**OPERATION ON PROCESS**

An O.S. must provide a mechanism for process creation and termination.

**PROCESS CREATION**

* A process can create a several new processes, via a create process system call, during the course of execution. The creation process is called a parent process, whereas the new processes are called children of that process.
* Each of these new processes may in turn create other processes, forming a tree of processes.
* In general, a process will need certain resources such as CPU time, memory, files, I/O devices to accomplish its task.
* The parent process may have to partition its resources among its children, it may be able to share some resources among several of its children. Restricting a child process to a subset of parents resources prevents any process from overloading the system by creating too many sub processes.
* When a process creates a new process, two possibilities exists.

In terms of execution.

1. The parent process continues to execute concurrently with the children.
2. The parent waits until some or all children have terminated in terms of address space.

In terms of address space.

1. The child process is duplicate of parent process.
2. The child process has new program loaded into it.

* Most O.S. identify processes according to unique process identifier which is typically integer number.

**fork()**

* It is the creation of child process that differs from the parent process only in PID (process ID) and PPID (parent process ID), and in the fact that resource utilization are set to zero. This mechanism allows the parent process to communicate easily with its child process.
* Both processes (parent & child) continue execution at the instruction after the fork() system call, with one difference: the return code for the fork() system call is zero for the new (child) process, whereas (non-zero) process identifier of the child is returned to the parent.
* If the call to fork() is executed successfully then

1. Make two identical copies of address spaces, one for the parent and one for the child.
2. Both processes will start their execution at the next statement following the call. In this case both the processes will start their execution.

**exec()**

* It is a type of system call used after fork() system call by one of the two processes to replace the processes memory space with a new program.
* The exec() system call loads a binary file into memory- destroying the memory image of the program containing exec() system call- and start its execution. In this manner, two process are able to communicate, and then go their separate ways.

**PROCESS TERMINATION**

A process terminate when it finishes executing its final statement and ask the O.S. to delete it by using the exit() system call. At this point, the process may return a status value to its parent process. All the resources of the process including physical and virtual memory, open file, and I/O buffer – are deallocated by the O.S.

Termination can occur in other circumstances as well. A process can call the termination of another process via appropriate system call. Such a system call can be evoked only by the parent of the process that is to be terminated. Otherwise users could arbitrarily kill each other’s job.

A parent can terminate the execution of one of its children for a variety of reasons such as

1. The child has exceeded its usage of some of resources that it has been allocated.
2. The task assigned to child is no longer required.
3. The parent is exiting, and O.S. does not allow child to continue if its parent terminates.

**Cascading termination**

If a process terminates then all its children will also be terminated, this phenomenon is called cascading termination.