Lecture VIII
Scheduling

§1. Introduction

- What are we really scheduling?

  Processes on CPU(s). Hence this is also called "CPU scheduling"

  WHY? Because each CPU can run only one process at any given moment. But we won't talk much about multiprocessor scheduling.

  But we can also be scheduling printing jobs, etc.

- Long-term vs. short-term scheduler:

  (1) Short-term: CPU scheduling, picking next process to run.

  (2) Long-term (or Job scheduling): picking processes from a large pool (in disk) and loading them in memory (for the short-term scheduler). This is called SWAPPING.

  The shortterm scheduler runs frequently and so must be fast.

- HISTORY:

  Batch processing days.

  Interactive

  Real time systems.

- What are the elements in scheduling?

  We must remember that processes are (in simplest model) in one of three states: READY, RUN, WAIT.

  Scheduling amounts to putting a ready-process into the RUN state. THERE IS AT MOST ONE PROCESS IN RUN STATE AT ANY MOMENT.

- What is the simplest scheduling model?

  FCFS: there is a FIFO queue for READY processes. The scheduler takes a process from the queue and puts it into RUN state. When it terminates, we repeat.

- SCHEDULING EVENT: this is defined to be an event that the scheduler may have to re-act to.

  For FCFS, the scheduling events are:

  (1) when a process terminates

  (2) when a process blocks or waits.

- PRE-EMPTIVE SCHEDULER: schedulers that may interrupt a running process.

  Pre-emption is essential in real time systems, and in interactive applications.

  FCFS is an example of a NON-preemptive scheduler.

  PRE-EMPTIVE SCHEDULER: such schedulers has two other kinds of events:

  (3) when a process goes from WAIT to READY state

  (4) when a process goes from RUN to READY state

  Actually, event (4) might be associated with the alarm going off at the end of a time quantum.

  Such alarm clocks are tied to hardware clock that goes off every 50 or 60 Hz.

- What do we need to achieve pre-emptive scheduling?

  (1) The pre-empted process must first get into some "consistent" state. E.g., it may have to finish up a critical task, release some resources, etc.

  (2) The environment of pre-empted process must be saved. This takes less than 10 microseconds in current computers.

  (3) Pre-emption may not be disallowed in some system calls.

  (4) The software module that passes control of the CPU to another process is called the DISPATCHER.

- CRITERIA for evaluating Schedulers:

  1. CPU utilization
  2. Throughput (this is not the same as CPU utilization because there are other devices besides CPU, e.g., disks)
  3. Wait time (i.e., time for the job to be first run, or time in ready queue)

     There is another idea of "response time", but that is hard to distinguish from "wait time" in our simple framework.
  4. Turn around time (total time till completion, including wait time)
  5. Fairness (e.g., proportionality of turn around time to job length)
  6. Balance (goal is to keep all units busy all the time)
  7. Predictability (in real time systems, guaranteed turn around, etc)
  8.

- Burst Classification:

  We classify the CPU execution time into alternating CYCLES of CPU burst and I/O burst.

Each "burst" is a time period.

Based on this, we roughly classify processes into 2 classes:

(a) CPU-bound processes have long CPU bursts, and few I/O bursts.

(b) I/O-bound processes have short CPU bursts and many I/O bursts.

SCHEDULERS should try to take advantage of these characteristics.

- Gantt Charts:

  These show a time line, divided into each intervals, each interval is associated with the processes that runs in that interval.

  If there are multiprocessors, we have as many time lines as processors.

- BATCH SCHEDULING ALGORITHMS

  1. Batch scheduling is still useful – e.g., for a high-end printer, for a network, etc.
  2. First Come First Serve (FCFS).
     Simplest.
     E.g., 3 jobs with CPU time (24 secs, 3 sec, 3 sec)
  3. Shortest Job First (SJF).
     With each job, we associate the LENGTH OF THE NEXT CPU BURST.
     E.g., 4 jobs with CPU bursts (6 secs, 8 sec, 7 sec, 3 sec)
     THEOREM: If all jobs are released at the same time, then SJF gives the minimum average wait time.
     CATCH: All jobs must arrive at the same time.
     CATCH: Don't know the length of next CPU burst.
     Solution: estimate.
     It t_n is actual time of the nth CPU burst, we estimate e_{n+1} as follows: $e_{n+1} = \alpha t_n + (1-\alpha)e_n$.
  4. Shortest Remaining Time First (SRTF).
     This only makes sense in a preemptive setting.
     We must guess the remaining time.
  5.

- Policies vs Mechanism

Exercises

Exercise 1.1: *As CPU's become faster (relative to I/O devices), this makes CPU scheduling more critical.*

Argue both sides.

◊

Exercise 1.2: Give a method to give scheduling priority to shorter processes. ♢

Exercise 1.3: To give scheduling priority to IO-Bound processes, we need some way estimating the probability that the process is IO-Bound. How can we do this? ♢

END EXERCISES