Operating Systems (UE18CS302)

Aronya Baksy

August 2020

1 Process Scheduling

- While one process waits for some device or I/O response, the scheduler loads one more process to keep the CPU busy in the meantime.
- Such an arrangement increases the level of multiprogramming, and keeps the CPU busy at all times, with no idle periods of inefficiency.
- The success of this arrangement is due to the fact that processes commonly alternate between periods of heavy CPU activity (a **CPU burst**) and periods of high I/O activity (an **I/O burst**).
- The frequency distribution of CPU and I/O bursts (CPU bound progs have few long CPU bursts, I/O bound progs have many short CPU bursts, and vice versa for I/O bursts) is important when choosing an appropriate scheduling algorithm.

1.1 Preemptive and Non Preemptive Scheduling

- CPU scheduling takes place in the following four situations:
    1. Process switching from **running** to the **waiting** state, for an I/O or memory access, or invocation of wait() for child process termination.
    2. Process switching from **running** to **ready**, when an interrupt occurs.
    3. Process switching from **waiting** to **ready** state, for example when an I/O op. completes.
    4. Process **terminates**
- The situations 1 and 4 come under the purview of **non-preemptive** or **cooperative** scheduling.
- Situations 2 and 3 come under **preemptive** scheduling.
- Under non-preemptive scheduling, once a process is allocated the CPU, it takes control of the CPU until it has to wait or it terminates.

- Preemptive scheduling can lead to race conditions. If one process is preempted while it is writing to some data location, and the new process reads from that location, then that data is an inconsistent state.
- Preemptive scheduling can also affect kernel design. If a kernel process is preempted while it is changing some kernel data (eg: an I/O queue), then chaos ensues. UNIX family OSes solve this problem by waiting for the entire system call or I/O op to complete before performing the context switching. This is a working solution but is not feasible in real time OSes.
- The **dispatcher** gives control of the CPU to the process that is selected by the short term scheduler. This involves a context switch, a switch to user mode, and jumping to the appropriate location in the user program to start execution.
- Every process switch invokes the dispatcher, and the time taken for the dispatcher to stop one process and start another is called the **dispatch latency**.

1.2 Scheduling Criteria

- **CPU Utilization:** The fraction of time the CPU is kept busy. Theoretically can be anywhere from 0 to 100%, but real world values range from 40% to 90%.
- **Throughput:** Number of processes completed per unit time.
- **Turnaround Time:** Interval between process submission and process completion. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
- **Waiting Time:** Sum of all time periods spent by process in the wait queue.
- **Response Time:** Time between process submission and first response.

maximize: CPU Util, Throughput **minimize:** Turnaround time, wait time, response time

1.3 Scheduling Algorithms

1.3.1 First Come First Serve (FCFS) Scheduling

- Processes are added to the ready queue in the order in which they arrive.
- The average wait times for FCFS depend very heavily on the process mix, and the burst times for each process. A long process arriving behind many short ones can lead to short wait time, but a long process in front of short ones may lead to longer wait times.

- The effect of a long process in front of many short processes, making the short processes wait for CPU time, is called the **convoy effect**.
- The FCFS is non preemptive because it allocates CPU time entirely to the new process, and it causes problems in time sharing systems where each user must get CPU time at regular intervals.

1.3.2 Shortest Job First (SJF) Scheduling

- The process with the shortest CPU burst time is selected for CPU time.
- To predict the length of the next CPU burst, in a long term system the process time limit per user can be used, which is supplied by the user.
- For a short term scheduling situation, the next CPU burst time is predicted using an exponential average of the previous CPU burst times.

  $$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n \tag{1}$$

  Where τ_i refers to the *predicted* CPU burst value at the time index i, and t_i refers to the *actual* CPU burst time at time index i.
- The parameter α is chosen to be a constant with value < 1, most often a value of 0.5
- The preemptive version of SJF, also called shortest waiting-time first, where the process with the shortest CPU burst time is chosen, and the current process is pre-empted in its place.