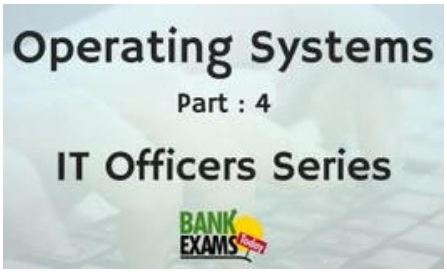
Operating Systems - Part 4

Published on Wednesday, February 17, 2016

Hello Readers,



Today, we'll take up '**process synchronization and CPU scheduling**' part of OS.
Previous article on Threads can be found at Operating Systems - Part 3.
Any doubts/Clarifications - please speak up in comments.

We know that processes in OS can execute concurrently and concurrent access can lead to data inconsistency. To maintain the consistency, there has to be a system of orderly execution of processes. For that, we have the concept of critical section in OS.

Critical Section

That part of the program where the **shared memory is accessed** is called critical section. This part may not be concurrently executed by more than one process at a time. Each process must ask permission to enter critical section with ENTRY section and then with EXIT section, comes out of that and works with remainder section. Following code snippet shows the access of shared memory by process:

```
do {
        entry section
            critical section
        exit section
            remainder section
} while (true);
```

There are 3 principles involved in dealing with critical section:

1. Mutual Exclusion

If one process is executing in critical section, no other process is allowed entry to that.

2. Progress

If no process is currently in critical section and some processes want to enter then only those processes which are not executing in their remainder section can

participate in decision making that which one will enter the critical section next.

3. Bounded Waiting

Once a process enters critical section, it can not enter again until a waiting process gets its turn. Entry is managed as a queue.

Many systems even provide hardware support for implementing critical section code. They are based on idea of locks. The algorithm for using locks looks something like as below:

```
do {
        acquire lock
                critical section
        release lock
                remainder section
} while (TRUE);
```

Semaphores

A semaphore is a protected integer variable that can facilitate and restrict access to shared resources in multi programming environment. They were invented by **Edsger Djikstra**. Two most common kinds of semaphores are Binary and Counting semaphores.

Counting semaphores represent multiple resources and **binary semaphore** represent two possible states i.e 0 or 1 (locked and unlocked). They are accessed by two operations i.e **Wait ( )** and **Signal( ).**

Deadlock and Starvation

Deadlock is a situation where two or more processes are **waiting indefinitely** for an event that an be caused by only one of the waiting process.

Starvation is a situation where a process may **never be removed** from semaphore queue in which it is suspended.

CPU Scheduling

CPU scheduler selects from among the processes in ready queue which are to be allocated CPU next. CPU scheduling decision takes places when a process moves through following states:

- Running to Waiting
- Running to Ready
- Waiting to Ready
- Termination

Scheduling for all of above transitions is non preemptive in nature. All of rest are preemptive in nature. **Non Preemptive scheduling** is when CPU is given to a process, it can not be taken away unless process finishes execution where as

preemptive scheduling is when a high priority task is allocated the CPU while interrupting the currently running process. It is based on idea that highest priority process should always be the process that is currently utilized.

Dispatcher

Dispatcher gives **control of CPU** to process selected by the scheduler. It involves **switching context**, switching to user mode and jumping to proper location to restart the program. **Dispatch latency** is defined as the time it takes for dispatcher to stop one process and start another.

More on scheduling will be covered in next article.

Interesting fact about OS

97% of world's fastest computers are powered by Linux and its dominance continues to rise.