Process Scheduling

B.Ramamurthy

Introduction

- An important aspect of multiprogramming is scheduling. The resources that are scheduled are IO and processors.
- The goal is to achieve
  - High processor utilization
  - High throughput
    - number of processes completed per unit time
  - Low response time
    - time elapse from the submission of a request to the beginning of the response

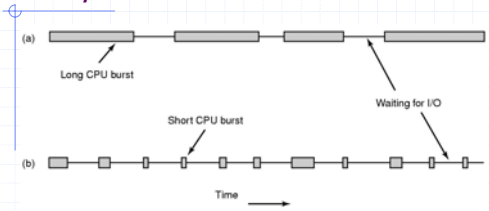
Topics for discussion

- Motivation
- Types of scheduling
- Short-term scheduling
- Various scheduling criteria
- Various algorithms
  - Priority queues
  - First-come, first-served
  - Round-robin
  - Shortest process first
  - Shortest remaining time and others
- Queuing Model and Performance Analysis

The CPU-I/O Cycle

- We observe that processes require alternate use of processor and I/O in a repetitive fashion
- Each cycle consist of a CPU burst (typically of 5 ms) followed by a (usually longer) I/O burst
- A process terminates on a CPU burst
- CPU-bound processes have longer CPU bursts than I/O-bound processes

CPU/IO Bursts



- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process
  - an I/O bound process

Motivation

- Consider these programs with processing-component and IO-component indicated by upper-case and lower-case letters respectively.

A1 a1 A2 a2 A3
0 30 50 80 120 130 ===> JOB A
B1 b1 B2
0 20 40 60 ====> JOB B
C1 c1 C2 c2 C3 c3 C4 c4 C5
0 10 20 60 80 100 110 130 140 150
=>JOB C

Motivation

- The starting and ending time of each component are indicated beneath the symbolic references (A1, b1 etc.)
- Now lets consider three different ways for scheduling: no overlap, round-robin, simple overlap.
- Compare utilization U = Time CPU busy / Total run time

Types of scheduling

- Long-term : To add to the pool of processes to be executed.
- Medium-term : To add to the number of processes that are in the main memory.
- **Short-term** : Which of the available processes will be executed by a processor?
- IO scheduling: To decide which process's pending IO request shall be handled by an available IO device.

Scheduling Algorithm Goals

All systems
Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems
Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

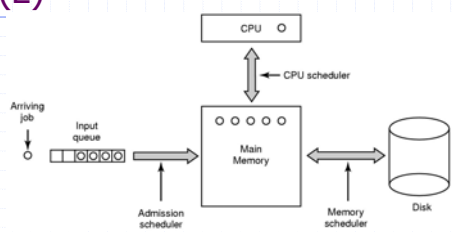
Interactive systems
Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems
Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

Scheduling in Batch Systems (1)

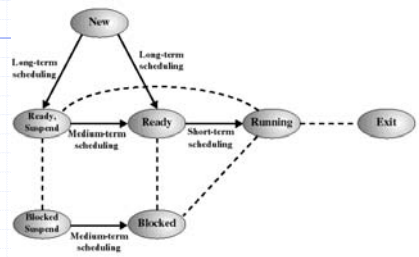
An example of shortest job first scheduling

Scheduling in Batch Systems (2)



Three level scheduling

Classification of Scheduling Activity



- Long-term: which process to admit
- Medium-term: which process to swap in or out
- Short-term: which ready process to execute next

Short-term scheduling

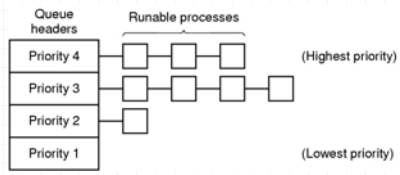
- Short-term scheduler is invoked whenever an event occurs that may lead to the interruption of the current process or that may warrant the preemption of the current process in favor of another.
- Clock-interrupts, IO interrupts, OS calls, signals, real-time system policies are some such events.
- The main objective of short-term scheduling is to allocate processor time in such a way as to optimize one or more aspects of system behavior.

