CS418— Operating Systems

Lecture 9

Processor Management, part 1

Textbook: Operating Systems
by William Stallings

1. Basic Concepts

- Processor — is also called CPU (Central Processing Unit).
- Process — an executable program, also called task, activity.
- Job — a unit of work that is submitted by the user. A job is composed a set of processes.
- Processor management for a single-user system is easy: set a job either idle or busy.
- Multiprogramming — many users with many jobs on the same system.
- Processor management can be further divided into two parts: **Job scheduler** which is at a high-level and **Process scheduler** which is at a low-level.
- We will mainly focus on process scheduler.

2. Process Scheduler

- **Process scheduler** assigns the CPU to execute the processes of those jobs placed in the READY queue by the **Job Scheduler**.

- There are 2 classes of jobs: I/O-oriented and CPU-oriented.

- Each process in the system is represented by a data structure called **Process Control Block (PCB)**.

- PCBs are usually linked into queues.

- What makes a good scheduling policy?
  - 1. Maximize **throughput**: running as many jobs as possible in a fixed period of time. /* running short jobs or jobs with no interrupts */
  - 2. Minimize **response time**: satisfying interactive requests. /* running only interactive jobs, letting batch jobs wait */
  - 3. Minimize **turnaround time**: moving entire jobs in and out of the system quickly. /* running batch jobs first */
  - 4. Minimize **waiting time**: moving jobs out of READY queue as quickly as possible. /* reducing the number of users */
  - 5. Maximize **CPU efficiency**: keeping CPU busy 100% of the time. /* running only CPU-bound jobs */
  - 6. Ensure fairness for all jobs: giving each job an equal amount of CPU and I/O time. /* disregarding priority */
- **Preemptive scheduling policy:** A scheduling strategy that interrupts the processing of a job and transfers the CPU to another job.
- **Nonpreemptive scheduling policy:** A scheduling strategy that does not allow external interrupts.

3. Scheduling Algorithms

- First Come First Serve (nonpreemptive)
  - 1. The earlier the jobs arrive, the sooner they are served.
  - 2. No WAIT queue is needed (as there is no interrupt).

- Shortest Job Next (nonpreemptive)
  - 1. Does not work in an interactive system.
  - 2. SJN is optimal when all the jobs are available at the same time and CPU times are estimated accurately.

- Priority Scheduling (nonpreemptive)
  - 1. One of the most common methods used in batch systems.
  - 2. High-priority jobs will be run first, tie is broken by arriving time.
  - 3. It is usually hard to set priorities. **Non-technical factor:** position, fee. **Technical factor:** Memory requirements, # of peripheral devices, total CPU time, time in system (**aging**).

- Shortest Remaining Time (preemptive)
  - 1. CPU time is divided into small fragments.
  - 2. SRT will try to finish the job closest to completion.
  - 3. Not suitable for interactive systems.
  - 4. **Context switching**: When a job is preempted, all its running information must be kept.

- Round Robin (preemptive)
  - 1. CPU time is divided into small fragments (slices).
  - 2. Suitable for interactive systems.
  - 3. First come first serve.
  - 4. Slices too large $\rightarrow$ FCFS.
  - 5. Slices too small $\rightarrow$ too much context switching.

- Highest Response Ratio Next (Nonpreemptive)
  - 1. Objective: run jobs to minimize (turnaround_time/service_time)-ratio.
  - 2. As both turnaround_time and service_time cannot be known in advance in most situations, we approximate them based on past history (especially when the job takes several slices to run).
  - 3. Approximate ratio $R = \frac{(waiting\_time+expected\_service\_time)}{expected\_service\_time}$.
  - 4. Aging $\rightarrow$ a job has a long waiting time $\rightarrow$ it's ratio R gets bigger $\rightarrow$ it will get serviced earlier.

- Multi-level Queues
- It is a combination of some of the previous algorithms.
- Example: in a system which handles both batch and interactive jobs, we have a background queue for the first one and a foreground queue for the latter ones.
- *How do we switch queues? How fair is this?*
  - 1. No movement between queues. *Good for high-priority jobs.* High-priority queue empties first.
  - 2. With movement between queues. **Feedback Scheduling:** *Divided CPU time into slices and once a job is in system its priority is disregarded — it can be preempted and moved to the end of the next lower queue.* This is the fairest if the jobs are divided into CPU-bound and I/O-bound. *Good for I/O-bound jobs.*
  - 3. Movement between queues with variable time slices. *Highest-priority queue get time slices t. Second highest-priority queue get time slices $2t$. ....* A job in system can be preempted and moved to the end of the next lower queue. *Good for CPU-bound jobs. It also handles aging well.*

4. Threads

- Multithreading — some modern operating systems support multiple threads of execution within a single process.

- In a multithread environment, a process is defined as the unit of protection and the unit of resource allocation.
  - Each process contains a virtual address space that holds the information related to it.
  - Each process contains protected access to processors, other processes, files and I/O devices.

- Benefits of using threads — *Performance*: it takes less time to create a new thread than to create a new process.

- Given a process, there might be several threads, each with the following:
  - A thread execution state
  - A saved thread context when not running
  - An execution stack
  - Some pre-thread static storage for local variables
  - Access to the memory and resources of its process (threads in one process all have the same access)

- Thread states:
  - Spawn
  - Block (wait)
  - Unblock (ready)
  - Finish

5. Symmetric Multiprocessing

- Flynn proposed the following categories of computer systems in 1972.
  - Single Instruction Single Data (**SISD**) stream. Example, a PC runs MS-DOS.
  - Single Instruction Multiple Data (**SIMD**). Example, an array of processors doing numeric computations.
  - Multiple Instruction Single Data (**MISD**) stream. *Never been implemented.*
  - Multiple Instruction Multiple Data (**MIMD**) stream. *Most commonly used in practice.*

- In a symmetric multiprocessor the system kernel can execute on any processor.

- Clusters (Coarse Grained Multicomputers, Coarse Grained Multiprocessors) received a lot of attention since 1990s. Examples: PVM (Parallel Virtual Machine) and MPI (Massive Parallel Interface).