PROCESS

A process or task is an instance of a program in execution. A process is a running program with some specific tasks to do. A process is more than the program code, which is some times known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the Processor's registers. It generally includes the process stack, data section, heap during process run time.

Note A program by itself is not a process.

Difference bet^n a Program and a Process

PROGRAM	PROCESS
① A program is a static object that exist in a file.	① A process is a dynamic object that is a program in execution.
② A program is a sequence of instructions. in any Lang.	② A process is a sequence of instruction execution in machine code
③ A program loaded into a secondary storage devices.	③ A process loaded into main memory.
④ Time span of program is unlimited	④ Time span of process is limited.
⑤ A program is a passive entity.	⑤ A process is an active entity.

Every time we run a command shell runs that commands as a separate process. When a process is created, it requires several resources such as CPU time, memory files, stack, registers to run the program. In other words A process is a tuple. (process id, code, data, register, values, PC value)

where :

⇒ process id is unique in the system.

⇒ code is the program code.

⇒ data is the data used during its execution i.e. the data space and the values of the data

⇒ register values are the values in the machine registers and

⇒ PC value is the address in the program code.

PROCESS STATES :- The lifetime of a process can be divided into several stages or states. As a process starts executing, it goes through one state to another state.

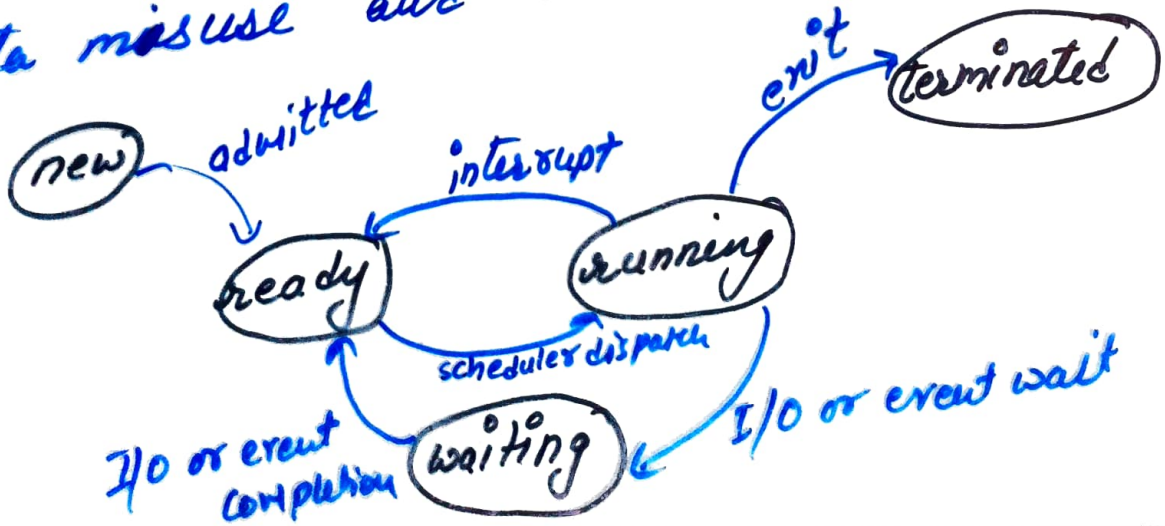
NEW :- The process has been created. Any Program i.e. is created or submitted to the computer is in the new state.

READY :- The process is waiting to be assigned to a CPU when it is time to select a process to run, The operating system selects one of the jobs from the ready queue and move the process from ready state to running state.

