Operating Systems Concepts

xv6 scheduling

CS 4375, Fall 2025

Instructor: MD Armanuzzaman (***Arman***)

marmanuzzaman@utep.edu

October 8, 2025

Summary

- Main Memory:
  - Fixed and Dynamic Memory Allocation
  - External and Internal Fragmentation
  - Address Binding
  - Hardware Address Protection
  - Paging
  - Segmentation

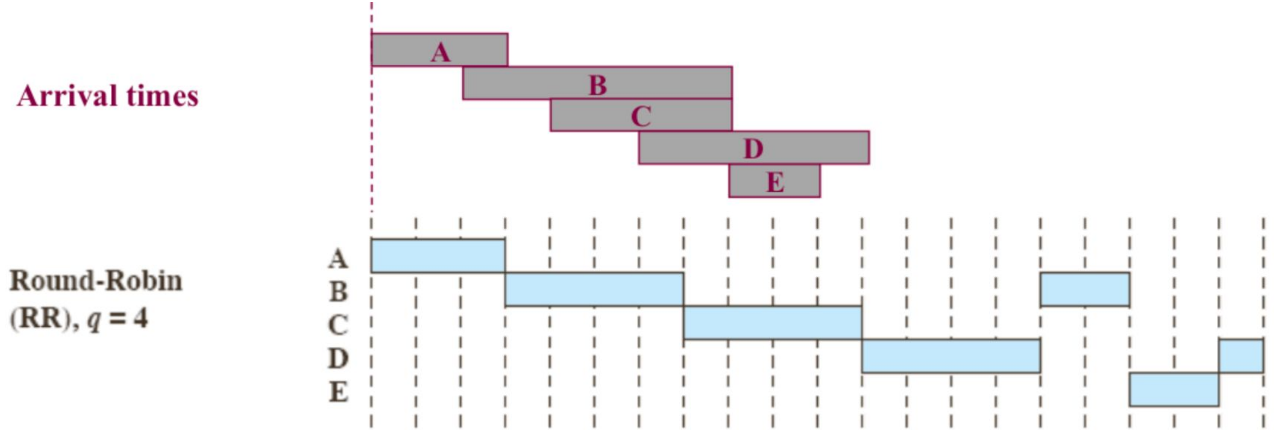
Agenda

- xv6 scheduling
  - Default round robin scheduling of xv6
  - Understand the code base
- Implement priority scheduling in xv6
  - Tasks in homework 3
- Implement aging policy

Scheduling: Round-Robin (RR)

- A crucial parameter is the quantum **q** (~10-100ms)
  - **q** should be large compared to context switch latency (~10μs)
  - **q** should be less than the longest CPU burst, or RR degenerates to FCFS

RR (q = 4) Scheduling Policy



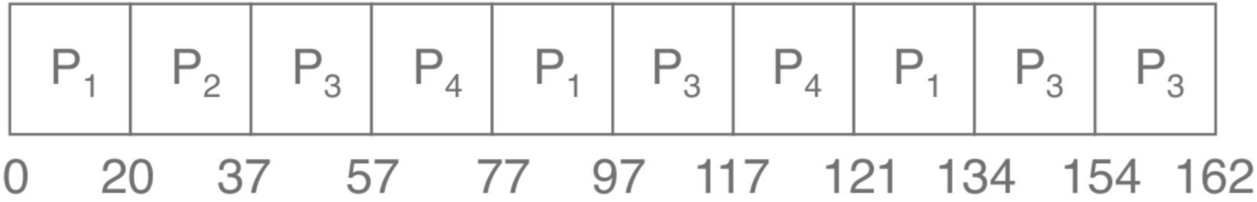
RR $q = 4$		A	B	C	D	E	Mean
	Finish Time	3	17	11	20	19	
	Turnaround Time (T_r)	3	15	7	14	11	10.00
	T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71

