Process Management

Harsh Vardhan

A process can be thought as a program in execution.

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

- New: The process is being created.
- Running: The instructions are being executed.
- Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- Ready: The process is wating to be assigned to a processor.
- Terminated: The process has finished execution.

Each process is represented in the operating system by a *process control block* (PCB) - also called *task control block*. It contains many pieces of information, like :

- Process State
- Program Counter
- CPU registers
- CPU-scheduling information
- Memory Management information
- Account Information
- I/O status information

Context Switch

When an interrupt occurs the operating system needs to save the current context of the process running on the CPU, so that it can restore that context when the processing is done, essentially suspending the process and then resuming it.The context is represented in the PCB of the process. It includes the value of the CPU registers, the process state and memory management information. Generally, we perform a state save of current state of the CPU be it in kernel or user mode, and then state restore to resume operations.

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch. Context switch time is pure overhead because the system does no useful work while switching.The Processsor cache switches between processes and cache misses might occur while switching and reading data from the memory. When the cache is hot most process data is in the cache so the process performance will be at its best.

Process Creation

A new process is created by the *fork()* system call. The new process consists of a copy of address space of the original process.This mechanism allows parent process to communicate easily with its child process. Both processes continue execution at the instruction after the *fork()*, with one difference: the return code for the fork() of new(child) process is zero, whereas (nonzero) process identifier of the child is returned to the parent.

After a *fork()* system call, one of the two system processes typically uses *exec()* system call to replace the process's memory space with a new program.

Interprocess Communication

Processes executing concurrently in the operating system may either be independent processes or cooperating processes. A process is independent if it cannot affect or be affected by any other processes that is being executed in the system. Any process that does not share any data with any other process is independent. A process is cooperating if it can affect or be affected by the other processes executing in the system.

Reasons for providing an environment that allows cooperation:

- Information Sharing
- Computation Speedup
- Modularity
- Convenience

Cooperating processes require an interprocess communication(IPC) mechanism that will allow them to exchange data and information. There are two fundamental models of interprocess communication: *shared memory* and *messaging passing*. In shared memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region. In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes.Message passing model is easier to implement in a distributed system than shared memory. Shared memory suffers from cache coherency issues, which arise because shared data migrates among the several caches.