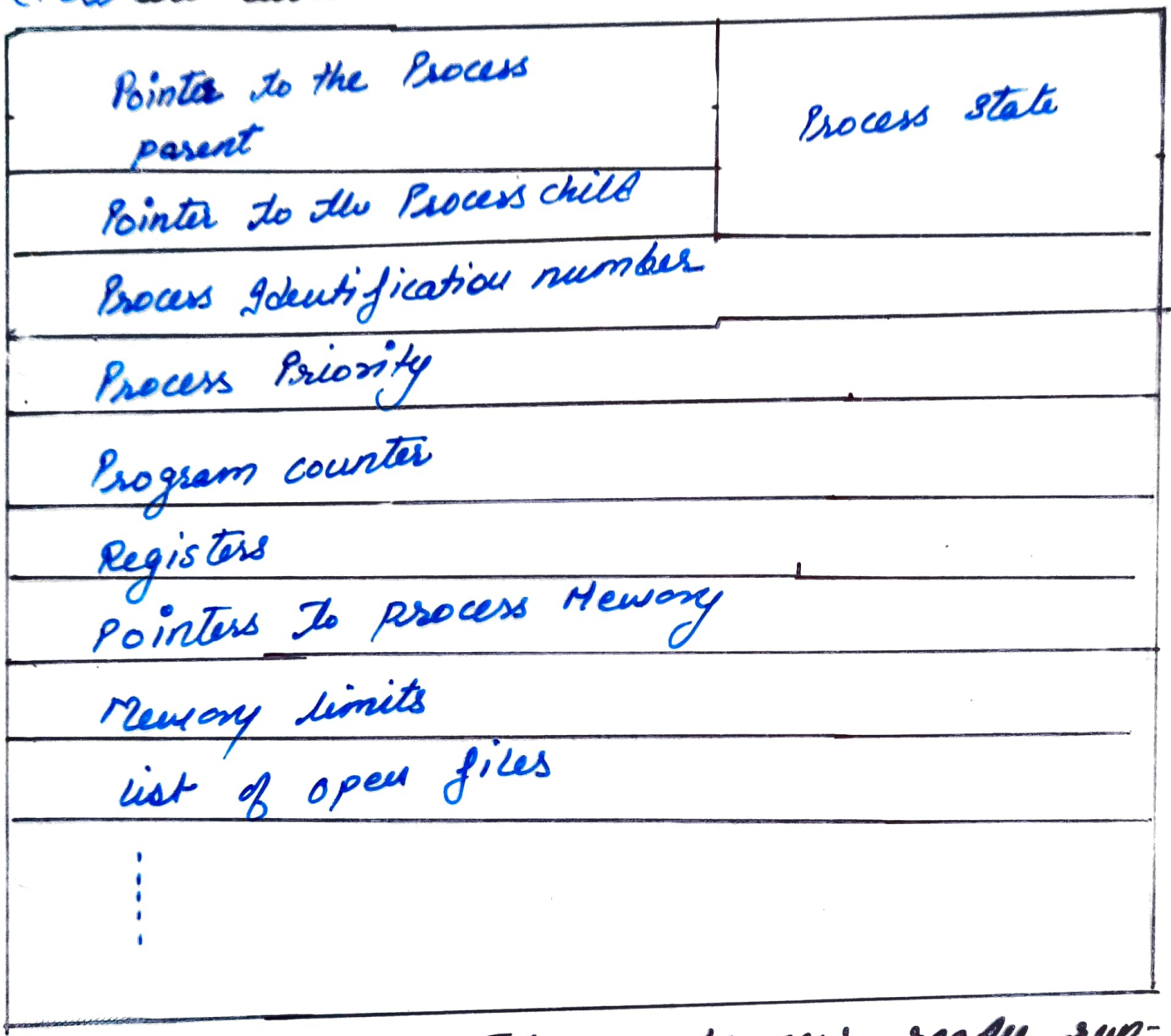
RUNNING :- A CPU is currently allocated to the Process and the process is in execution, or you can say instructions are being executed : when a process gets a control from CPU plus other resources it starts executing.

WAITING / BLOCKED / SUSPENDED :- A process is put into the waiting state, if the process need an event occur, or an I/O devices require. The OS does not provide the I/O or event immediately then the process moved to waiting state by the operating system.

