CSCE 451/851
Operating Systems Principles

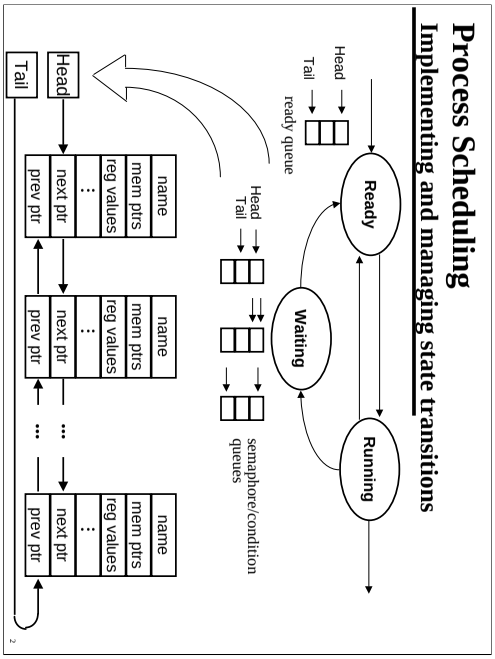
Processor Scheduling

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

1

Process Scheduling
Implementing and managing state transitions



2

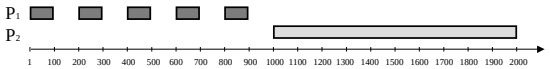
Why Schedule?

Scheduling goals

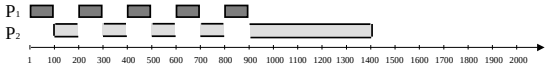
- Example: two processes execute in parallel

```
process P1                    process P2
begin                         begin
  for i := 1 to 5 do            <execute for 1 sec >
    <read a char>             end P2
    <process a char>
  end for
end P1
```

- Performance without scheduling



- Performance with scheduling



3

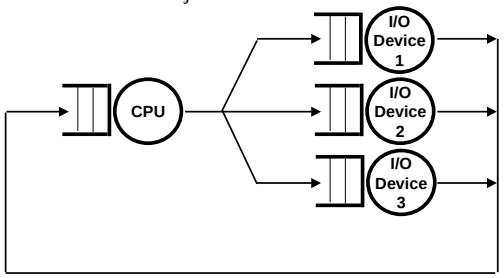
Types of Schedulers

- Long term schedulers
  - » adjust the level of multiprogramming through admission control
- Medium term schedulers
  - » adjust the level of multiprogramming by suspending processes
- Short term schedulers
  - » determine which process should execute next

4

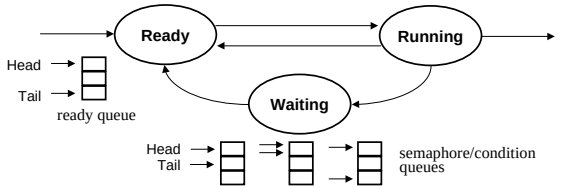
Long Term Scheduling
Balancing CPU & I/O demand

- Example — The *Convoy Effect*
  - » 1 CPU bound job
  - » *n* >> 1 I/O bound jobs



5

Short Term Scheduling
When to schedule



When a process makes a transition...

1. from *running* to *waiting*
2. from *running* to *ready*
3. from *waiting* to *ready*

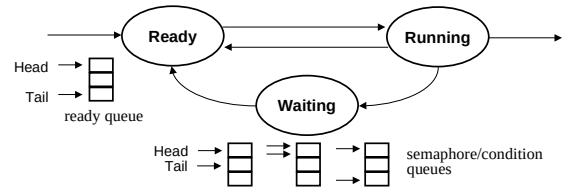
(3a. a process is *created*)

4. from *running* to *terminated*

6

Short Term Scheduling

How to schedule — Implementing a context switch




```
context_switch(queue : system_queue)
var next : process_id
begin
  DISABLE_INTS
  insert_queue(queue, runningProcess)
  next := remove_queue(readyQueue)
  dispatch(next)
  ENABLE_INTS
end context_switch
```

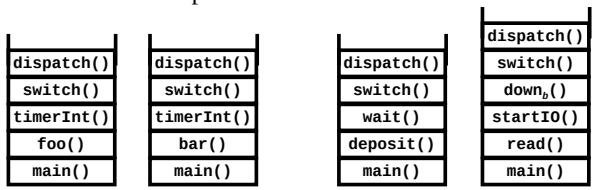
```
dispatch(proc : process_id)
begin
  <save memory image of runningProcess>
  <save processor state of runningProcess>
  <load memory image of proc>
  <load processor state of proc>
  runningProcess := proc
end dispatch
```

7

Implementing a Context Switch

Dispatching

- Case 1: Preemption
- Case 2: Yield



"running" "next" "running" "next"

"running's" dispatch:

```
dispatch()
begin
   <save state of running>
      :
end dispatch
```

"next's" dispatch:

```
dispatch()
begin
   <save state of running>
   <load state of next>
end dispatch
```

8

Scheduling Policies
Evaluation criteria

- CPU/device utilization
- System throughput
- Waiting time
- Turnaround time
- Response time

9

Scheduling Policies
First-Come-First-Served (FCFS)

- The discipline corresponding to FIFO queueing
- Example — 3 processes w/ compute times 12, 3, and 3
  - » Job arrival order P_1, P_2, P_3

  - » Job arrival order P_2, P_3, P_1

10

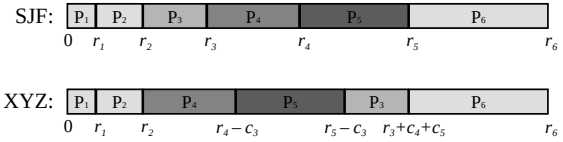
Scheduling Policies
Shortest-Job-First (SJF)

- Select the job that is closest to finishing
  - » enqueue jobs in order of estimated completion time

  t_n — duration of the n^{th} CPU burst
  τ_{n+1} — predicted duration of the n+1st CPU burst

  $$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, \text{ for } 0 \le \alpha \le 1$$

- An optimal policy for minimizing response time

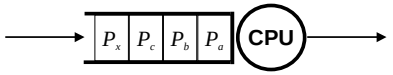


11

Scheduling Policies
Priority Scheduling (PS)

- Assign a priority (a number) to each job and schedule jobs in order of priority
  - » typically low priority values = "high priority"

    E.g., if priority = τ_n, then a priority scheduler becomes a SJF scheduler.

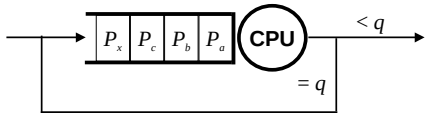


- Aging — Avoiding starvation
  - » gradually increase a process's priority (decrease it's priority value) over time

12

Scheduling Policies
Round-Robin Scheduling (RR)

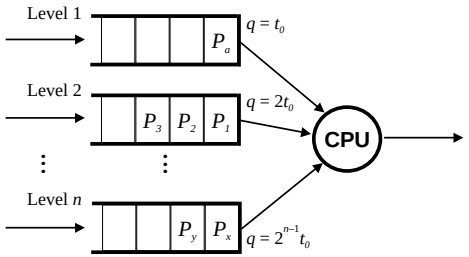
- Allocate the processor in discrete units called *quantums* (or *time-slices*)
- Switch to the next ready process at the end of each quantum
  - » Processes execute every $(n - 1)q$ time units



13

Scheduling Policies
Multi-level feedback queues (MLF)

- *n* priority levels — priority scheduling between levels, round-robin within a level
- Quantas decrease with priority level
- Jobs are demoted to lower priority levels if they don't complete within the current quantum



14