Scheduling: RR Example

- Consider **q** = 20

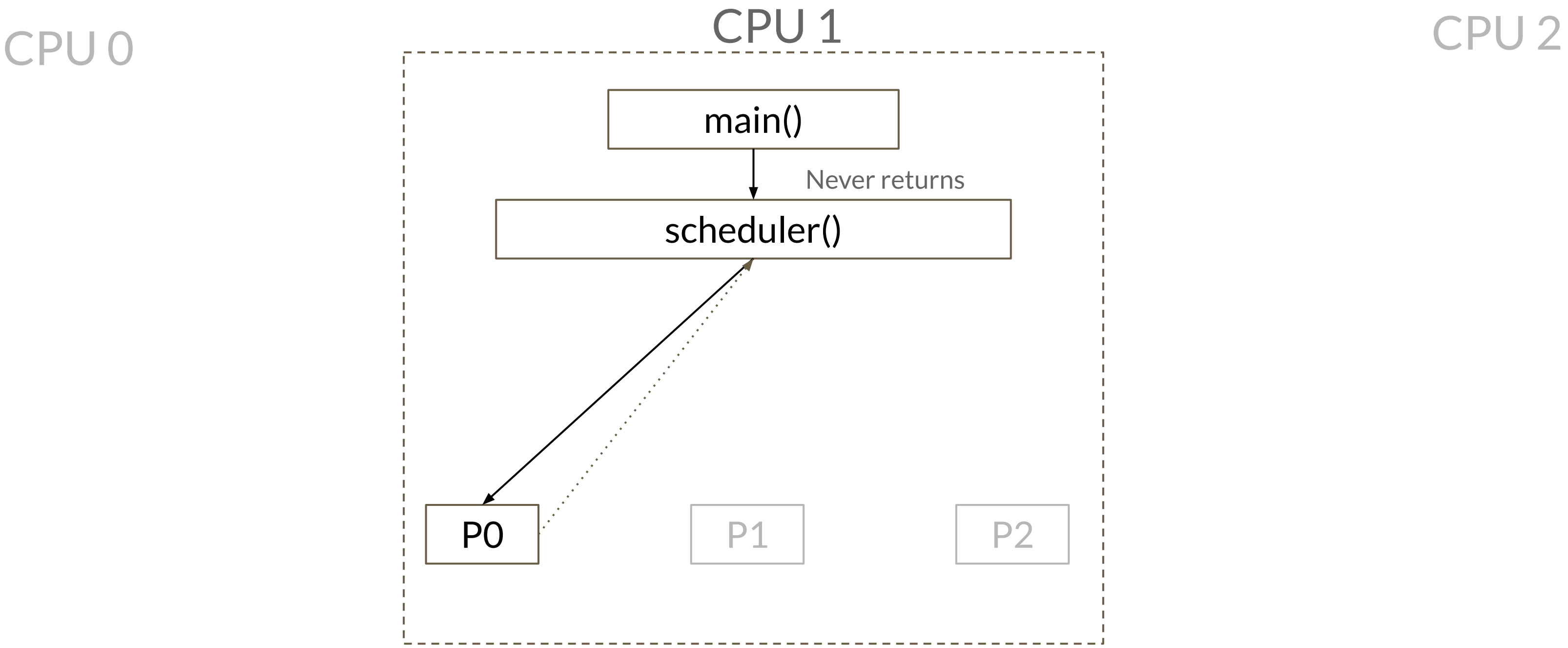
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart for the schedule is:



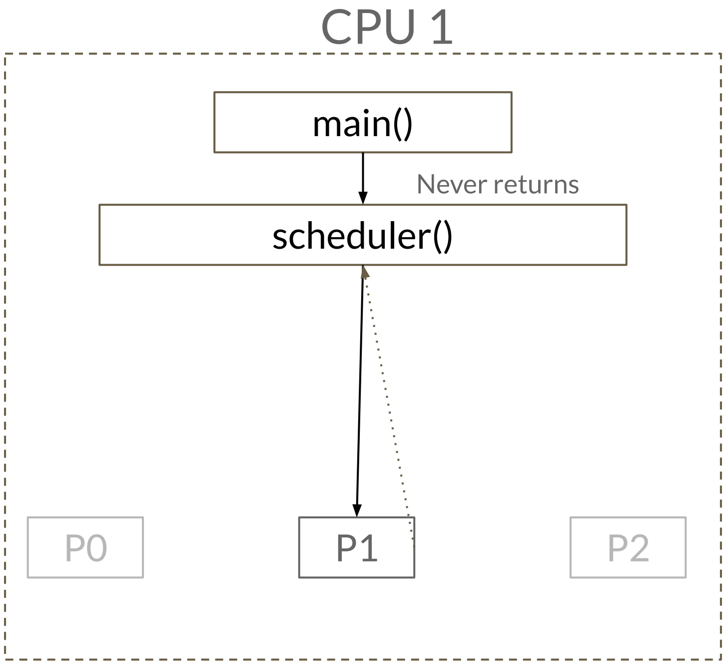
- Typically, higher average turnaround than SJF, but better ***response***

xv6 scheduling



xv6 scheduling

CPU 0



CPU 2

Round Robin Scheduling of xv6

- `scheduler()` in `proc.c`
  - Invoked from `kernel/main.c`
  - INFINITE LOOP -> it never returns
    - Control loop that executes all user applications
  - Selects the next `RUNNABLE` state
  - Switches to that process (`swtch.S`)
    - Where it last yielded/slept, or its start
    - NEED TO SAVE THE CONTEXT OF SCHEDULER ITSELF!!!
  - Releases the lock at the end (pay attention to lock acquire releases)

