Operating Systems
[5A. CPU Scheduling]

Chung-Wei Lin
cwlin@csie.ntu.edu.tw
CSIE Department
National Taiwan University

From Operating System Concepts (10th Edition)

Objectives

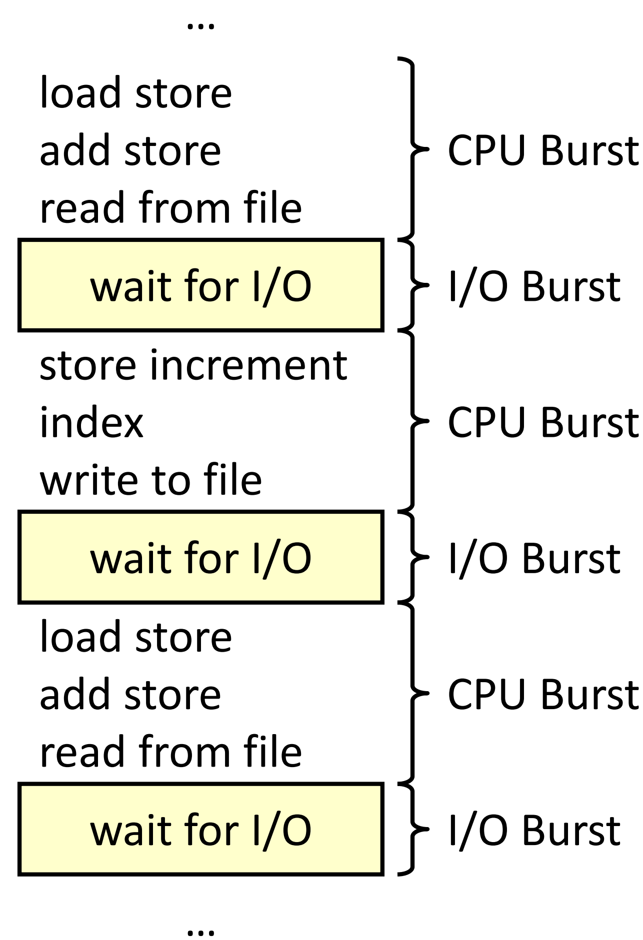
- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on scheduling criteria
- Explain the issues related to multiprocessor and multicore scheduling
- Describe various real-time scheduling algorithms
- Describe the scheduling algorithms used in the Windows, Linux, and Solaris operating systems
- Apply modeling and simulations to evaluate CPU scheduling algorithms

Outline

- **Basic Concepts**
  - **CPU-I/O Burst Cycle**
  - CPU Scheduler
  - Preemptive and Nonpreemptive Scheduling
  - Dispatcher
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating-System Examples
- Algorithm Evaluation

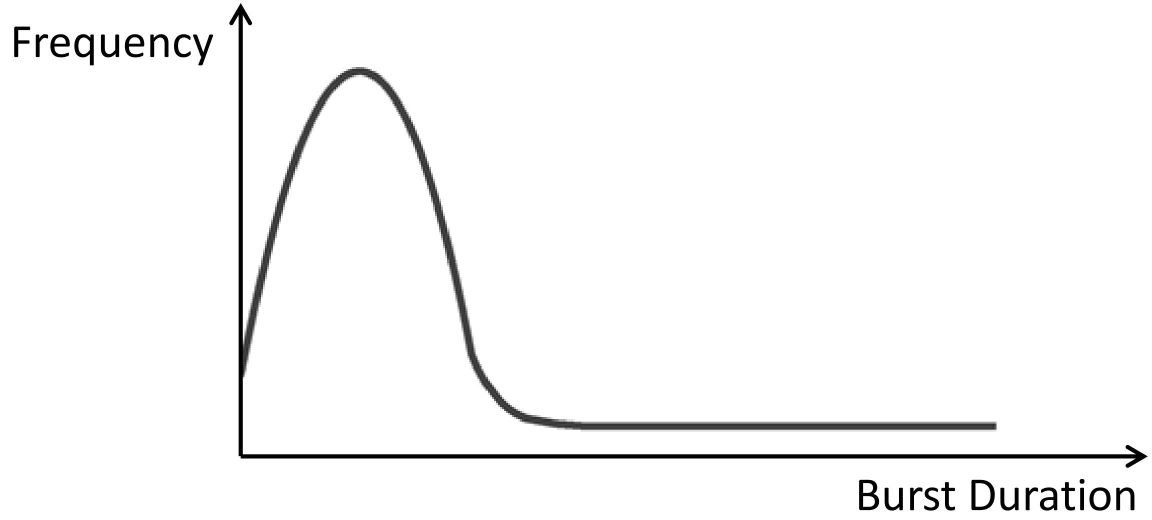
Basic Concepts

- **Objective**
  - Maximize CPU utilization with multiprogramming
- **CPU-I/O burst cycle**
  - Process execution consists of a **cycle** of CPU execution and I/O wait
  - Interleaving **CPU bursts** and **I/O bursts**



