Aim:

To create a new child process using fork system call.

Fork ( )

- The fork system call is used to create a new process called child process.

  * The return value is 0 for child process.

  * The return value is negative if process creation is unsuccessful.

  * For the parent process, return value is positive.

- The child process is an exact copy of the parent process.

- Both the child and parent continue to execute the instructions following fork call.

- The child can start execution before the parent or vice-versa.

Algorithm:

- Declare a variable x to be shared by both child and parent.

- Create a child process using fork system call.

- If return value is -1 then
  print "process creation successfull"
  Terminate using exit system call
- If return value is 0 then
  print "child process".
  print process id of the child using getpid system call.
  print value of x
  print process id of the parent using getppid system call.
- otherwise
  print "parent process"
  print process id of the parent using getpid system call.
  print value of x.
  print process id of the shell using getppid system call.
- stop

FORK() SYSTEM CALL

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
        pid_t pid;
        int x=5;
        pid= fork();
        x++;
        if(pid<0)
        {
                printf("Process creation error");
                exit(-1);
        }
        else if(pid==0)
        {
                printf("Child process:");
                printf("\n Process id is %d",getpid());
                printf("\n value of x is %d",x);
                printf("\nProcess id of parent is %d\n",getppid());
        }
```

```
else
{
        printf("\n parent process:");
        printf("\n process id is %d",getpid());
        printf("\n value of x is %d ",x);
        printf("\n Process id of shell is %d\n",getppid());
}
}
```

Result

Thus a child process is created with copy of its parent's address space.