**Process creation and termination:**

fork() is a system call used to create a new process. The new process is called a child process and the original process is called the parent process.

The child process by default is a duplicate of the parent process. By duplicate we mean that the child process has the same code as the parent process but the memory space of both the processes is separate.

A process during its execution can create many new processes via system call(depending upon the OS). The creating process is called the parent while the created process is called the child. Each child process may in turn create new child process.

Every process in the system is identified with a process identifier(PID) which is unique for

 each process.

**How the child process gets its resources?**

There are two ways to allocate resources to a child process:

1. The operating system allocates the resources to the child from the set of resources available in the system. But this can cause problems if a process creates to many child process.

2. The child process gets the resources from the set of resources available with its parent.

Execution Possibilities for parent process

1. The parent process *Continues* to execute concurrently with its children
2. The parent process *Waits*until some or all the children terminate. This refers to situations where the parent needs some result or value from the child. For example, suppose the parent is to execute c=a+b; while the child has to execute a=e+f; Now, the parent can not proceed until the value of ‘a’ is calculated by the child.

**Address Space of child process**

* Is a duplicate of the parent process – which means that the child process will also have the same code as the parent process. Thus, the code in the parent should clearly define the task for itself and the child process. (Refer the code below)
* Has a new program loaded into it

let a prgram has 5 lines of code/statements

and if all these will be copied from parent to child , then no use for child process

as it is same

but instead we have to give separate as

let p1 is parent and P2 is child

p1( 2 statements/task)

p2 (3 statements/task)

then it will be helpful

fork creates the processes

value returned by the fork let it be p

**three cases for value of p as:**

1. if (p<0)-----error , no process is created.

2. if (p==0)--- it means child process is created.

3.if (p>0)--- it means parent process is created.

-----------------------------------------------

man 2 fork

# 2 is for?

Ans:- 2 is used to see the information related to system calls

 --------------------------------------------

Fork

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

pid\_t pid;

fork();

printf("my id is %d\n",pid); //statement will be printed

//1 for parent and 1 for child -- prints for 2^n times

}

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

pid\_t pid;

pid=fork();//will print my id is 6202my id is 0 – 0 is for child process and can be printed before parent like my id is 0my id is 6202

printf("my id is %d\n",pid);

}

//fork

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

pid\_t p;

printf("See before fork\n");

p=fork();

if(p==0);//for child process

{

printf("I am child having id %d\n",getpid());

printf("my parent id is %d\n",getppid());

}

else

{

printf("my child id is %d\n",p);//for parent process

}

}

Fork part 2

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

pid\_t p;

printf("See before fork\n");

p=fork();

if(p==0)//for child process

{

printf("I am child having id %d\n",getpid());

printf("my parent id is %d\n",getppid());

}

else

{

printf("my child id is %d\n",p);//for parent process

printf("I am parent having id %d\n",getpid());

}

printf("Common\n");

}