Histogram of CPU-Burst Times

- CPU-burst distribution is of main concern
  - A large number of short CPU bursts
  - A small number of long CPU bursts
  - An I/O-bound program typically has many short CPU bursts
  - A CPU-bound program might have a few long CPU bursts

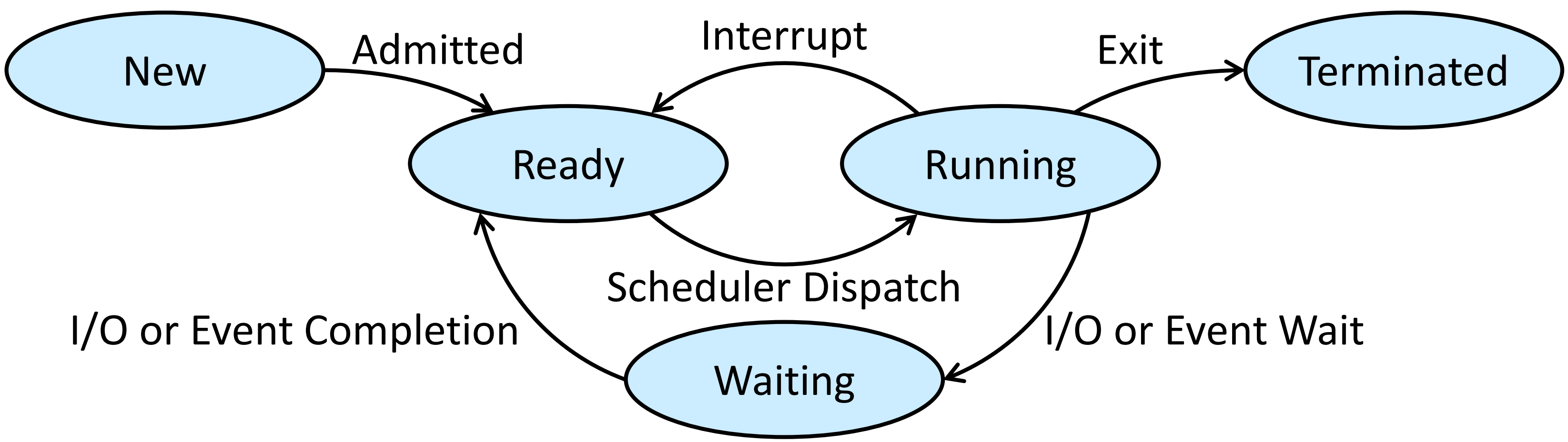


Outline

- **Basic Concepts**
  - CPU-I/O Burst Cycle
  - **CPU Scheduler**
  - Preemptive and Nonpreemptive Scheduling
  - Dispatcher
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating-System Examples
- Algorithm Evaluation

Recap: Process State

- As a process executes, it changes **state**
  - **New**: the process is being created
  - **Ready**: the process is waiting to be assigned to a processor
  - **Running**: instructions are being executed
    - Only one process can be running on any processor core at any instant
  - **Waiting**: the process is waiting for some event to occur
    - Examples: I/O completion, reception of a signal
  - **Terminated**: the process has finished execution



CPU Scheduler

- The **CPU scheduler** selects a process from the processes in memory that are ready to execute and allocates the CPU to it
  - The ready queue may be ordered in various ways
- CPU scheduling decisions may take place when a process
  - Switches from the running state to the waiting state
  - Switches from the running state to the ready state
  - Switches from the waiting state to the ready state
  - Terminates
- Situations 1 and 4
  - There is no choice in terms of scheduling
  - A new process (if one exists in the ready queue) must be selected
- Situations 2 and 3
  - There is a choice

Outline

- **Basic Concepts**
  - CPU-I/O Burst Cycle
  - CPU Scheduler
  - **Preemptive and Nonpreemptive Scheduling**
  - Dispatcher
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating-System Examples
- Algorithm Evaluation

Preemptive and Nonpreemptive Scheduling

- When scheduling takes place only under situations 1 and 4, the scheduling scheme is **nonpreemptive**
  - Once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state
- Otherwise, it is **preemptive**
  - Virtually all modern operating systems including Windows, MacOS, Linux, and UNIX use preemptive scheduling algorithms

