Chapter 6: Processor Scheduling

Introduction Analogy: You have laundry to do. You have tests to prepare for. You have to get some food ready for dinner. You have to call mom and wish her happy birthday. There is only one of you and you have to get all of this done in a timely manner. You are going to prioritize these activities but you also know that not all of them need your constant attention. For example, once you get the washing machine for the laundry started until it beeps at you at the end of the cycle, you don't have to pay attention to it.

So, you create a plausible schedule

1. Start the wash cycle
2. Stick the food in the microwave and zap it
3. Call Mom
4. Prepare for tests

The idea is that you have this notion of **priority** and you are trying to find a way to divide up scarce resources

Programs and Processes

A **process** is a program in execution

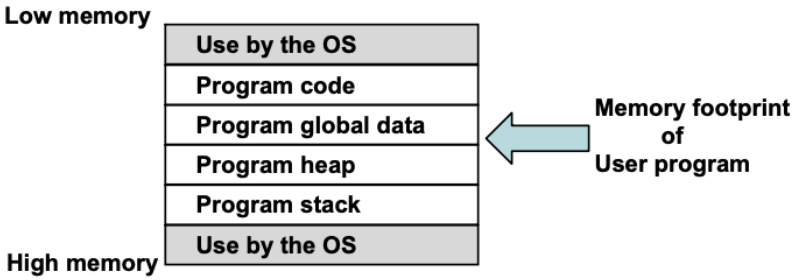