Limited Direct Execution Protocol

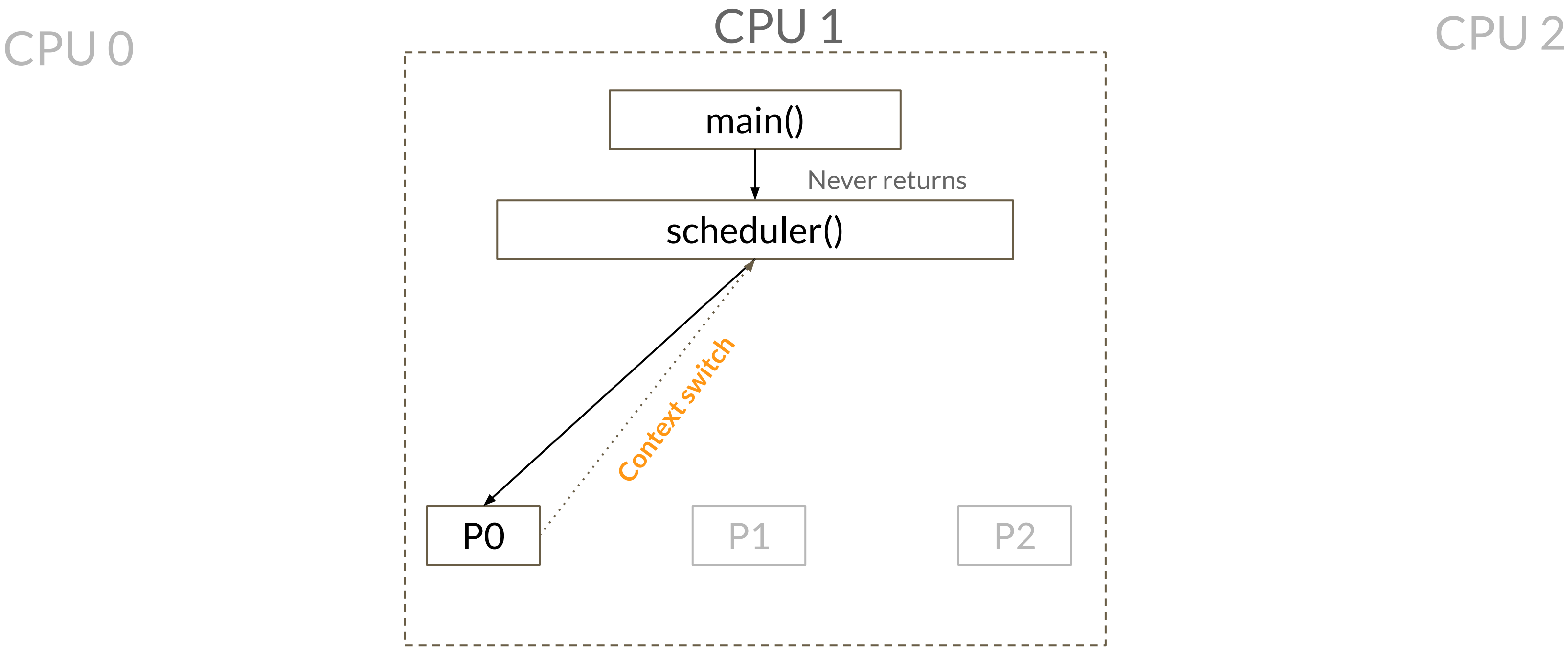
OS @ boot (kernel mode)	**Hardware**
initialize trap table	
	remember address of... syscall handler

Phase 1

OS @ run (kernel mode)	**Hardware**	**Program (user mode)**
Create entry for process list Allocate memory for program Load program into memory Setup user stack with argv Fill kernel stack with reg/PC **return-from-trap**		
	restore regs from kernel stack move to user mode jump to main	
		Run main() ... Call system call **trap** into OS
	save regs to kernel stack move to kernel mode jump to trap handler	
Handle trap Do work of syscall **return-from-trap**		
	restore regs from kernel stack move to user mode jump to PC after trap	
		... return from main **trap** (via `exit()`)
Free memory of process Remove from process list		

Phase 2

xv6 scheduling



Round Robin Scheduling of xv6

- A user process to give up CPU
  - exit()
  - yield()
  - sleep()
  - Preempted by the timer interrupt
    - 100ms

`sched() in proc.c`

Round Robin Scheduling of xv6

- `sched()` in proc.c
  - myproc()
    - Currency process
  - Hold only p->lock
  - Process must not be RUNNING
    - RUNNABLE, SLEEPING, or ZOMBIE
  - Interrupt bookkeeping
  - mycpu()->context
    - **Holds the context of the `scheduler` process**