Preemptive Scheduling and Race Conditions

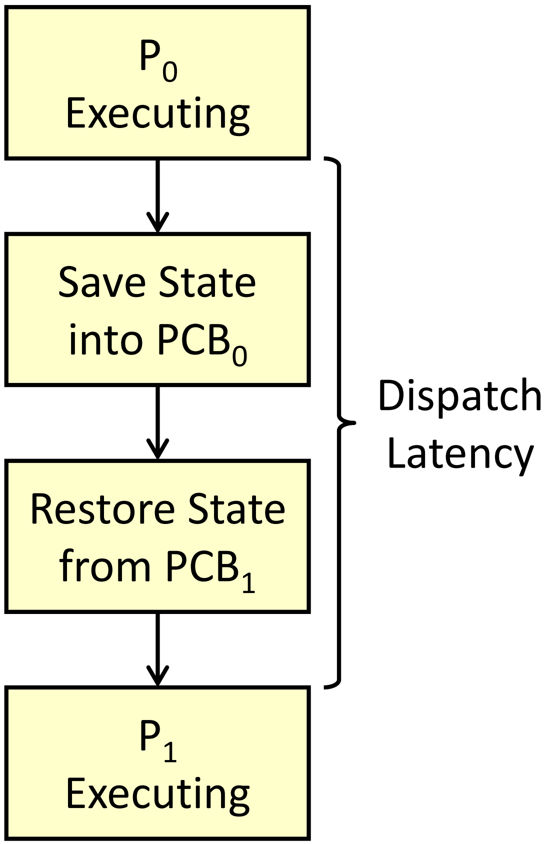
- Preemptive scheduling can result in race conditions when data are shared among several processes
  - Example
    - Two processes share data
    - While one process is updating the data, it is preempted so that the second process can run
    - The second process then tries to read the data, which are in an inconsistent state
- This issue will be explored in detail in
  - Chapter 6: Synchronization Tools
  - Chapter 7: Synchronization Examples

Outline

- **<u>Basic Concepts</u>**
  - CPU-I/O Burst Cycle
  - CPU Scheduler
  - Preemptive and Nonpreemptive Scheduling
  - **<u>Dispatcher</u>**
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating-System Examples
- Algorithm Evaluation

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU scheduler, involving
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- **Dispatch latency**
  - Time it takes for the dispatcher to stop one process and start another running



PCB: Process Control Block

Outline

- ❑ Basic Concepts
- ❑ **Scheduling Criteria**
- ❑ Scheduling Algorithms
- ❑ Thread Scheduling
- ❑ Multi-Processor Scheduling
- ❑ Real-Time CPU Scheduling
- ❑ Operating-System Examples
- ❑ Algorithm Evaluation

Scheduling Criteria

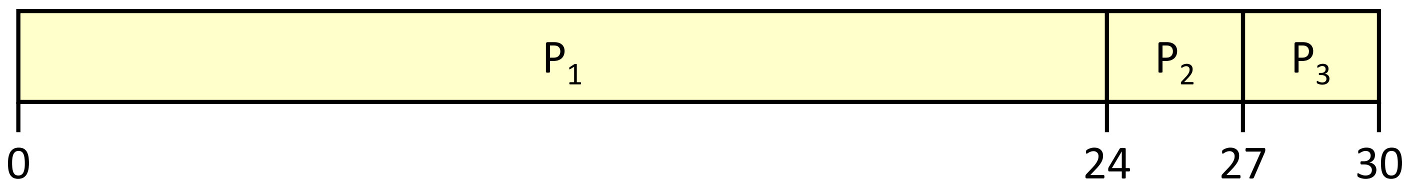
- Maximize **CPU utilization**
  - Keep the CPU as busy as possible
- Maximize **throughput**
  - Number of processes that complete their execution per time unit
- Minimize **turnaround time**
  - Amount of time to execute a particular process
- Minimize **waiting time**
  - Amount of time a process has been waiting in the ready queue
- Minimize **response time**
  - Amount of time it takes from when a request was submitted until the first response is produced

Outline

- Basic Concepts
- Scheduling Criteria
- **Scheduling Algorithms**
  - **First-Come, First-Served Scheduling**
  - Shortest-Job-First Scheduling
  - Round-Robin Scheduling
  - Priority Scheduling
  - Multilevel Queue Scheduling
  - Multilevel Feedback Queue Scheduling
- Thread Scheduling, Multi-Processor Scheduling
- Real-Time CPU Scheduling, Operating-System Examples
- Algorithm Evaluation

First-Come, First-Served (FCFS) Scheduling

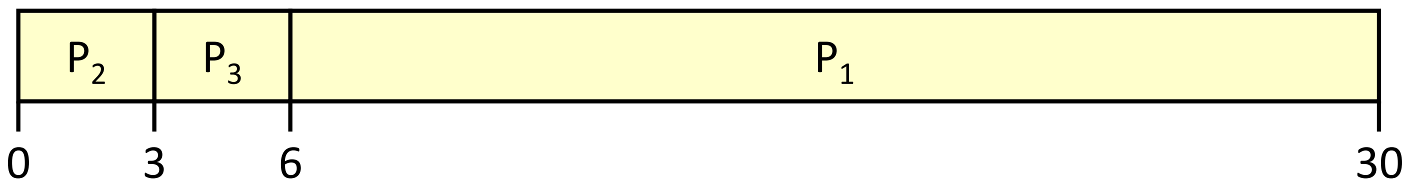
- Example
  - Process: P_1 P_2 P_3
  - Burst time: 24 3 3
- Suppose that the processes arrive in the order: P_1, P_2, P_3
- Gantt Chart for the schedule