Figure 6.1: Memory Footprint

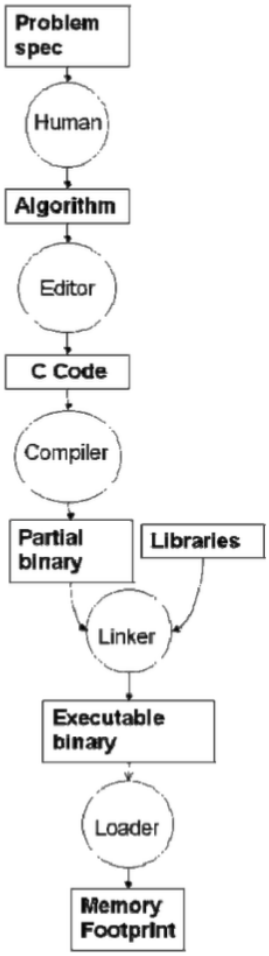


Figure 6.2: Life cycle of program creation

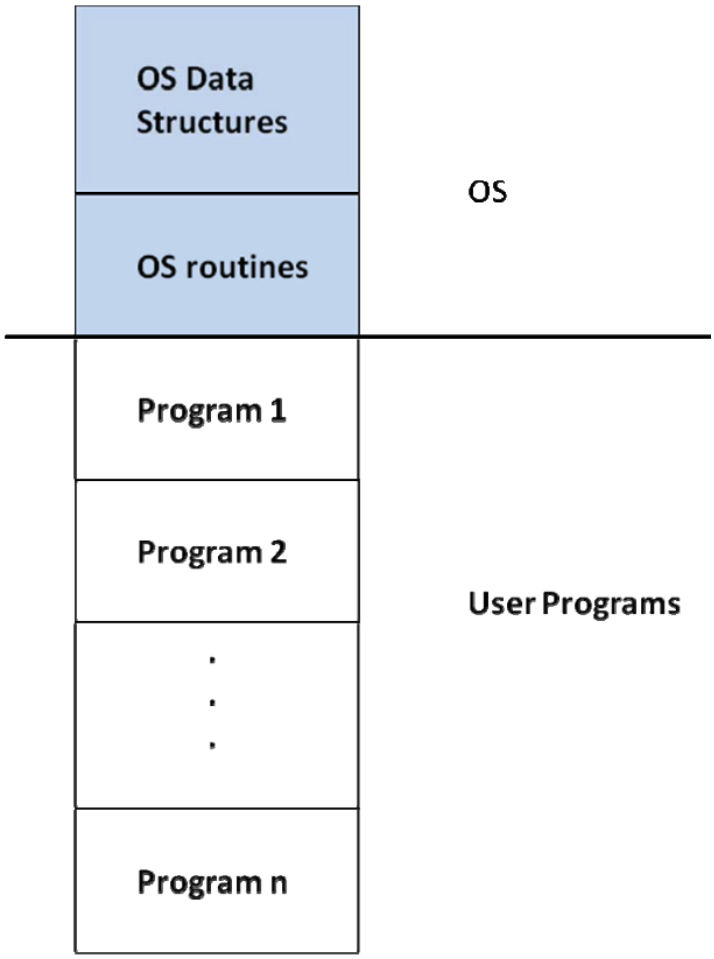


Figure 6.3: OS and user program in memory

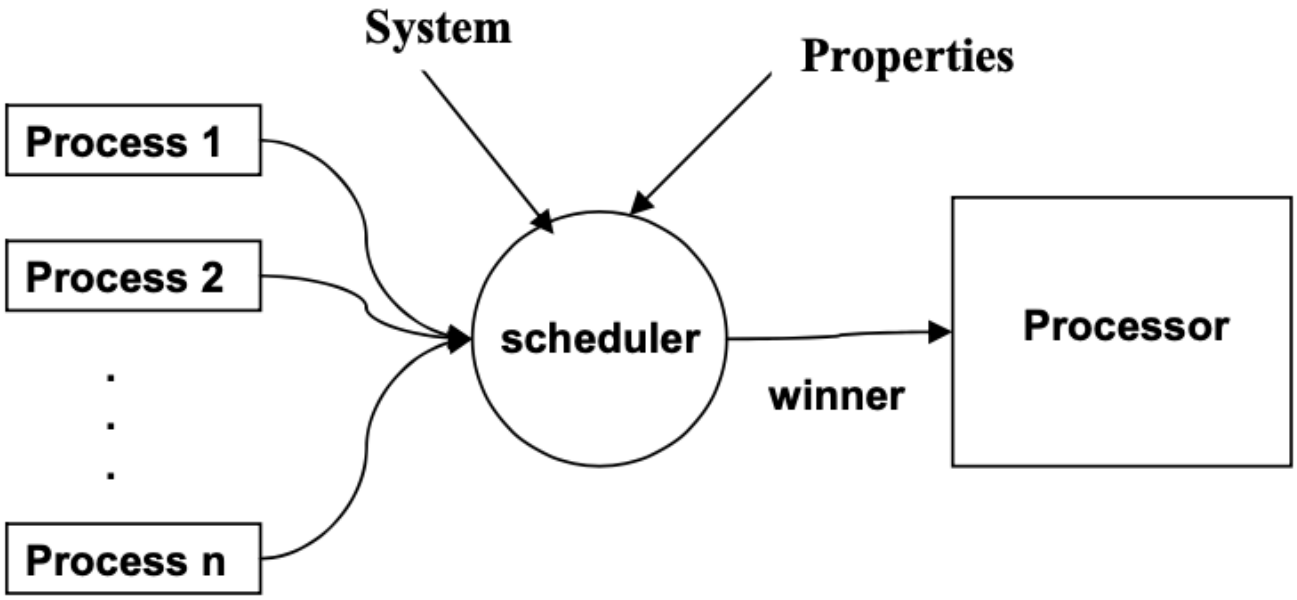


Figure 6.4: Scheduler

Terminology

- Tasks = unit of work
- Threads = unit of scheduling

More concretely, a process is a program + all the threads that are executing in that program

Name	Usual Connotation	Use in this chapter
Job	Unit of scheduling	Synonymous with process
Process	Program in execution; unit of scheduling	Synonymous with job
Thread	Unit of scheduling and/or execution; contained within a process	Not used in the scheduling algorithms described in this chapter
Task	Unit of work; unit of scheduling	Not used in the scheduling algorithms described in this chapter, except in describing the scheduling algorithm of Linux

