Lecture 2: Scheduling and Synchronization

Scheduling in an OS

- When does the scheduling process run?
- When a process:
  1. Terminates
  2. Blocks for I/O
  3. Gets pre-empted
  4. Becomes ready

First Come First Serve (FCFS)

In this example, all processes arrive at the same time (a batch).

The turn-around time using FCFS will be:
- P1: 5
- P2: 10
- P3: 15
- P4: 20
- P5: 25
- P6: 30
- P7: 35

Average turn-around time is 20 sec.

Can you schedule the jobs in any other way to improve the turn-around time?

First Come First Serve (FCFS)

Again, all processes arrive at the same time (a batch).

The turn-around time using FCFS will be:
- P1: 20
- P2: 21
- P3: 22
- P4: 23
- P5: 24
- P6: 25
- P7: 26

Average turn-around time is 23 sec.

Can you schedule the jobs in any other way to improve the turn-around time?
Shortest Job First

| 1 | 1 | 1 | 1 | 1 | 1 | 20 |

Again, all processes arrive at the same time (a batch).

The turn-around time using FCFS will be:
- P1: 26
- P2: 1
- P3: 2
- P4: 3
- P5: 4
- P6: 5
- P7: 6

Average turn-around time is 6.7 sec.

What did this affect?

Let’s look at a dynamic system

| 1 | 1 | 1 | 1 | 1 | 20 |

Here, assume that P1 and P2 arrive at 0, and an infinite number of small processes arrive after P2, at a rate of one process per second.

What could go wrong?

How can you improve it?

- Shortest Remaining Time Next
  - Scheduler runs in cases 1, 2, and 4
- Round Robin
  - Scheduler runs in cases 1, 2, and 3
- And there are many more policies designed for different types of systems.

Multi-threading

Using Java

Threads

- Sometimes called light-weight processes
- Threads are very useful when a process has multiple tasks to perform independently of the others.
  - To speed up a computation
  - To optimally use resources
- Threads within a process can share the process data
- Demo time!
Synchronization

Using Semaphores

Why do we need synchronization?
- Control over the thread sequence
- Mutual exclusion

There are many OS synchronization primitives
- We’ll discuss Semaphores

What’s a Semaphore?
- It’s a variable whose numeric value can only be changed through two methods:
  - Sleep / acquire / wait / p
  - Wakeup / signal / release / v
- The acquire() method:
  \[
  \text{IF } S > 0 \text{ THEN } S := S - 1 \text{ ELSE (wait on S)}
  \]
- The signal() method:
  \[
  \text{IF } (\text{one or more processes are waiting on } S) \text{ THEN (let one of these processes proceed) ELSE } S := S + 1
  \]

Mutual exclusion example
- Semaphore s = new Semaphore (1);

P1

\[
\begin{align*}
&\text{// Some code} \\
&s.\text{acquire();} \\
&\text{//Critical section} \\
&s.\text{release();} \\
&\text{//rest of code}
\end{align*}
\]

P2

\[
\begin{align*}
&\text{// Some code} \\
&s.\text{acquire();} \\
&\text{//Critical section} \\
&s.\text{release();} \\
&\text{//rest of code}
\end{align*}
\]
Rendezvous example

- Semaphore s1 = new Semaphore (0);
- Semaphore s2 = new Semaphore (0);

P1
// Some code
// Rendezvous point
s1.release();
// rest of code

P2
// Some code
// Rendezvous point
s2.release();
// rest of code

Group Exercise

Barber shop problem

Solution

**Semaphores:**
- sofa: 4
- chairs: 3
- worker: 3
- cashier: 1
- done: 0
- pay: 0
- receipt: 0

**Customer:**
- enter();
- sofa.acquire();
- sit_on_sofa();
- cashier.acquire();
- sofa.release();
- sit_in_chair();
- ready.acquire();
- chair.release();
- go_to_cashier();
- pay.acquire();
- receipt.acquire();
- leave();

**Barber:**
- ready.acquire();
- worker.acquire();
- cut_hair();
- done.release();
- worker.release();
- chairs.release();

**Cashier:**
- pay.acquire();
- worker.acquire();
- make_transaction();
- worker.release();
- Receipt.release();