- Waiting time
  - $P_1 = 0$, $P_2 = 24$, $P_3 = 27$
- Average waiting time
  - $(0 + 24 + 27) / 3 = 17$

FCFS Scheduling

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



- Waiting time
  - $P_1 = 6$, $P_2 = 0$, $P_3 = 3$
- Average waiting time
  - $(6 + 0 + 3) / 3 = 3$
- **Convoy effect**
  - All other processes wait for one big process to get off the CPU
    - Consider one CPU-bound and many I/O-bound processes
    - Result in lower CPU and device utilization

Outline

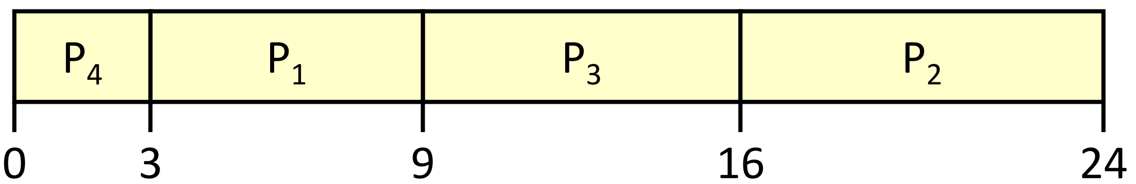
- Basic Concepts
- Scheduling Criteria
- **Scheduling Algorithms**
  - First-Come, First-Served Scheduling
  - **Shortest-Job-First Scheduling**
  - Round-Robin Scheduling
  - Priority Scheduling
  - Multilevel Queue Scheduling
  - Multilevel Feedback Queue Scheduling
- Thread Scheduling, Multi-Processor Scheduling
- Real-Time CPU Scheduling, Operating-System Examples
- Algorithm Evaluation

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal for minimizing average waiting time of a given set of processes
  - Preemptive version called **shortest-remaining-time-first**
- The difficulty is knowing the length of the next CPU request
  - How do we determine the length of the next CPU burst?
    - Estimate
    - Ask the user

SJF Scheduling

- Example
  - Process: P_1 P_2 P_3 P_4
  - Burst time: 6 8 7 3
- SJF scheduling chart



- Average waiting time
  - (3 + 16 + 9 + 0) / 4 = 7

Prediction of Length of Next CPU Burst

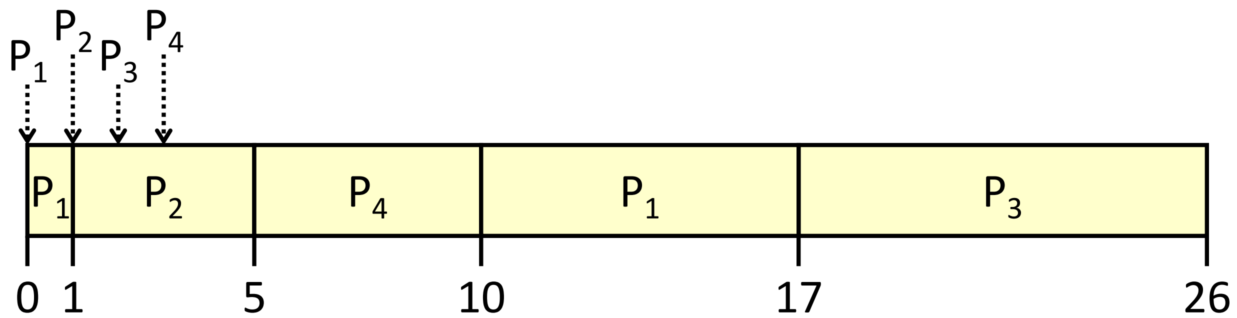
- Should it be similar to the previous one?
  - Pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
  - t_n = actual length of n-th CPU burst
  - τ_{n+1} = predicted value for the next CPU burst
  - α, $0 \leq \alpha \leq 1$
  - Define $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \tau_n$
- Example with $\alpha = 0.5$
  - t_i = X 6 4 6 4 13 13 13 …
  - τ_i = 10 8 6 6 5 9 11 12 …

Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get
  - $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_{n-1} + \ldots + (1 - \alpha)^j\, \alpha\, t_{n-j} + \ldots + (1 - \alpha)^{n+1}\, \tau_0$
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Shortest-Remaining-Time-First Scheduling