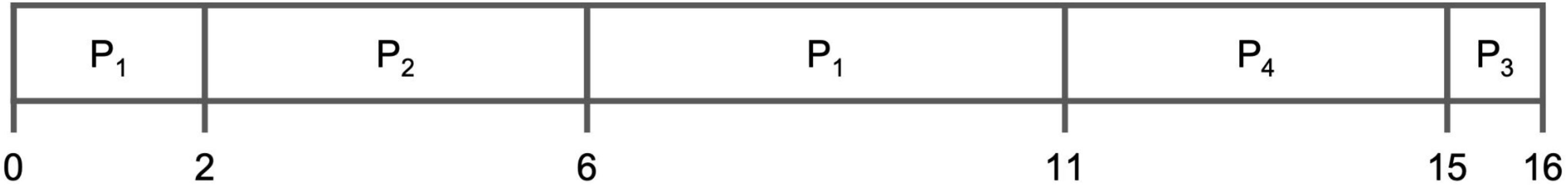
Implementing Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (largest)
- Xv6 Schemes:
  - Preemptive – The timer interrupt yields() the running process and should provide a chance to schedule a new process with higher priority
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem: Starvation– Low priority processes may never execute
- Solution: **Aging** – As time progresses, increase the priority of the process

Scheduling: Priority Example

Process	Arrival Time	Burst Time	Priority
P_1	0	7	2
P_2	2	4	1
P_3	4	1	4
P_4	5	4	3

- Non-preemptive:
  - $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_3$
- Preemptive:



Task 1

- Implement `getpriority()` and `setpriority()` system calls
    - Where do you add the priority field?
    - When do you initialize?
- Forked child should inherit the parent priority
    - `fork()`
    - Where do you set the priority when new child is created with fork?
- Modify `ps` program to show priority of each process
- Test program:
    - `task1.c`

Task 2

- Implement priority scheduling
  - Compile time option in `param.h` to select between round robin and priority scheduling
  - `#define Scheduler_ALGO` **0 or 1**
  - Which function should host the implementation of the priority scheduling?
- Test program
  - task2.c

Task 3

- Keep track of process aging
  - Add a field in proc to store `readytime`
  - When a process is changed for other state to `RUNNABLE`
- Modify ps program to print age of a process
  - Update given pstat.h
- age
  - current time - ready time
- Test your implementation

Task 4

- Implement and merge process aging with priority scheduling
  - Add a field in proc to store `readytime`
  - When a process is changed for other state to `RUNNABLE`
- Modify ps program to print age of a process
  - Update given pstat.h
- **age**
  - current time - ready time
- Test your implementation

Task 4

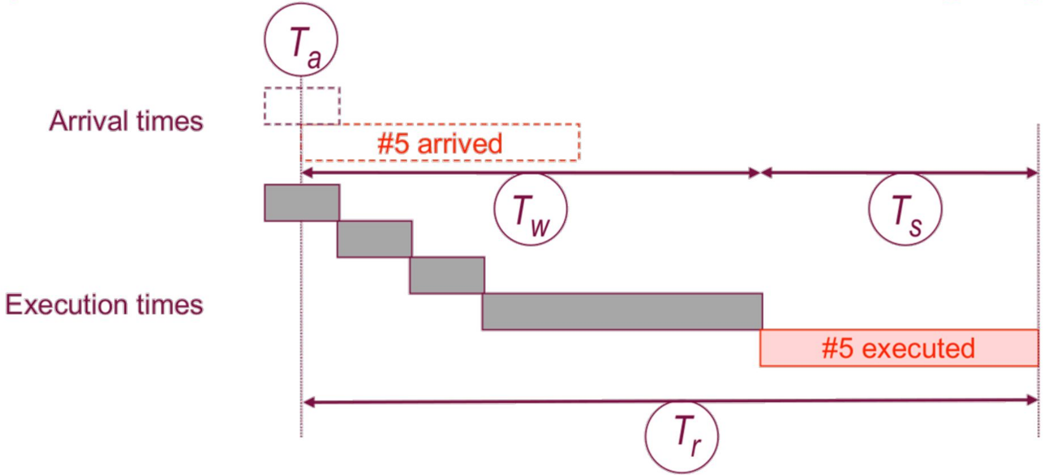
- **age**
  - current time - ready time

```
#define MAXEFFPRIORITY 99
#define AGING_DIV 25

effective_priority = min(MAXEFFPRIORITY, priority + (currtime -
    readytime)/AGING_DIV)
```

Scheduling Metrics (Extra credit)

- T_a - **Arrival time:** Time the process became "READY" [again]
- T_w - **Waiting time:** Time spent waiting for the CPU
- T_s - **Service time:** Time spent executing in the CPU
- T_r - **Turnaround time:** Time spent waiting and executing = $T_w + T_s$



$T_r / T_s = 2.5$

Announcement

- Quiz 3
  - On blackboard
- Homework 3
  - Due on Monday October 27th, 11.59 PM