DEPARTMENT OF AIML
23CST202- OPERATING SYSTEMS
II YEAR IV SEM AIML-B
UNIT 3-MEMORY MANAGEMENT
TOPIC –PROCESS CREATION

PROCESS CREATION

A process is an instance of a program running, and its lifecycle includes various stages such as creation, execution, and deletion.

- The operating system handles process creation by allocating necessary resources and assigning each process a unique identifier.
- Process deletion involves releasing resources once a process completes its execution.
- Processes are often organized in a hierarchy, where parent processes create child processes, forming a tree-like structure.

Process Creation

As discussed above, processes in most of the operating systems (both Windows and Linux) form hierarchy. So a new process is always created by a parent process. The process that creates the new one is called the parent process, and the newly created process is called the child process. A process can create multiple new processes while it's running by using system calls to create them.

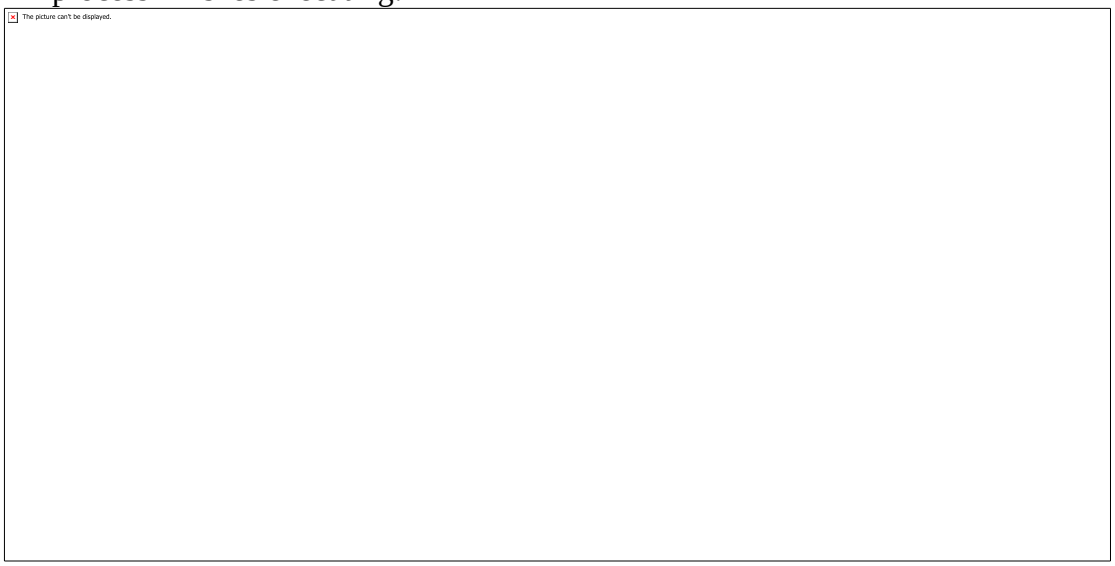
1. When a new process is created, the operating system assigns a unique Process Identifier (PID) to it and inserts a new entry in the primary process table.
2. Then required memory space for all the elements of the process such as program, data, and stack is allocated including space for its Process Control Block (PCB).
3. Next, the various values in PCB are initialized such as,

1. The process identification part is filled with PID assigned to it in step (1) and also its parent's PID.
2. The processor register values are mostly filled with zeroes, except for the stack pointer and program counter. The stack pointer is filled with the address of the stack-allocated to it in step (2) and the program counter is filled with the address of its program entry point.
3. The process state information would be set to 'New'.
4. Priority would be lowest by default, but the user can specify any priority during creation. Then the operating system will link this process to the scheduling queue and the process state would be changed from 'New' to 'Ready'. Now the process is competing for the CPU.
5. Additionally, the operating system will create some other data structures such as log files or accounting files to keep track of process activity.

Understanding System Calls for Process Creation in UNIX Operating System:

Process creation is achieved through the fork() system call. The new process that gets created is called the child process, and the one that started it (the one that was already running) is called the parent process. After the fork() call, you end up with two processes: the parent and the child, both running independently.

- The fork() system call creates a copy of the current process, including all its resources, but with just one thread.
- The exec() system call replaces the current process's memory with the code and data from a specified executable file. It doesn't return; instead, it "transfers" the process to the new program.
- The waitpid() function makes the parent process wait until a specific child process finishes executing.



Process creation in Unix

```
Example:
int pid = fork();
if (pid == 0)
{
/* Child process */
exec("foo");
}
else
{
/* Parent process */
waitpid(pid, &status, options);
}
```

Understanding System Calls for Process Creation in Windows Operating System:

In Windows, the system call used for process creation is CreateProcess(). This function is responsible for creating a new process, initializing its memory, and loading the specified program into the process's address space.

- CreateProcess() in Windows combines the functionality of both UNIX's fork() and exec(). It creates a new process with its own memory space rather than duplicating the parent process like fork() does. It also allows specifying which program to run, similar to how exec() works in UNIX.
- When you use CreateProcess(), you need to provide some extra details to handle any changes between the parent and child processes. These details control things like the process's environment, security settings, and how the child process works with the parent or other processes. It gives you more control and flexibility compared to the UNIX system.

Process Deletion

Processes terminate themselves when they finish executing their last statement, after which the operating system uses the exit() system call to delete their context. Then all the resources held by that process like physical and virtual memory, 10 buffers, open files, etc., are taken back by the operating system. A process P can be terminated either by the operating system or by the parent process of P.

A parent may terminate a process due to one of the following reasons:

1. When task given to the child is not required now.
2. When the child has taken more resources than its limit.
3. The parent of the process is exiting, as a result, all its children are deleted. This is called cascaded termination.

A process can be terminated/deleted in many ways. Some of the ways are:

1. **Normal termination:** The process completes its task and calls an exit() system call. The operating system cleans up the resources used by the process and removes it from the process table.
2. **Abnormal termination/Error exit:** A process may terminate abnormally if it encounters an error or needs to stop immediately. This can happen through the abort() system call.
3. **Termination by parent process:** A parent process may terminate a child process when the child finishes its task. This is done by the using kill() system call.
4. **Termination by signal:** The parent process can also send specific signals like SIGSTOP to pause the child or SIGKILL to immediately terminate it