Operating system
Lab 2
Process concepts

The fork

The fork system call creates a new process. When a program calls fork() there will be two copies of the programs running simultaneously. Each copy can do what it desires independent of the other.

The fork() function returns the child's process id to the parent. It returns 0 to the child and returns a negative number in case of failure.

Observe the output of the following programs and comment on the result

Program 1:

```
main()
{
        if(!fork())
        {
                printf("hello!!!!! i'm from child\n");
        }
        else
        {
                printf("hi !!!i'm from parent");
        }
}
```

Task 1: Make modifications in the above program. Print the process id of each process and its parent. Use *getpid()* and *getppid()*. Observe the program segment below. How many processes, do you think, are created? Reformat the code to clarify the process family structure.

```
main()
{
        fork();
         fork();
        fork();
        printf("process details\n");
}
```

Program 2:

```
main()
{
    int i,d;
    char c;

    if(!fork())
    {
        for(c='a';c<='z';c++)
        {
            printf("%c\t"c)l
            fflush(stdout);
            for(d=0;d<DEL1;d++);
        }
        exit(0)
        }

else{
        for(i=0;i<=10;i++)
        {
        printf("%i\n",i);
        fflush(stdout);
        for(d=0;d<=DEL2;d++);
        }
exit(0)
}
}
```

Task 2: change the value of the delay parameters DEL1 and DEL 2 and comment on the result.

Program 3:

```
main()
{
        int pid;
        int fork();

if(pid==0)
{
        printf("i'm the child, my process ID is %d\n",getpid());
        printf("I'm the child and my parent's ID is %d\n",getppid());

        sleep(20);

        printf("i'm the child, my process ID is %d\n",getpid());
        printf("I'm the child and my parent's ID is %d\n",getppid());
}

        else
{
        //anchor

        printf("I'm the parent, my process ID is %d",getpid());
        printf("the parent's process ID is %d", getppid());
}
}
```

Task 3: The result must have betrayed your expectations. Can you reason it? Enter a *sleep()* at *anchor*. Change the value of the argument to sleep at this point and observe the result.

In the above case the change in the child's parent ID is because the original parent expired and the child had to be adopted.

Program 4:

```
main()
{
        int i=0,j=0,pid;

        pid=fork();
        if(pid==0)
        {
                for(i=0;i<500;i++)
                printf("%d\t",i);
        }
        else
        {
        if(pid>0)
        {
                //anchor
                for(j=0;j<500;j++)
                printf("%d",j);
        }
        }
}
```

Task4: Comment on the above program. Now insert *wait(0)* at *anchor* and observe the result.

What actually happens is the parent waits for the child to complete. This feature is available in UNIX. A parent can wait until its child has completed. The reverse however is not permitted.

Surf on the net and comment on the following

- **Zombie process**
- **Orphan process**