CS 3305A

Multiprogramming

Lecture 7
Sept 30th 2019
Multiprogramming

- Assume we have two processes \( p \) and \( q \)
- Process \( p \) has an instruction that requires a read/write from/to disk
- Reading from disk is slow
- Why not have instructions from \( q \) execute while \( p \) is waiting?
Multiprogramming

- **Multiprogramming** allows for the execution of multiple processes
- But only one process **active** at any time
Why Multiprogramming?

- Operating systems allow for interleaved execution
  - On a single-processor system, no more than one process ever runs.
  - However, one process's instructions may be executed before the completion of the instructions from another process.
- The **objective** is to have some process running at all times in order to maximize CPU utilization.
Process Switching

- Current process executes an I/O operation
- OS needs to be able to suspend current process so that another process can execute
- This is referred to as context switching
Process Switching

- OS needs to be able to suspend current process
- OS captures information about a process
- Information captured must be sufficient to restore the hardware to the same configuration it was in when the process was switched out.
Characterizing a Process

- Each process is represented in the OS by a **process control block (PCB)** which contains all the state for a program in execution including (but not limited to):
  - Pointer to text, data etc. information
  - The program counter (PC) indicating the next instruction
  - Current values of the set of general-purpose registers
  - A set of operating system resources e.g., open files, network connections
  - Process identifier (PID)
  - Process priority (for scheduling purposes)
  - etc.
Process Execution States

- As a process executes, it changes execution state.
- The execution state of a process is defined in part by the current activity of the process.
- A process may be in one of the following execution states:
  - **New**: The process is being created.
  - **Ready**: The process is waiting to be assigned to a processor.
  - **Running**: Instructions are being executed.
  - **Waiting**: The process is waiting for some event to occur (such as an I/O completion or reception of signal).
  - **Exit**: The process has finished executing.
- Only **one** process can be **running** on any processor at any instant.
- **Many** processes may be **ready** and **waiting**.
Scheduling

- The purpose of multiprogramming is to have a process running at all times.
- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each process.
- The **process scheduler** selects an available process.
- There may be multiple processes to select from.
Scheduling Queues

- As processes enter the system, they are put into a **job queue**, which consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.
- Queues are implemented using linked list.
Process Execution States

- **New**: When you run a program, a new process is created.

- **Ready**: The process is loaded into memory and placed in the ready queue. If the system has sufficient memory, then the new process is loaded into memory and placed in the ready queue.

- **Running**: CPU scheduler takes a process from the head of a ready queue to execute. (Sometimes, there may be multiple ready queues.)

- **Exit**: When the wait for I/O is over, there is a return to the Ready state. If processes are scheduled in a round robin manner, then, when time quantum expires, the process is returned to the ready queue.

- **Waiting**: Event Wait