TERMINATE :- when the execution of a process has completed, then the OS terminates that process from running state. Some times operating system terminates the process some other reasons also include Time limit exceeded, memory unavailable, access violation, Protection error, I/O failure, data misuse and soon.



PROCESS CONTROL BLOCK :- The operating system groups all information that it needs about a particular process into a data structure called a Process Control Block (PCB) also called a Task Control Block.



Process state :- The state may be new, ready, running, waiting, haltes and soon.

Program counter :- The counter indicates the address of the next instruction to be executed for this process.

CPU registers :- It include accumulators, index registers, stack pointers and general purpose registers plus any condition code information.

CPU - Scheduling Information :- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory - Management Information :- It includes such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system.

Accounting Information :- It includes the amount of CPU and real time used, time limits, account numbers, job or process numbers and soon.

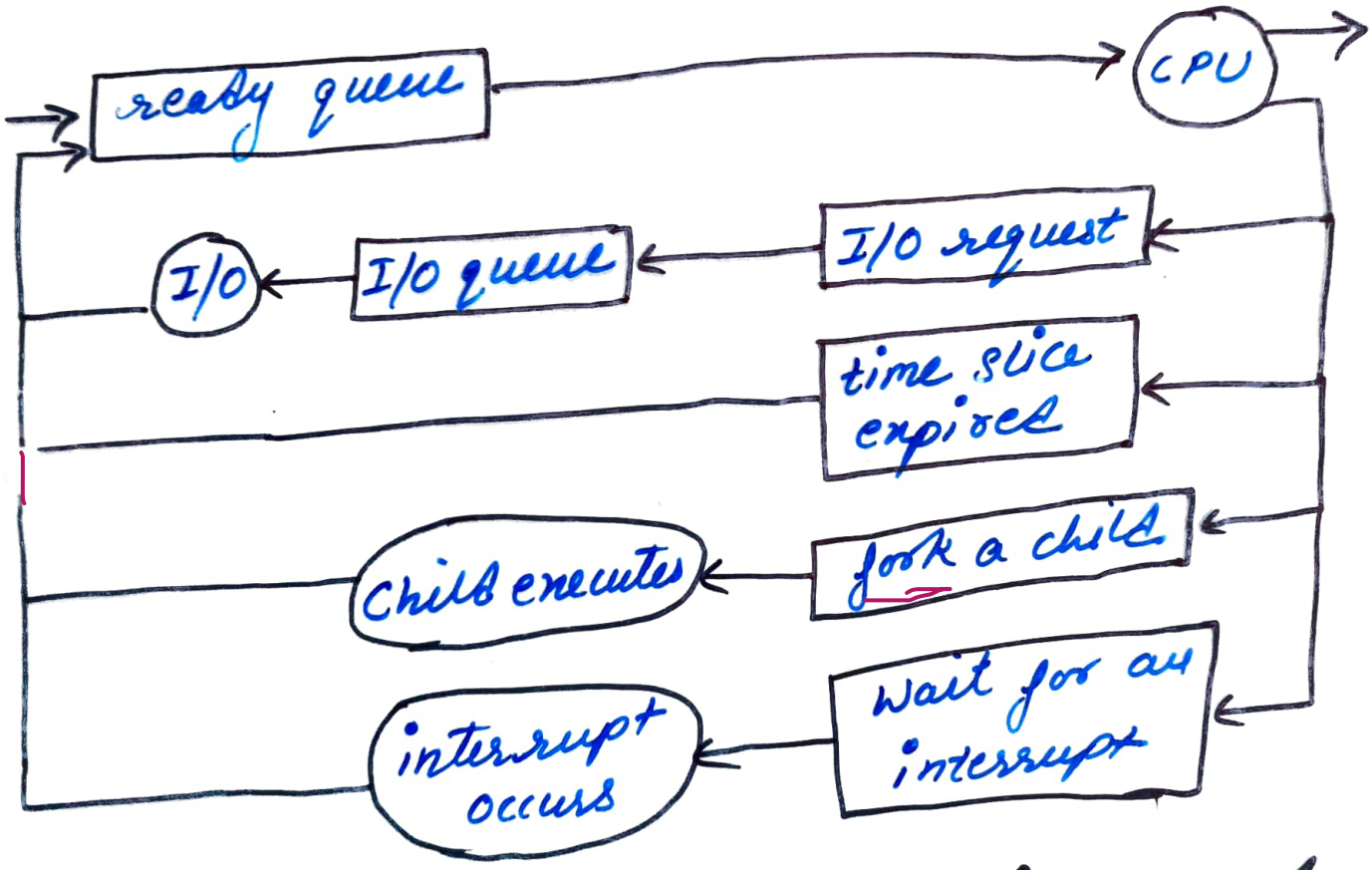
I/O status information :- It includes the list of I/O devices allocated to the process, a list of open files and soon.

