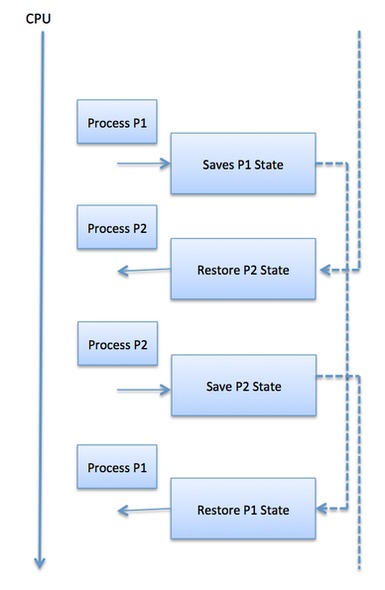
**Unit II - Process Management**

## Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.



**Process Operations:**

### Process Creation

There are two major operations Process Creation is one out of them. This is made through appropriate system calls, such as fork or spawn, processes may create other processes. The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.

There are two options for the parent process after creating the child:

Wait for the child process to terminate before proceeding. Parent process makes a wait() system call, for either a specific child process or for any particular child process, which causes the parent process to block until the wait() returns. UNIX shells normally wait for their children to complete before issuing a new prompt.

Run concurrently with the child, continuing to process without waiting. When a UNIX shell runs a process as a background task, this is the operation seen. It is also possible for the parent to run for a while, and then wait for the child later, which might occur in a sort of a parallel processing operation.

There are also two possibilities in terms of the address space of the new process:

* The child process is a duplicate of the parent process.
* The child process has a program loaded into it.

**Process Termination**

By making the exit(system call), typically returning an int, processes may request their own termination. This int is passed along to the parent if it is doing a wait(), and is typically zero on successful completion and some non-zero code in the event of any problem.

Processes may also be terminated by the system for a variety of reasons, including :

* The inability of the system to deliver the necessary system resources.
* In response to a KILL command or other unhandled process interrupts.
* A parent may kill its children if the task assigned to them is no longer needed i.e. if the need of having a child terminates.
* If the parent exits, the system may or may not allow the child to continue without a parent (In UNIX systems, orphaned processes are generally inherited by init, which then proceeds to kill them.)