- Add the concepts of varying arrival times and preemption
  - Process: P_1 P_2 P_3 P_4
  - Arrival time: 0 1 2 3
  - Burst time: 8 4 9 5
- Preemptive "SJF" scheduling chart



- Average waiting time
  - [(10 – 1 – 0) + (1 – 1) + (17 – 2) + (5 – 3)] / 4 = 26 / 4 = 6.5
  - Please figure out how to compute it by yourself

Outline

- Basic Concepts
- Scheduling Criteria
- **Scheduling Algorithms**
  - First-Come, First-Served Scheduling
  - Shortest-Job-First Scheduling
  - **Round-Robin Scheduling**
  - Priority Scheduling
  - Multilevel Queue Scheduling
  - Multilevel Feedback Queue Scheduling
- Thread Scheduling, Multi-Processor Scheduling
- Real-Time CPU Scheduling, Operating-System Examples
- Algorithm Evaluation

Round-Robin (RR) Scheduling

- Each process gets a small unit of CPU time (**time quantum** q)
  - Usually 10-100 milliseconds
  - After the time has elapsed, the process is preempted and added to the end of the ready queue
    - Timer interrupts every quantum to schedule next process
- Assume n processes in the ready queue
  - Each one gets 1/n of 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: FCFS
  - q small: q must be large, considering context switches
    - Q: If there is only one process of 10 time units, how many context switches are there with q = 1, 6, and 12?
    - A: 9, 1, and 0

RR Scheduling with Time Quantum = 4

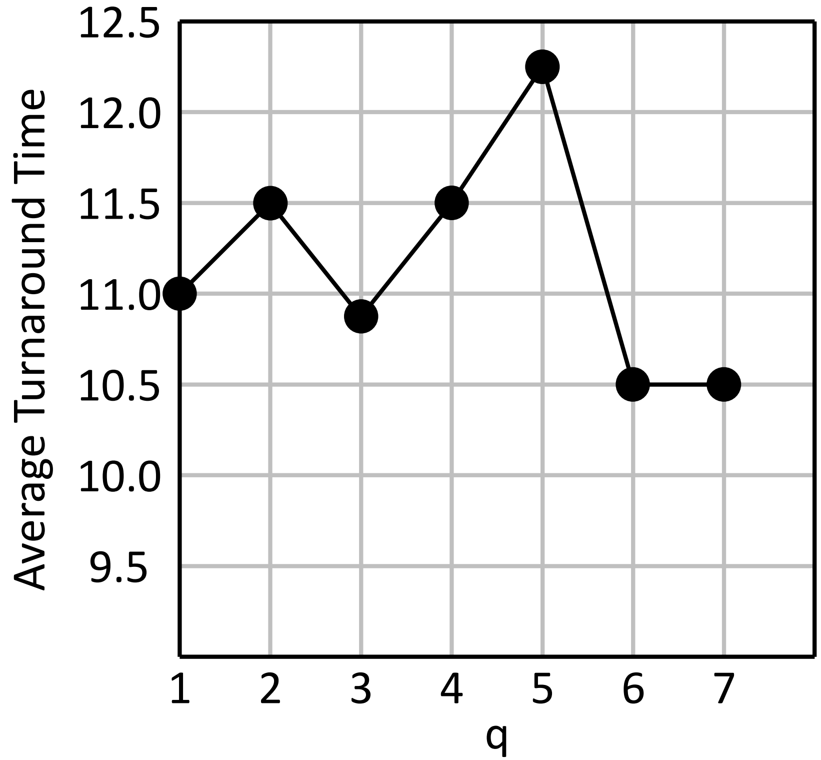
- Example
  - Process: P_1 P_2 P_3
  - Burst time: 24 3 3
- RR scheduling chart

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1
0–4	4–7	7–10	10–14	14–18	18–22	22–26	26–30

- Typically, higher average turnaround than SJF, but better **response**
- q should be large compared to context switch time
  - q is usually 10 milliseconds to 100 milliseconds
  - Context switch < 10 microseconds

Turnaround Time Varies With q

- 80% of CPU bursts should be shorter than q
- Example
  - Process: P_1 P_2 P_3 P_4
  - Burst time: 6 3 1 7



Outline

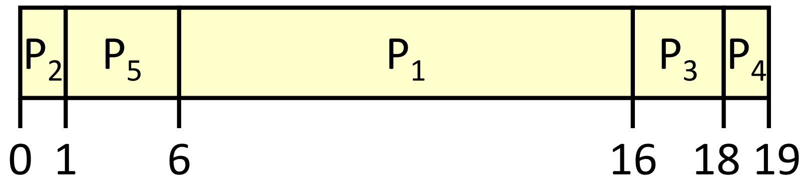
- Basic Concepts
- Scheduling Criteria
- **Scheduling Algorithms**
  - First-Come, First-Served Scheduling
  - Shortest-Job-First Scheduling
  - Round-Robin Scheduling
  - **Priority Scheduling**
  - Multilevel Queue Scheduling
  - Multilevel Feedback Queue Scheduling
- Thread Scheduling, Multi-Processor Scheduling
- Real-Time CPU Scheduling, Operating-System Examples
- Algorithm Evaluation