Short-term scheduling criteria

- **User-oriented:**
  - **Response time**: is the elapsed time between the submission of a request until the response begins to appear at the output.
  - Typical goal of such system should be maximize the number of users who would experience average or better than average response time.
- **System-oriented:**
  - **Throughput**: is the rate at which processes are completed.
  - Focus is on the efficient and effective use of processors.
  - **Utilization** : % of time a processor is kept busy.

Priority queues

- An important aspect of scheduling is the use of priorities.
- One ready queue for each priority level;
- Or one priority queue. Insertion into the queue is done according to the priority.
- When released from a blocked queue, a process enters a queue according to its priority.



Policy versus Mechanism

- Separate what is allowed to be done with how it is done
  - a process knows which of its children threads are important and need priority
- Scheduling algorithm parameterized
  - mechanism in the kernel
- Parameters filled in by user processes
  - policy set by user process

Alternative scheduling policies

- A selection function determines which among the ready processes is next selected for execution Selection functions are based on one or more of these items:

1) w - time spent in system so far, waiting and executing.
2) e - time spent on execution so far.
3) s - total service time required by the process.
4) resources required by the process

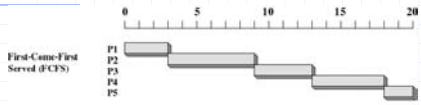
- When to apply the selection function is dependent on whether a **preemptive or non-preemptive** scheduling is used.

Case-study

- Lets study each of policies with a sample problem:

Process	Arrival time	Service time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

First Come First Served (FCFS)



- Selection function: the process that has been waiting the longest in the ready queue (hence, FCFS)
- Decision mode: nonpreemptive
  - a process run until it blocks itself

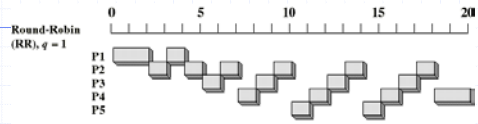
FCFS drawbacks

- A process that does not perform any I/O will monopolize the processor
- Favors CPU-bound processes
  - I/O-bound processes have to wait until CPU-bound process completes
  - They may have to wait even when their I/O are completed (poor device utilization)
  - we could have kept the I/O devices busy by giving a bit more priority to I/O bound processes

Round robin

- CPU is time-sliced among the ready processes.
- Time-quantum slightly higher than the typical interaction time.
- Appears fair but tends to favor CPU-bound jobs.

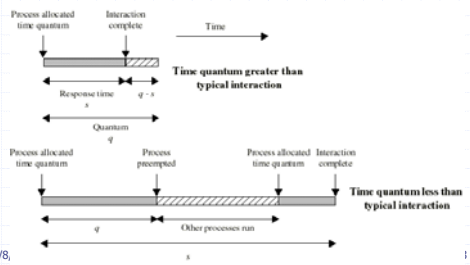
Round-Robin



- Selection function: same as FCFS
- Decision mode: preemptive
  - a process is allowed to run until the time slice period (quantum, typically from 10 to 100 ms) has expired
  - then a clock interrupt occurs and the running process is put on the ready queue

Time Quantum for Round Robin

- must be substantially larger than the time required to handle the clock interrupt and dispatching
- should be larger then the typical interaction (but



Round Robin: critique

- Still favors CPU-bound processes
  - A I/O bound process uses the CPU for a time less than the time quantum and then is blocked waiting for I/O
  - A CPU-bound process run for all its time slice and is put back into the ready queue (thus getting in front of blocked processes)
- A solution: virtual round robin
  - When a I/O has completed, the blocked process is moved to an auxiliary queue which gets preference over the main ready queue
  - A process dispatched from the auxiliary queue runs no longer than the basic time quantum minus the time spent running since it was selected from the ready queue

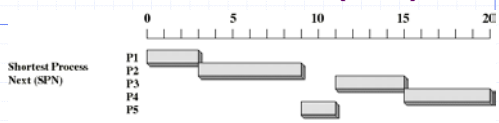
Shortest process next

- This is of special interest since it requires prediction of the next process sizes depending on the history.
- What are the mechanisms available for such prediction?
- Simple averaging? Exponential averaging?
- Lets study exponential averaging.
- Exponential tracks changes fast; larger value of alpha results more rapid reaction to changes in observed values.
- Possible starvation of larger jobs.

