Concurrency Control

Peter J. Denning

Society of Cooperating Processes

- Processes run autonomously at unknown speeds
- Processes outnumber the CPUs that execute them
- Processes exchange messages containing requests and responses
- Processes synchronize, forcing an order on some actions
- Processes wait for resources to be assigned

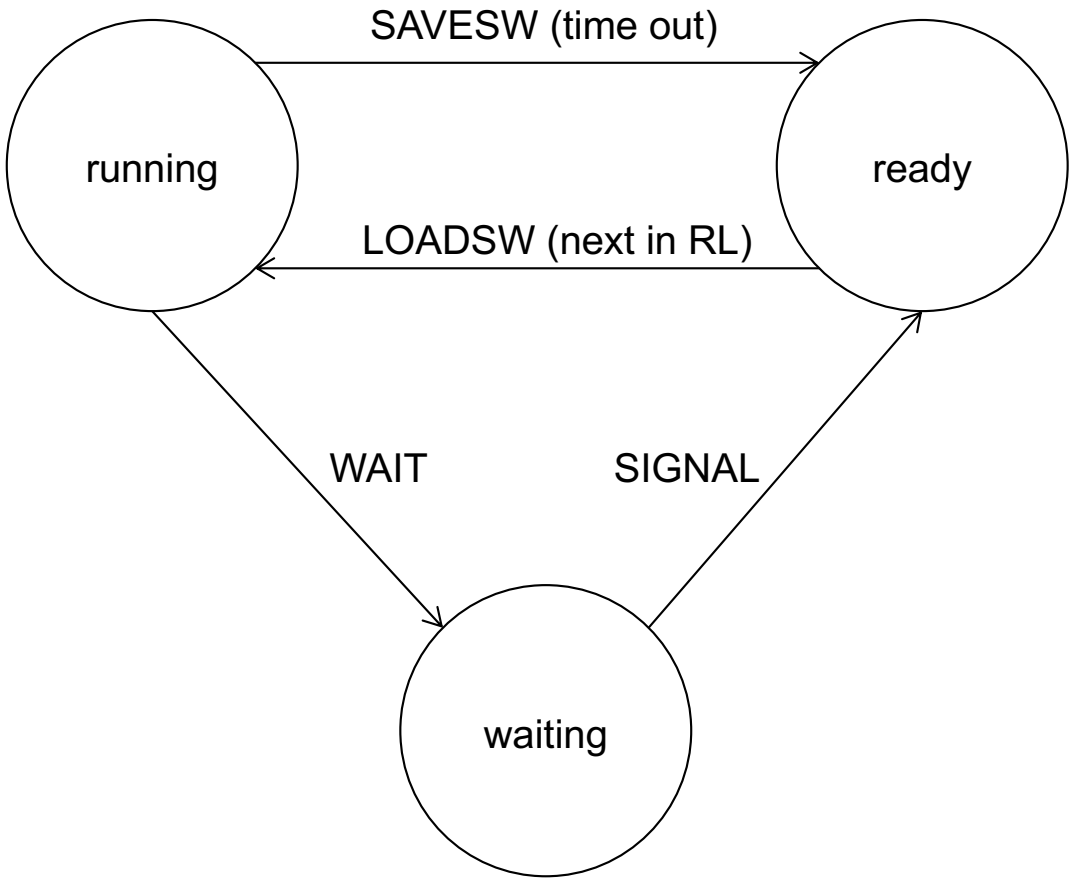
A Note on Terminology

- Thread = trace of instructions executing in a program
- Process = program in execution on a virtual machine
- Process includes one or more threads, working memory, and other elements
- Terms process and thread are often used interchangeably, including here
- Listen carefully to the conversation to see which meaning "process" has

The Job of Concurrency Control

- Implement thread states running, ready, and waiting; and manage the transitions among them.
- With this framework, the OS from kernel level 3 upwards looks like a society of cooperating processes.
- Use only tools available at Level 1: instructions, stacks, interrupts, kernel mode, privileged instructions, PSWs, statewords, memory mapping.

Process States



CPU multiplexing manages the transitions of threads between the running and ready states. Discussed in Module 1.

Semaphores implement the waiting state, WAIT and SIGNAL manage the transitions into and out of the waiting state. Discussed in this Module 3.

Basic Functions

- Representing processes and their states
  - Thread control blocks, queues
- Multiplexing
  - Of CPUs among available (ready) threads
- Basic synchronization
  - Semaphores
- Anticipation of common synchronization patterns
  - Serialization, mutual exclusion, producer-consumer, determinacy (removal of lurking bugs), deadlocks

Concurrency Control as Abstract Machine

- Internal hidden data structures
  - TCBs, SCBs, RL, semaphore queues
- Interface
  - `p=CREATE_THREAD(init IP), s=CREATE_SEM(init count)`
  - `DELETE_THREAD(p), DELETE_SEM(s)`
  - `SUSPEND(p), RESUME(p)`
  - `WAIT(s), SIGNAL(s)`
  - `SLEEP, WAKEUP(p)`

Performance!

- Operations must be extremely fast
- Small, fixed overhead independent of queue lengths
- TSL locks used only for very short lock times
- TSL locks not used above kernel level 2
- Procedure calls whenever possible instead of context switches
- Interrupt dispatch extremely fast