x — x — x — x — x — x — x — x — x

PROCESS SCHEDULING :- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. To meet these objectives, the process scheduler selects an available process for program execution on the CPU.

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 & waiting to execute are kept on a list called the ready queue.
- The list of processes waiting for a particular I/O device is called a device queue
- A common representation for a discussion of process scheduling is a queueing diagram. Rectangular box represents a queue. The circles represent the resources that serve the queues and the arrows indicate the flow of processes in the system.



A new process is initially put in the ready queue. It waits there until it is selected for execution, or is dispatched. Once the process is allocated the CPU and is executing

one of several events could occur:

- The process could issue an I/O request and then be placed in an I/O queue
- The process could create a new subprocess and wait for the subprocess's termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt and be put back in the ready queue.

CONTEXT SWITCH :- Switching the CPU to another process requires saving the state of an old process and loading the saved state of the new process. This task is known as a context switch.

- The context of a process is represented in the PCB of the process.
- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context switch time is pure overhead because the system does no useful work while switching.
- Context switch times are highly dependent on hardware support.

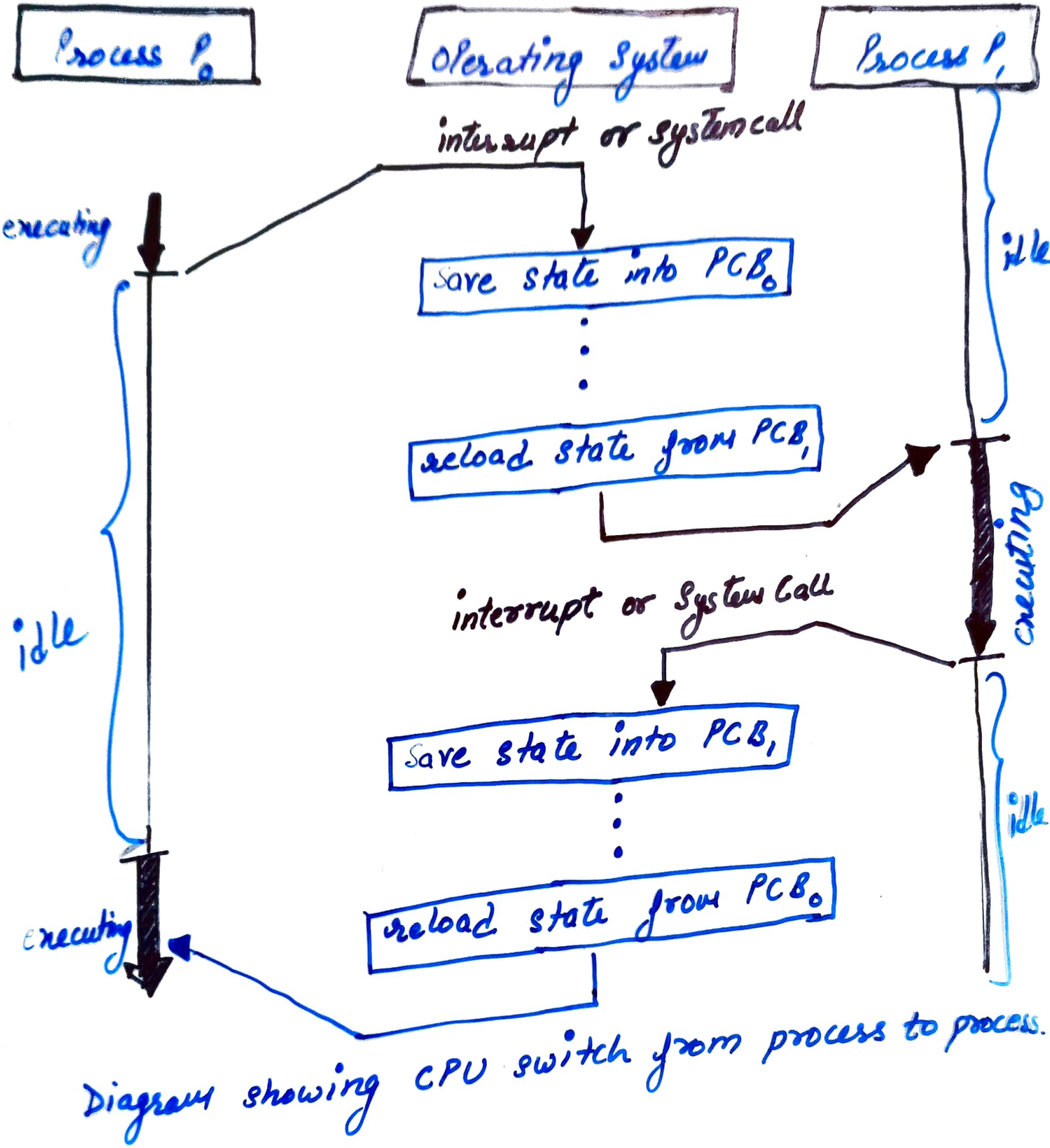


Diagram showing CPU switch from process to process.

Operations on Processes :-

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Operating systems must provide a mechanism for process creation and termination.

Process Creation :-

* A process may create several new processes, via a create-process system call, during the course of execution.
* The creating process is called a Parent Process.
* The new processes are called the 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 to accomplish its task.
