Under : Process Creation:

Add the below content after the existing content:

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 program loaded into it.

To illustrate these different implementations, let us consider the **UNIX** operating system. In UNIX, each process is identified by its **process identifier**, which is a unique integer. A new process is created by the **fork** system call. The new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process. Both processes (the parent and the 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 the (non zero) process identifier of the child is returned to the parent.**

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

**Below is a C program to illustrate forking a sparate process using UNIX (made using Ubuntu)**

#include <stdio.h>

void main(int argc, char \*argv[])

{

int pid;

/\* fork another process \*/

pid=fork();

if(pid<0)

{

//error occurred

fprintf(stderr, "Fork Failed");

exit(-1);

}

else if (pid == 0)

{

//child process

execlp("/bin/ls","ls",NULL);

}

else

{

//parent process

//parent will wait for the child to complete

wait(NULL);

printf("Child complete");

exit(0);

}

}

**Gate Numerical Tip**: if fork is called for **n times**, the number of child process or **new process** created are : **2^n – 1**.