CAB

Lab- 2

Subject: Operating System (CSC-259)

Create Process, Thread, Implement IPC Techniques & Simulate Process Scheduling Algorithms

Create Process

- Each processes in a Linux system is identified by unique process ID, reffered to as **pid .**
- Process Ids are 1-32768 numbers that are assigned sequentially by Linux as new process are created.
- Every process also has a parent process [except **init (Init** is the root/parent of all **processes** executing on **Linux**] .Thus , we can think the processes in the linux are arranged in a tree, within the init process at its root.
- The parent process ID , or **ppid** is simply the process ID of the process's parent.
- Most of the process manipulation functions are declared in the header file **<unistd.h>**
- A program can obtain the process ID of the process its running with the **getpid()** system call & it can obtain the obtain the process ID of its parent process with the **getppid()** system call.
- Linux provides one function **fork ()** that makes the child process that is an exact copy of its parent process. Linux also provides another set of functions , the exec family that replaces current process image with anew process image.

Program 1: Creation of single process & display their id

```
#include<stdio.h>
#include<unistd.h> /*contains fork prototype*/
int main(void)
{       printf("Hello World!\n");
        fork();
        printf("I am after forking\n");
        printf("\t I am process % d \n", getpid());
}
```

When this program is executed , it first prints Hello World!. When the fork is executed, an identical process called child is created .Then both parent & child process begain execution at the next statement.

/*Output

Hello World!

I am after forking

I am process 3245

I am after forking

I am process 3246 */

Program 2: Program getting id of process.

```
#include<stdio.h>
#include<unistd.h> /*contains fork prototype*/
int main(void)
{
        int pid;
        printf("Hello World\n");
        printf("I am the process & pid is : %d . \n",getpid());
        printf("Here i am before use of forking\n");
        pid=fork();
        printf("Here I am just after forking \n");
        if(pid==0)
                printf("I am the child process and pid is : %d.\n",getpid());
        else
                printf("I am parent process and pid is : %d .\n",getpid());
}
```

/*Output

Hello World

I am the process & pid is : 3069 .

Here i am before use of forking

Here I am just after forking

I am parent process and pid is : 3069 .

Here I am just after forking

I am the child process and pid is : 3070.*/

Program 3: Use the multiple fork

```
include<stdio.h>
#include<unistd.h> /*contains fork prototype*/
void main(void)
{
        printf("Here I am just before first forking statement \n");
        fork();
        printf("Here I am after first forking statement\n");
        fork();
        printf("Here I am after second forking statement\n");
        printf("\tHello World from process %d !\n",getpid());
}

/*output
```

Here I am just before first forking statement

Here I am after first forking statement

Here I am after second forking statement

Hello World from process 4520 !

Here I am after second forking statement

Hello World from process 4522 !

Here I am after first forking statement

Here I am after second forking statement

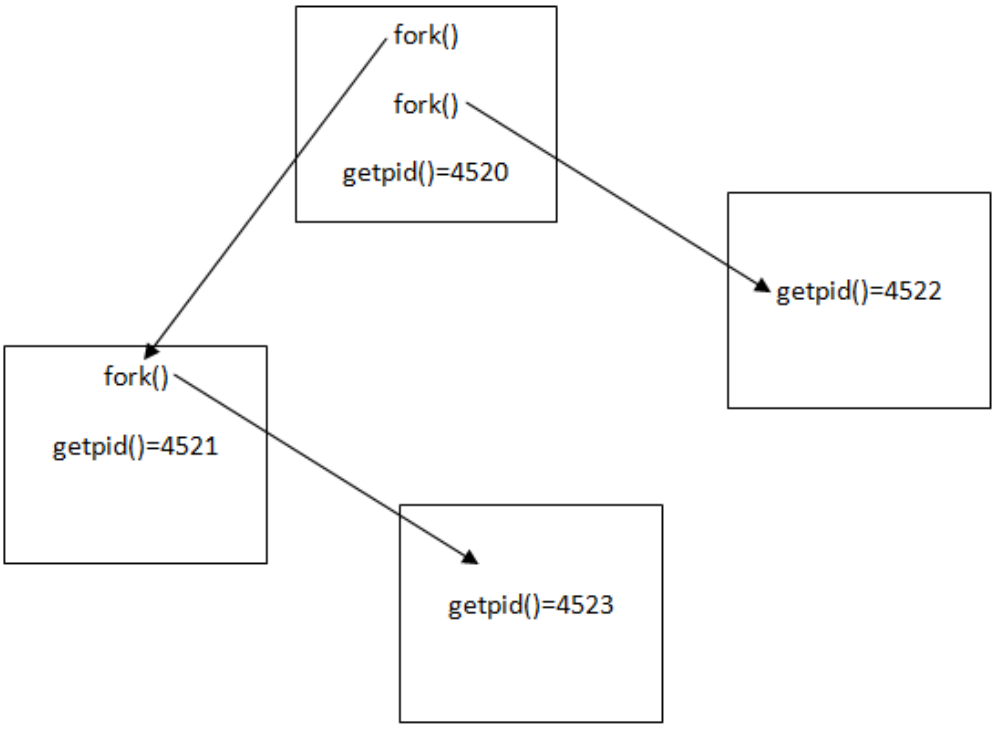
Hello World from process 4521 !

Here I am after second forking statement

Hello World from process 4523 !

*/

Output Analysis how multiple fork call execution



By: Santosh Sharma
College of Applied Business
Department of Computer Science & IT