# **Experiment No: 4**

## AIM: Concurrent programming; use of threads and processes, system calls (fork and v-fork).

### **Objective:** To study and implement use of threads, processes and fork v-fork system call.

#### Theory:

**Fork()**

System call **fork()** is used to create processes. It takes no arguments and returns a process ID. The purpose of **fork()** is to create a ***new*** process, which becomes the *child* process of the caller. After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()*** system call. The child and parent processes are located in separate physical address spaces. As a result, the modification in the parent process doesn't appear in the child process.Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of **fork()**:

* If **fork()** returns a negative value, the creation of a child process was unsuccessful.
* **fork()** returns a zero to the newly created child process.
* **fork()** returns a positive value, the ***process ID*** of the child process, to the parent. The returned process ID is of type **pid\_t** defined in **sys/types.h**. Normally, the process ID is an integer. Moreover, a process can use function **getpid()** to retrieve the process ID assigned to this process.

**Source Code:**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

// Write C code here

fork();

printf("Process id=%d\n", getpid());

return 0;

}

Output:

**Vfork()**

Similar to fork(), vfork() creates a new subprocess for the calling process. However, vfork() is specifically designed for subprocesses to execute exec() programs immediately.

vfork() creates a child process just like fork(), but it does not copy the address space of the parent process to the child process completely, because the child process will immediately call exec (or exit), so the address space will not be accessed. However, before a child process calls exec or exit, it runs in the space of the parent process. Another difference between vfork() and fork() is that vfork() ensures that the child process runs first, and that the parent process may not be scheduled until it calls exec or exit. (if the child process depends on further actions of the parent process before these two functions are called, a deadlock can result.)

Source Code:

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

//main function begins

int main(){

pid\_t p= vfork(); //calling of fork system call

if(p==-1)

printf("Error occured while calling fork()");

else if(p==0)

printf("This is the child process with ID=%d\n", getpid());

else

printf("This is the parent processwith ID=%d\n", getpid());

return 0;

}

**Output:**