TTY       TIME  COMMAND
root      0     0   0   Jan 19    ?       10:53   swapper
root      1     0   0   Jan 19    ?        0:17   init
root      2     0   0   Jan 19    ?        7:39   telnetd
…
mikhail 1546 1321   0  15:38:45 pts/17     3:31   netscape
```

- list processes (HP UNIX):
  - ps — my processes, little detail
  - ps -fl — my processes, more detail
  - ps -efl — all processes, more detail
- note user processes and OS processes

Process creation and termination

- Reasons for process creation
  - User logs on
  - User starts a program
  - OS creates process to provide a service (e.g., printer daemon to manage printer)
  - Program starts another process (e.g., netscape calls xv to display a picture)
- Reasons for process termination
  - Normal completion
  - Arithmetic error, or data misuse (e.g., wrong type)
  - Invalid instruction execution
  - Insufficient memory available, or memory bounds violation
  - Resource protection error
  - I/O failure

Process state

- The *process state* consists of (at least):
  - Code for the program
  - Program’s static and dynamic data
  - Program’s procedure call stack - contains temporary data - subroutine parameters, return addresses, temporary variables
  - Contents of general purpose registers
  - Contents of Program Counter (PC) —address of next instruction to be executed
  - Contents of Stack Pointer (SP)
  - Contents of Program Status Word (PSW) — interrupt status, condition codes, etc.
  - OS resources in use (e.g., memory, open files, connections to other programs)
  - Accounting information

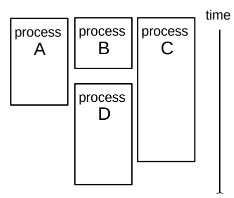
↙ Everything necessary to resume the process’ execution if it is somehow put aside temporarily

Process control block (PCB)

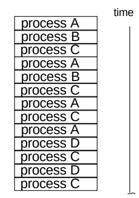
- For every process, the OS maintains a *Process Control Block* (*PCB*), a data structure that represents the process and its state:
  - Process id number
  - Userid of owner
  - Memory space (static, dynamic) - base/limit register contents
  - Program Counter, Stack Pointer, general purpose registers
  - Process state (running, not-running, etc.)
  - CPU scheduling information (e.g., priority)
  - List of open files
  - I/O states, I/O in progress - list of I/O devices allocated to process, list of open files, etc. status of I/O requests
  - Pointers into CPU scheduler's state queues (e.g., the waiting queue)
  - …

Interleaving execution of processes

From the user's standpoint processes multiple processes are executed concurrently

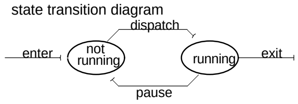


In reality OS interleaves the process execution



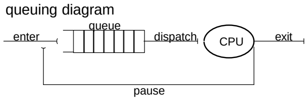
Two state process model

state transition diagram



a process is either *running* (executing CPU instructions or *not running*

queuing diagram



OS takes a ready to run process and puts it in a queue each process is allocated a quantum of CPU time called *time slice*

- CPU scheduling (round-robin)
  - Queue is first-in, first-out (FIFO) list
  - CPU scheduler takes process at head of queue, runs it on CPU for one time slice, then puts it back at the tail of the queue

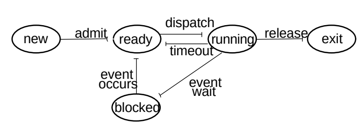
Process Transitions in the Two-State Process Model

- When the OS creates a new process, it is initially placed in the not-running state
  - It's waiting for an opportunity to execute
- At the end of each time slice, the *CPU scheduler* selects a new process to run
  - The previously running process is *paused* — moved from the running state into the not-running state (at tail of queue)
  - The new process (at head of queue) is *dispatched* — moved from the not-running state into the running state
    - If the running process completes its execution, it exits, and the CPU scheduler is invoked again
    - If it doesn't complete, but its time is up, it gets moved into the not-running state anyway, and the CPU scheduler chooses a new process to execute

Blocked processes

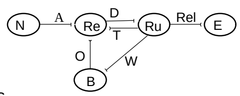
- problem with two state model: process may waste valuable CPU time waiting for something to happen; for example:
  - Wait for user to type the next key
  - Wait for output to appear on the screen
  - Program tried to read a file — wait while OS decides which disk blocks to read, and then actually reads the requested information into memory
  - Netscape tries to follow a link (URL) — wait while OS determines address, requests data, reads packets, displays requested web page
- solution: OS should not try to run blocked processes at all; therefore the OS should differentiate between:
  - Processes that are ready to run and are waiting their turn for another time slice
  - Processes that are waiting for something to happen (OS operation, hardware event, etc.)

Five state process model



- the *not-running* state in the two-state model has now been split into a *ready* state and a *blocked* state
  - *running* — currently being executed
  - *ready* — prepared to execute
  - *blocked* — waiting for some event to occur (for an I/O operation to complete, or a resource to become available, etc.)
  - *new* — just been created
  - *exit* — just been terminated

State transitions in five-state process model



- new → ready
  - Admitted to ready queue; can now be considered by CPU scheduler
- ready → running
  - CPU scheduler chooses that process to execute next, according to some scheduling algorithm
- running → ready
  - Process has used up its current time slice
- running → blocked
  - Process is waiting for some event to occur (for I/O operation to complete, etc.)
- blocked → ready
  - Whatever event the process was waiting on has occurred