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 priority scheduling where priority is the inverse of predicted next CPU burst time
- **Starvation**
  - Low priority processes may never execute
- **Aging**
  - As time progresses, increase the priority of the process

Priority Scheduling

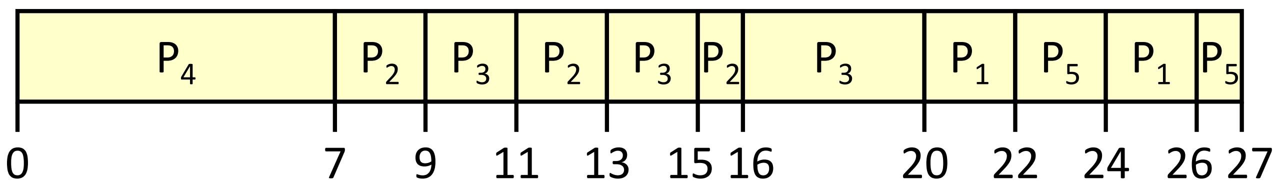
- Example
  - Process: P_1 P_2 P_3 P_4 P_5
  - Burst time: 10 1 2 1 5
  - Priority: 3 1 4 5 2
- Priority scheduling chart



- Average waiting time
  - [6 + 0 + 16 + 18 + 1] / 5 = 8.2

Priority Scheduling w/ Round-Robin

- Example
  - Process: P_1 P_2 P_3 P_4 P_5
  - Burst time: 4 5 8 7 3
  - Priority: 3 2 2 1 3
- Run the process with the highest priority, and processes with the same priority run round-robin
- Scheduling chart with time quantum = 2

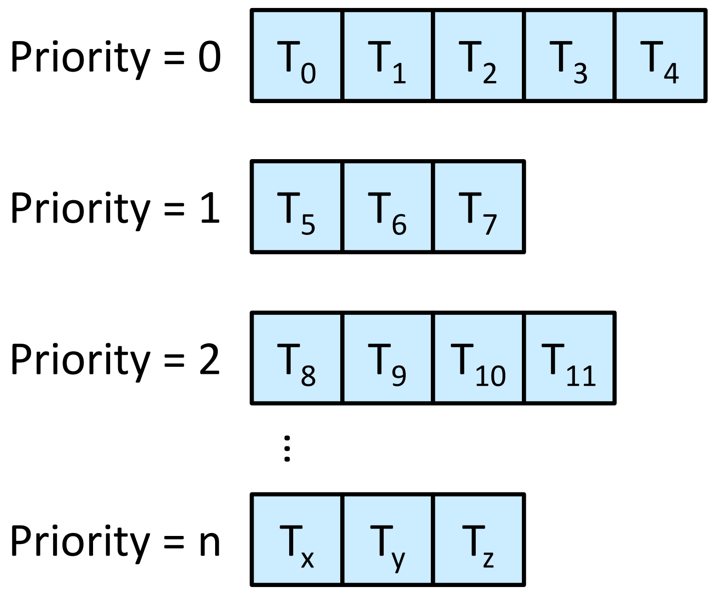


Outline

- Basic Concepts
- Scheduling Criteria
- <u>**Scheduling Algorithms**</u>
  - First-Come, First-Served Scheduling
  - Shortest-Job-First Scheduling
  - Round-Robin Scheduling
  - Priority Scheduling
  - <u>**Multilevel Queue Scheduling**</u>
  - Multilevel Feedback Queue Scheduling
- Thread Scheduling, Multi-Processor Scheduling
- Real-Time CPU Scheduling, Operating-System Examples
- Algorithm Evaluation

