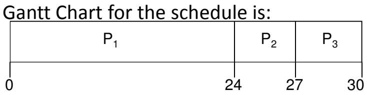
First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



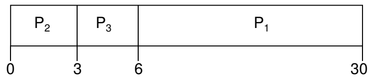
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for P_1 = 6; P_2 = 0; P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- *Convoy effect* short process behind long process

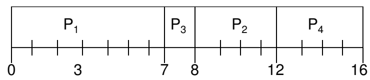
Shortest-Job-First (SJR) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

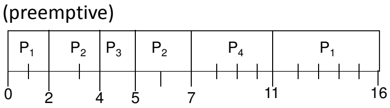


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - nonpreemptive
- 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

Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

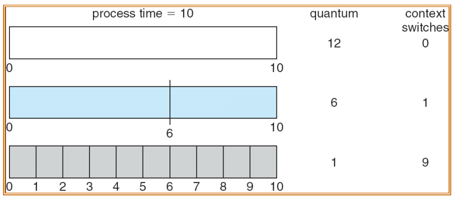
- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

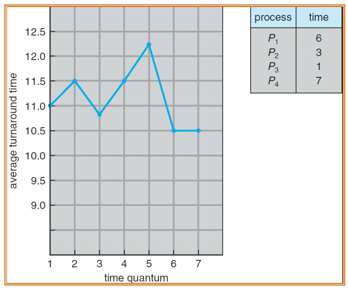
0 20 37 57 77 97 117 121 134 154 162

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

Time Quantum and Context Switch Time



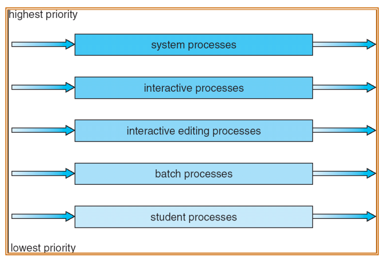
Turnaround Time Varies With The Time Quantum



Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling



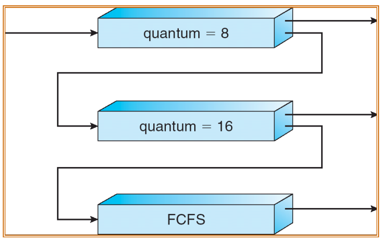
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues



Multiple-Processor Scheduling

- CPU scheduling more complex hen multiple CPUs are available
- *Homogeneous processors* within a multiprocessor
- *Load sharing*
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing

Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM THREADS 5
int main(int argc, char *argv[])
{
    int i;
   pthread t tid[NUM THREADS];
   pthread attr t attr;
   /* get the default attributes */
   pthread attr init(&attr);
   /* set the scheduling algorithm to PROCESS or SYSTEM */
   pthread attr setscope(&attr, PTHREAD_SCOPE_SYSTEM);
   /* set the scheduling policy - FIFO, RT, or OTHER */
   pthread attr setschedpolicy(&attr, SCHED_OTHER);
   /* create the threads */
   for (i = 0; i < NUM THREADS; i++)
        pthread create(&tid[i],&attr,runner,NULL);
```

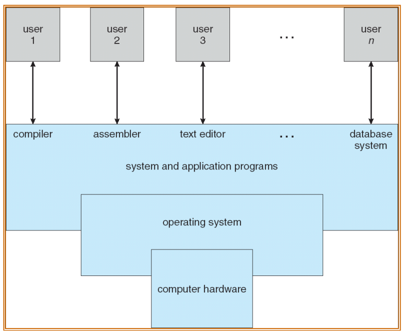
Pthread Scheduling API

```
    /* now join on each thread */
    for (i = 0; i < NUM THREADS; i++)
        pthread_join(tid[i], NULL);
}
 /* Each thread will begin control in this
    function */
void *runner(void *param)
{
    printf("I am a thread\n");
    pthread exit(0);
}
```

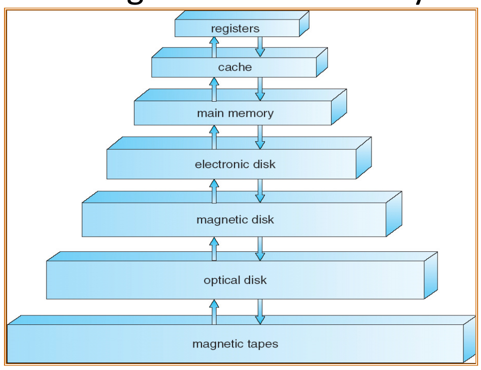
Review Chapter 1

- Overview of the “Operating System”
- Kernel mode/user mode*
- Storage (I/O)
- Memory
- Interrupt *

Four Components of a Computer System



Storage-Device Hierarchy



Chapter 2

- How to design an operating system
- Structure of an “operating system” *
  - Layers
  - Microkernels
  - Modules
  - Comparing them?
- What’s a system call? *
- Eg. Question: What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices?
- Eg.Question: in what ways is the modular kernel approach similar to the layered one? In what ways they’re different?
- Virtual machine?

Chapter 3 Review

- Process Concept *
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication*
- Communication in Client-Server Systems
- RPC (*stub, marshall*)
- RMI

Chapter 3 Review

- Process Scheduling
  - Status diagram
  - I/O-bound process and a CPU-bound process
  - Queues (*many*)
  - Context switch
  - Process life cycle (create, terminate)

Chapter 3 Review

- Interprocess Communication
  - Shared memory
    - Producer-consumer problem
    - Data structure (*circular array*)
  - Message passing
    - Direct/indirect
    - syncronization

Chapter 3 Review

- Communication in Client-Server Systems/RPC/RMI
  - What a socket? Port number?
  - RPC: port? Marshalling?
  - RMI: concept of stub, skeleton, marshalling
  - Compare RPC/RMI

Chapter 4 Threads

- Multithreading Models*
  - 1 to 1
  - 1 to many
  - Many to many
- Threading Issues
  - Fork() and exec()
- PThread/Win32/Java Threading
- Compare thread and process
  - Eg. Q: which of the follwing are shared across threads…
    - Register
    - Heap
    - Globals
    - Stack

Chapter 5

- Basic Concepts
- Scheduling Criteria
  - Turnaround time
  - Waiting time
  - Response time …
- Scheduling Algorithms *
  - FCFS
  - SJF
  - RR
  - Priority …
- Multiple-Processor Scheduling
- *Eg. The exercise handout*