The process state machine

CS 447
Monday 3:30-5:00
Tuesday 2:00-3:30
Process States

- Process undergoes state changes
- Responds to requests based on its current state
- What states need to be considered?
Inputs to decide the state space

- Is the process in run queue?
- Queue no.?
- Wait queue?
- On which device? For how much time?
- Is it actually ‘running’? – current time slice belongs to the process?
- Is the process exited?
3-state machine

- running
  - scheduled
  - Time sliced
    - Blocking call
      - exit
    - Device ready/wait ends
      - create

- ready

- blocked
Trace: p1 kills p2 (bash kills 1234)

- P1: running
- P2: ready
- P1 executes kill p2 – signal delivered in the mailbox of p2
- P1 continues till its time slice
- ....
- Eventually p2 is scheduled – it handles the signal or gets terminated
A Unix-like state machine: worked out in class
Kernel Functionality

System call API

Exceptions generated by processes

Hardware interrupts by devices

System Processes such as page daemon, swapper
Bootstrapping

- Initialize memory
- Set up environment for processes
- Create few initial processes which further create other processes
System Call

- A wrapper routine
  - Push syscall number on user stack
  - Invoke a trap instruction
  - Syscall() – trap handler in kernel mode
## Context and Mode

<table>
<thead>
<tr>
<th>Process context</th>
<th>Kernel context</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User mode</strong></td>
<td><strong>Kernel mode</strong></td>
</tr>
<tr>
<td><strong>i=i+1</strong></td>
<td><strong>Return syscall value</strong></td>
</tr>
<tr>
<td>Syscall wrapper – i.e. invoke a system call</td>
<td>Pick syscall args</td>
</tr>
<tr>
<td>Not possible</td>
<td><strong>v() on a semaphore – increment s</strong></td>
</tr>
</tbody>
</table>

- `v()` on a semaphore – increment `s`
Process Address space components

- User Address Space
  - Text – executable code
  - Initialized data – objects initialized in program
  - non-initialized data (OS generates 0 filled pages)
  - Shared memory
  - Shared libraries
  - Heap – dynamically allocated memory
  - User stack – kernel allocates a stack for all processes
Process Address space components

- Control information – data structures of interest to kernel (proc structure)
- Credentials – uid, gid, ..
- File descriptor table for open files
- Environment variables
- Hardware context – registers, memory management registers
Process Table

- Process control block for each process
- A limited number of processes
- Index in PT is pid
- Various queues are superimposed on process table