Scheduling Environment

Name	Environment	Role

Name	Environment	Role
Long Term Scheduler	Batch Oriented OS	Control the job mix in memory to balance use of system resources (CPU, memory, IO)
Loader	Every OS	Load user program from disk into memory
Medium Term Scheduler	Every modern OS (time-shared, interactive)	Balance the mix of processes in the memory to avoid thrashing
Short Term Scheduler	Every modern OS	Scheudle the memory resident processes onto the CPU
Dispatcher	Every OS	Populate the CPU registers with the state of the process selected for running by the short-term scheduler

In short, the steps involved in scheduling are

1. Grab attention of processor
2. Save state of currently running process
3. Select new process to run
4. Dispatch newly selected process to run on the processor

Thrashing: A phenomenon wherein the dynamic memory usage of the processes currently in the ready queue exceeds the total memory capacity of the system

- Generally too much context switching

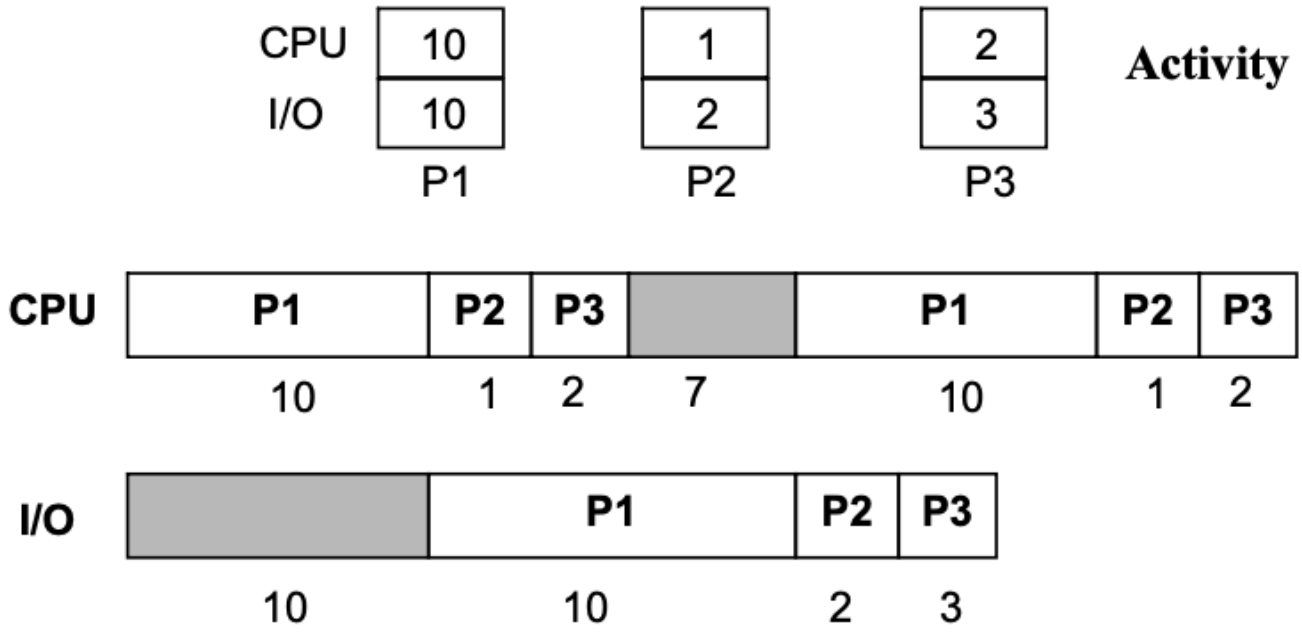
Performance Metrics

Metric	Explanation	How to find	Centric
Throughput	Number of jobs executed in a period of time	numJobs/time	System
Average Turnaround time	Average elapsed time between when jobs enter the system and when they leave	sum(leave - enter)/numJobs	System
Average Waiting Time	Average time a job is in the ready queue and not execution	sum(waitTime)/numJobs	System

Metric	Explanation	How to find	Centric
Response Time	Turnaround time of a specific process	completionTime	User
CPU Utilization	what % of time CPU is doing something	busyTime/totalTime	System