* When a process creates a subprocess, that subprocess may be able to obtain its resources directly from the OS or of the parent process.
* The Parent may have to partition its resources among its children, or it may be able to share some resources among several of its children.
* In addition to the various physical and logical resources that a process obtains when it is created, initialization data may be passed along by the parent process to the child process.

When a process creates a new process, two possibilities exist in Terms of execution:

1) The parent continues to execute concurrently with its children.
2) The parent waits until some or all of its children have terminated.

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

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

In this manner, the two processes are able to communicate and then go their separate ways. The parent can then create more children or if it has nothing else to do while the child runs, it can issue a wait() system call to move itself off the ready queue until the termination of the child.

PROCESS TERMINATION :-

* A process terminates when it finishes executing its final statement and asks the operating System to delete it by using the exit() system call.
* At that point, the process may return a status value to its parent process.

- All the resources of the process - including physical and virtual memory, open files, and I/O buffers - are deallocated by the operating system
- A process can cause the termination of another process via an appropriate system call.
- Such a system call can be invoked by only the parent of the process that is to be terminated. Otherwise, users could arbitrarily kill each other's job.

A parent may terminate the execution of one its children for a variety of reasons, such as these:

- The child has exceeded its usage of some of the resources that it has been allocated.
- The task assigned to the child is no longer required.
- The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

Many systems, do not allow a child to exist if its parent has terminated. In such systems, if a process terminates then all its children must also be terminated. This phenomenon referred to as cascading termination.