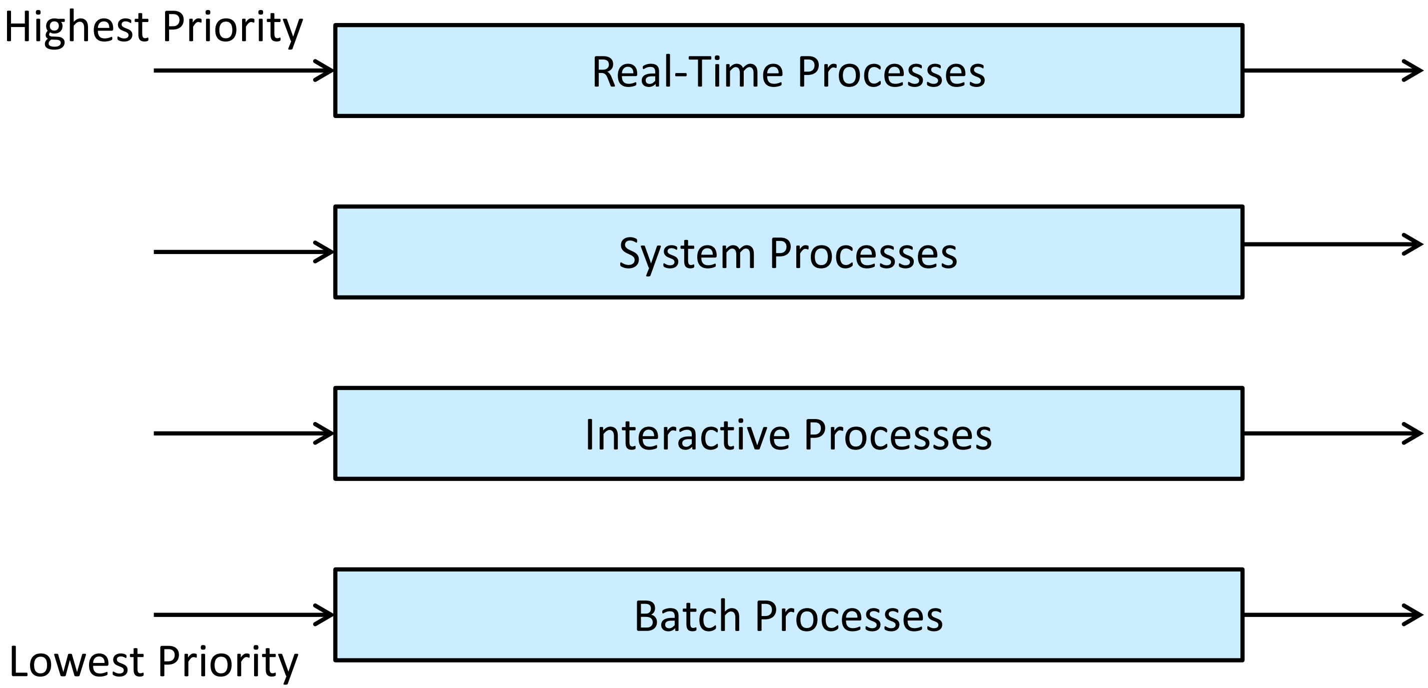
Multilevel Queue Scheduling

- Have separate queues for each priority
- Schedule the process in the highest-priority queue



Multilevel Queue Scheduling

- Prioritization based upon process type
  - Each queue might have its own scheduling algorithm



Outline

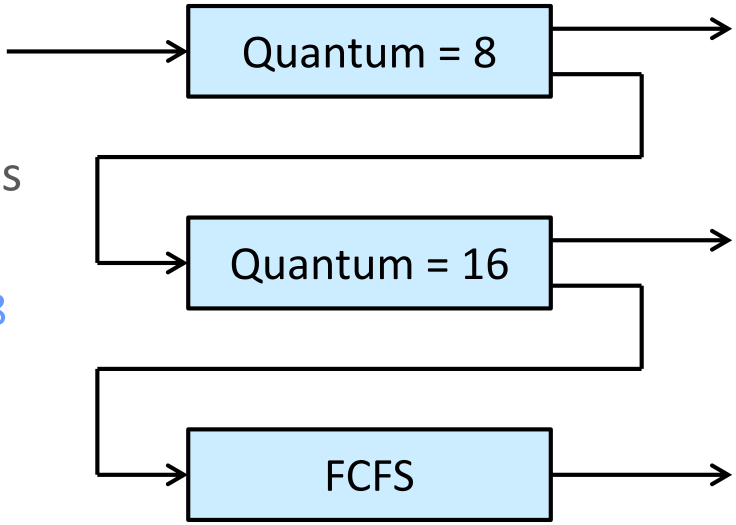
- Basic Concepts
- Scheduling Criteria
- **Scheduling Algorithms**
  - First-Come, First-Served Scheduling
  - Shortest-Job-First Scheduling
  - Round-Robin Scheduling
  - Priority Scheduling
  - Multilevel Queue Scheduling
  - **Multilevel Feedback Queue Scheduling**
- Thread Scheduling, Multi-Processor Scheduling
- Real-Time CPU Scheduling, Operating-System Examples
- Algorithm Evaluation

Multilevel Feedback Queue Scheduling

- A process can move between the various queues
- A multilevel-feedback-queue scheduler is defined by
  - 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
- Aging can be implemented using multilevel feedback queue

Multilevel Feedback Queue Scheduling

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



Outline

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- **Thread Scheduling**
  - Contention Scope
  - Pthread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating-System Examples
- Algorithm Evaluation