First Come First Serve (FCFS)

Ready queue is ordered by arrival time of a process

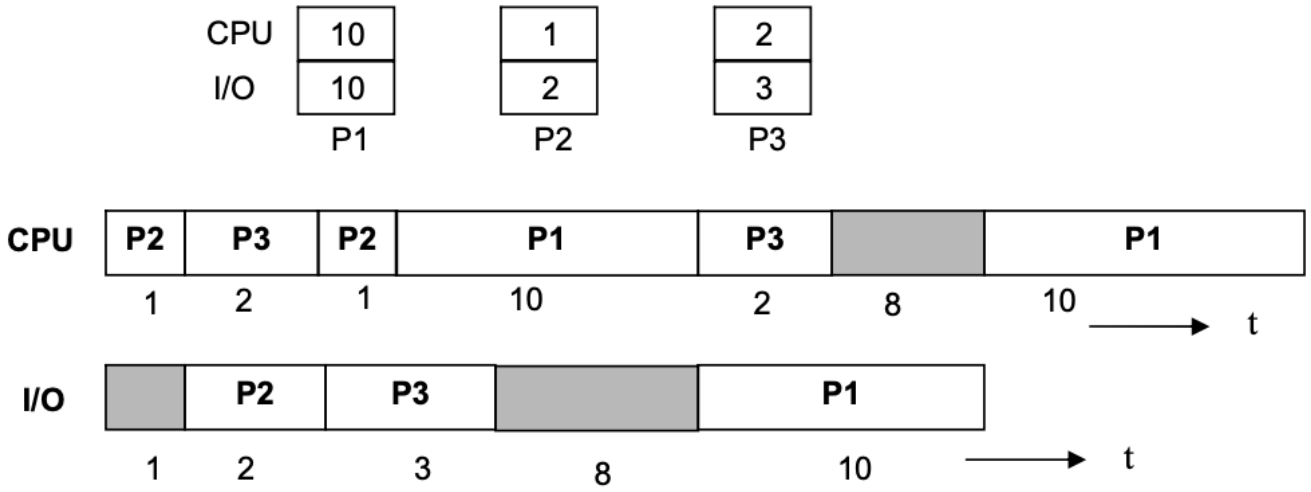


We run into problems like the convoy effect (one process monopolizes resources)

- This is a non-preemptive scheduling algorithm

Shortest Job First (SJF)