Shortest process next (contd.)

- Assume that a process is made of several cpu bursts. Each process has its own sequence of burst. From the pool of process now ready to run, OS has to predict the one that will have shortest burst.
- Initial process burst size is set to 0. (So = 0). Why? Want to let the process have priority over the existing ones so that you may be able get an idea of its burst times.

Shortest Process Next (SPN)



- Selection function: the process with the shortest expected CPU burst time
- Decision mode: nonpreemptive
- I/O bound processes will be picked first
- We need to estimate the required processing time (CPU burst time) for each process

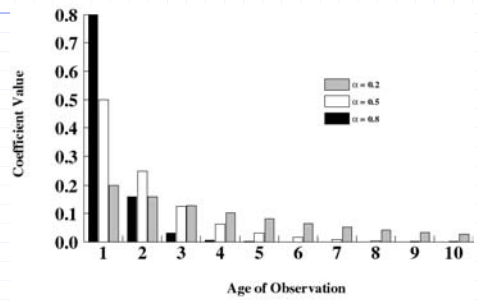
Estimating the required CPU burst

- Let T[i] be the execution time for the ith instance of this process: the actual duration of the ith CPU burst of this process
- Let S[i] be the predicted value for the ith CPU burst of this process. The simplest choice is:
  - $S[n+1] = (1/n) \Sigma_{\{i=1 \text{ to } n\}} T[i]$
- To avoid recalculating the entire sum we can rewrite this as:
  - $S[n+1] = (1/n) T[n] + ((n-1)/n) S[n]$
- But this convex combination gives equal weight to each instance

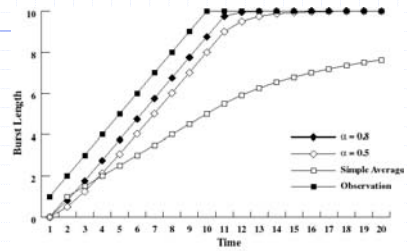
Estimating the required CPU burst

- But recent instances are more likely to reflect future behavior
- A common technique for that is to use exponential averaging
  - $S[n+1] = \alpha\, T[n] + (1-\alpha)\, S[n]$; $0 < \alpha < 1$
  - more weight is put on recent instances whenever $\alpha > 1/n$
- By expanding this eqn, we see that weights of past instances are decreasing exponentially
  - $S[n+1] = \alpha T[n] + (1-\alpha)\alpha T[n-1] + \ldots (1-\alpha)^i \alpha T[n-i] + \ldots + (1-\alpha)^n S[1]$
  - predicted value of 1st instance S[1] is not calculated; usually set to 0 to give priority to to new processes

Exponentially Decreasing Coefficients



Exponentially Decreasing Coefficients



- Here S[1] = 0 to give high priority to new processes
- Exponential averaging tracks changes in process behavior much faster than simple averaging

Shortest Process Next: critique

- Possibility of starvation for longer processes as long as there is a steady supply of shorter processes
- Lack of preemption is not suited in a time sharing environment
  - CPU bound process gets lower priority (as it should) but a process doing no I/O could still monopolize the CPU if he is the first one to enter the system
- SPN implicitly incorporates priorities: shortest jobs are given preferences
- The next (preemptive) algorithm penalizes directly longer jobs

Other policies

- Turnaround time = Finish time - Arrival time
- Normalized turnaround time = Turnaround time / service time
- Response time = arrival time - start time
- Overall wait time = response time + wait times in the ready queue (ready to run, but CPU not avail)
- Multilevel Priority Queues

Scheduling in Real-Time Systems

Schedulable real-time system

- Rate Monotonic Scheduling:
- Given
  - *m* periodic events
  - event *i* occurs within period P_i and requires C_i seconds
- Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Summary

- Scheduling is important for improving the system performance.
- Methods of prediction play an important role in Operating system and network functions.
- Simulation is a way of experimentally evaluating the performance of a technique.