Scheduler implements method

1. Multiplexing

1. Use context switch to switch from one process to another.

2. Use timer interrupt handler to drive context switches.

3. Use locking plan to avoid races and switch among processes concurrently.

4. Free some resources at appropriate time, but can’t free its own kernel stack.

1. Context switching

Two kinds of CS at a low level:

process’s kernel thread → current CPU’s scheduler thread

scheduler thread → process’s kernel thread

NEVER directly switches from one user-space process to another

1. Scheduling
2. A process that gives up CPU must acquire the process table lock ptable.lock, release any other locks it is holding, update it’s own state (proc->state), and then call sched.
3. Sched double-checks those conditions, and then an implication of those conditions: since a lock is held, the CPU should be running with interrupts disabled.
4. Sched calls swtch to save the current context in proc->context and switch to the scheduler context in cpu->scheduler. Swtch returns on the scheduler stack as though scheduler’s swtch had returned.
5. The scheduler continues the for loop, finds a process to run, switches to it, and the cycle repeats.
6. Sleep and wakeup mechanism
7. Pipes
8. Wait, exit, and kill