Also non-preemptive, we just sort ready queue by CPU burst time needed



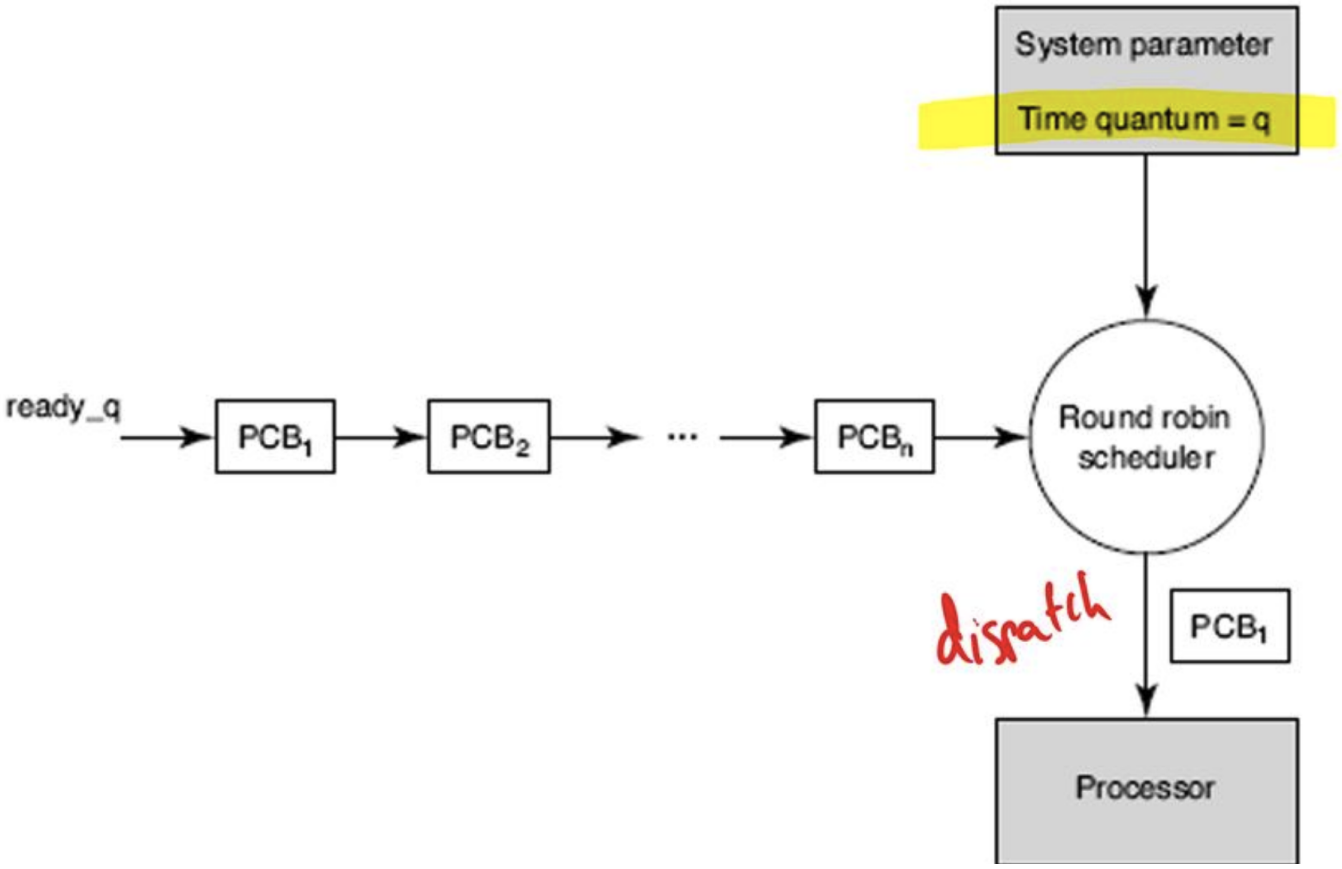
We will run into problems with starvation however (imagine if we had a lot of processes that have really short CPU bursts – the long ones will never be run!)