Thread Scheduling

- On most modern operating systems, it is kernel-level threads, not processes, that are being scheduled
  - Distinction between user-level and kernel-level threads
    - User-level threads are managed by a thread library, and the kernel is unaware of them
    - To run on a CPU, user-level threads must ultimately be mapped to an associated kernel-level thread
    - The mapping may be indirect and may use a lightweight process (LWP)

Outline

- ❑ Basic Concepts
- ❑ Scheduling Criteria
- ❑ Scheduling Algorithms
- ❑ **Thread Scheduling**
  - ➢ **Contention Scope**
  - ➢ Pthread Scheduling
- ❑ Multi-Processor Scheduling
- ❑ Real-Time CPU Scheduling
- ❑ Operating-System Examples
- ❑ Algorithm Evaluation

Contention Scope

- **Process-contention scope (PCS)**
  - With the many-to-one and many-to-many models, the thread library "schedules" user-level threads to run on an available LWP
    - Scheduling competition is within the process, typically done via priority set by programmers
- **System-contention scope (SCS)**
  - The kernel uses SCS to decide which kernel-level thread to schedule onto a CPU
    - Scheduling competition is among all threads in the system
    - Systems (such as Windows and Linux) with the one-to-one model schedule threads using SCS only

Outline

- ❑ Basic Concepts
- ❑ Scheduling Criteria
- ❑ Scheduling Algorithms
- ❑ **Thread Scheduling**
  - ➢ Contention Scope
  - ➢ **Pthread Scheduling**
- ❑ Multi-Processor Scheduling
- ❑ Real-Time CPU Scheduling
- ❑ Operating-System Examples
- ❑ Algorithm Evaluation

Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
  - **`PTHREAD_SCOPE_PROCESS`** schedules threads using PCS scheduling
  - **`PTHREAD_SCOPE_SYSTEM`** schedules threads using SCS scheduling
- It can be limited by OS
  - Linux and macOS only allow **`PTHREAD_SCOPE_SYSTEM`**

Pthread Scheduling API

```c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM");
      else
         fprintf(stderr, "Illegal scope value.\n");
   }
```

Pthread Scheduling API

```
   /* set the scheduling algorithm to PCS or SCS */
   pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i], &attr, runner, NULL);
   /* 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)
{
   /* do some work ... */
   pthread_exit(0);
}
```

Outline

- ❑ Basic Concepts
- ❑ Scheduling Criteria
- ❑ Scheduling Algorithms
- ❑ Thread Scheduling
- ❑ **Multi-Processor Scheduling**
- ❑ Real-Time CPU Scheduling
- ❑ Operating-System Examples
- ❑ Algorithm Evaluation

Q&A

Practice Exercise 1

- Example
  - Process: P_1 P_2 P_3
  - Arrival time: 0.0 0.4 1.0
  - Burst time: 8 4 1
- Use nonpreemptive scheduling
  - What is the average turnaround time for these processes with the FCFS scheduling algorithm?
  - What is the average turnaround time for these processes with the SJF scheduling algorithm?
  - Is SJF scheduling algorithm the optimal one? Why?

Practice Exercise 1

- **Example**
  - Process: P_1 P_2 P_3
  - Arrival time: 0.0 0.4 1.0
  - Burst time: 8 4 1
- **Use nonpreemptive scheduling**
  - What is the average turnaround time for these processes with the FCFS scheduling algorithm?
    - $P_1, P_2, P_3 \rightarrow ((8 - 0) + (12 - 0.4) + (13 - 1)) / 3 = 10.53$
  - What is the average turnaround time for these processes with the SJF scheduling algorithm?
    - $P_1, P_3, P_2 \rightarrow ((8 - 0) + (9 - 1) + (13 - 0.4)) / 3 = 9.53$
  - Is SJF scheduling algorithm the optimal one? Why?
    - $P_2, P_3, P_1 \rightarrow ((4.4 - 0.4) + (5.4 - 1) + (13.4 - 0)) / 3 = 7.27$
    - $P_3, P_2, P_1 \rightarrow ((2 - 1) + (6 - 0.4) + (14 - 0)) / 3 = 6.87$

Practice Exercise 2

- We have actually introduced a set of scheduling algorithms
- What is the set relation between the following pairs of algorithms (or algorithm sets)?
  - Priority and FCFS
  - RR and FCFS
  - Multilevel feedback queues and FCFS
  - Priority and SJF
  - RR and SJF

Midterm Updates

- Responses to midterm survey
  - Some wording
  - Early-bird policy
  - TA
- Midterm exam
  - Performance
  - Difficulty of having same questions for two sections
    - Emphasized concepts, wording
    - Versus "separated sections"; "same lecture before/after midterm"
  - Reason of having many questions
  - Grading policy
  - Regrade request
    - Mainly for mis-clicked rubric items or mis-categorized/mis-read your answers
  - Grade interpretation