CS 3214: Computer Systems
Lecture 3: Processes & Unicode

Instructor: Huaicheng Li

Aug 30 2022

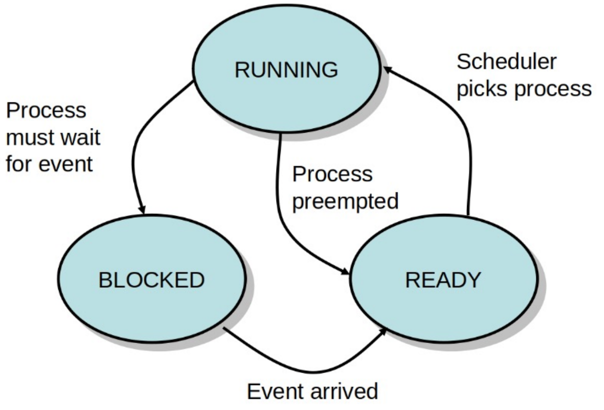
VIRGINIA TECH™

Announcements

- Ex0 released, deadline: 9/2 11:59pm

Process States

- **Running:** executing its instructions on CPUs
- **Ready:** ready to execute but waiting for its turn
- **Blocked:** stopped due to external events, cannot make use of CPUs even if some are available
- Running → Blocked
  - Input, exclusion access to a lock, signal, sleep(), waiting for child process
- Blocked → Ready
  - OS adds the process to a ready queue
- Ready → Running
  - 1 process per CPU, scheduling policy
- Running → Ready
  - De-scheduled (yield or preempted)



Discussion Questions

1. What happens if an n CPU system has exactly n READY processes?
2. What happens if an n CPU system has 0 READY processes?
3. What happens if an n CPU system has $k < n$ READY processes?
4. What happens if an n CPU system has 2n READY processes?
5. What happens if an n CPU system has $m >> n$ READY processes?
6. What is a typical number of BLOCKED/READY/RUNNING processes in a system (e.g., your phone or laptop?)
7. How does the code you write influence the proportion of time your program spends in the READY/RUNNING state?
8. How can the number of processes in the READY/RUNNING state be used to measure CPU demand?
9. Assuming the same functionality is achieved, is it better to write code that causes a process to spend most of its time BLOCKED, or READY?

Answers (permuted order)

- Prefer BLOCKED to READY because it does not consume CPU; use OS facilities to wait for events rather than poll in a loop
- 150 − 500 BLOCKED, and 0 − 2 RUNNING
- Every process takes about twice as long as it normally would
- The load average is a weighted moving average of the size of the ready queue (including RUNNING processes); it says how many CPUs could be kept busy
- System becomes very laggy, processes take much longer than normal
- n − k CPUs are idle, k CPUs run exactly 1 process
- Each CPU runs exactly 1 process
- Performing computation without performing I/O means the process is READY at all times and will be RUNNING if scheduled.
- The system is idle and goes into a low-power mode

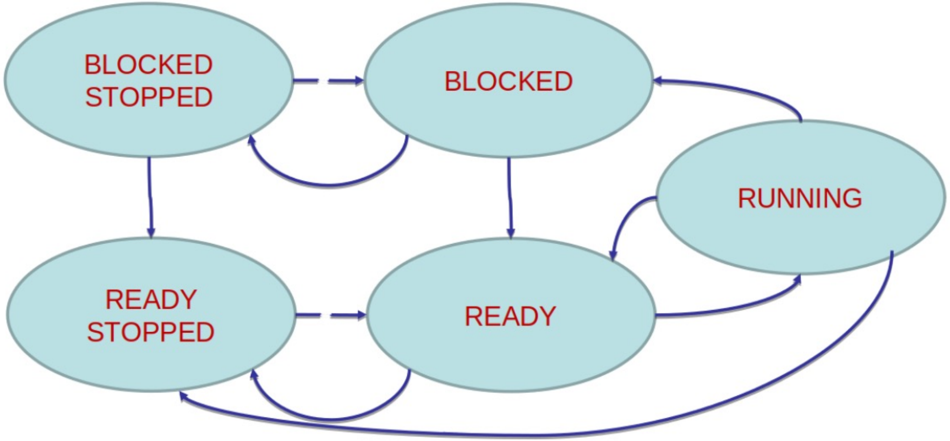
Process States

- Our model is simplified, real OS often maintain state diagrams with 5-15 states for their threads/tasks
  - Linux uses the following states

D uninterruptible sleep (usually IO)
I Idle kernel thread
R running or runnable (on run queue)
S interruptible sleep (waiting for an event to complete)
T stopped by job control signal
t stopped by debugger during the tracing
X dead (should never be seen)
Z defunct ("zombie") process, terminated but not reaped by its parent

Process States and Job Control

- Job control: stop/suspend, and resume a process
  - Linux commands: *jobs, bg, fg, Ctrl-Z*
- Job control and process states



Programmer's View

- Process state transitions are guided by decisions or events outside the programmer's control (user actions, user input, I/O events, inter-process communication, synchronization) and/or decisions made by the OS (scheduling decisions)
- They may occur frequently, and over small time scales
  - e.g., on Linux preemption may occur every 4ms for RUNNING processes
  - when processes interact on shared resources (locks, pipes) they may frequently block/unblock)
- For all practical purposes, these transitions, and the resulting execution order, are unpredictable
- The resulting concurrency requires that programmers not make any assumptions about the order in which processes execute; rather, they must use signaling and synchronization facilities to coordinate any process interactions