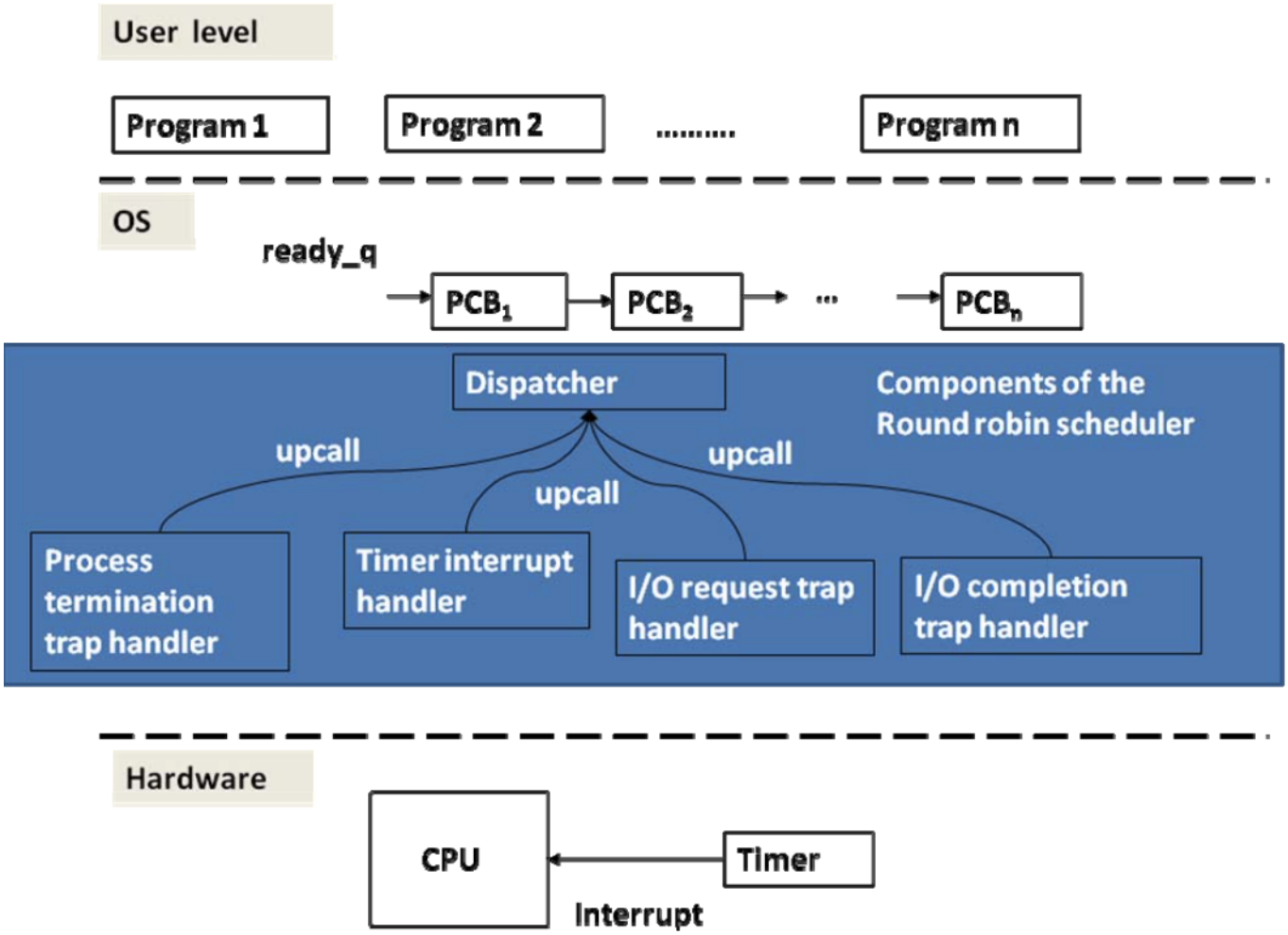
Shortest Remaining Time First (SRTF)

We will always run the program with the shortest remaining time first (in its CPU burst). In the case that another process is still running, we preempt that process and push our shortest process on there

Round Robin

- A hardware device, timer, interrupts the processor when the **time quantum** q expires
- At any point of time, the processor is hosting either one of the user programs or the scheduler
- Upon an interrupt, the timer interrupt handler takes control of the processor
- **Handoff** - handler saves the context of the currently running user program in the associated PCB and hands over control to the scheduler


System parameter
Time quantum = q
ready_q
PCB1
PCB2
...
PCBn
Round robin scheduler
dispatch
PCB1
Processor


User level
Program 1
Program 2
.....
Program n
OS
ready_q
PCB1
PCB2
...
PCBn
Dispatcher
Components of the Round robin scheduler
upcall
upcall
upcall
Process termination trap handler
Timer interrupt handler
I/O request trap handler
I/O completion trap handler
Hardware
CPU
Timer
Interrupt

Scheduler

- get head of ready queue
- set timer
- dispatch

Timer interrupt handler

- save context in PCB
- move PCB to end of ready queue
- upcall to dispatcher

I/O request trap

- save context in PCB
- move PCB to I/O queue
- upcall to dispatcher

I/O completion interrupt handler

- save context in PCB
- move PCB of I/O completed process to ready queue
- upcall to dispatcher

Process termination trap handler

- Free PCB
- Upcall to dispatcher

A preemptive scheduler can be implemented with **any scheduling discipline**
A preemptive scheduler can be implemented with **any scheduling discipline**

Upon context switch, the scheduler saves the volatile state of the current process in the **PCB for that process**