Indian Institute of Information Technology Vadodara CS266:

Operating Systems Lab

Lab 5

Roll No. 201951105 **Name: Nishant Andoriya**

1. Write a C program to create the process using fork() and display the parent and child process ID using getpid() and getppid() system calls, respectively.

Ans:-

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
 int c=fork();
 if(c==0)
 {
 printf("PARENT PROCESS IS: ");
 printf("%d", getpid());
 printf("\n");
 printf("CHILD PROCESS IS: ");
 printf("%d", getppid());
 }
 return 0;
}
```

```
PARENT PROCESS IS: 30
CHILD PROCESS IS: 1

...Program finished with exit code 0
Press ENTER to exit console.
```

2. Write a C program to demonstrate the exec() system call for (1) display the content of directory (command: ls)

Ans:-

```c
#include<stdio.h>
#include<unistd.h>
#include <limits.h>
int main()
{
 char current_dir[PATH_MAX];
 if (getcwd(current_dir, sizeof(current_dir)) != NULL)
 {
 printf("Current working dir: %s\n", current_dir);
 }
 else
 {
 printf("error running code");
 return 1;
 }
 return 0;
}
```

```
Current working dir: /home


...Program finished with exit code 0
Press ENTER to exit console.
```

(2) display the process tree (command: pstree).

Ans:-

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
 char *args[]={"./EXEC",NULL};
 execv(args[0],args);
 return 0;
}
```

execv() will then start executing lab51 which will print the current directory .

3. Write a C program to demonstrate wait and sleep system calls return the pid of the child that terminated.

Ans:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
void fillprocess()
{

 int process_id[5];
 for (int i=0; i<5; i++)
 {
 if ((process_id[i] = fork()) == 0)
 {
 sleep(1);
 exit(i);
 }
 }
 for (int i=0; i<5; i++)
 {
 process_id[i] = wait(NULL);
 printf("Process terminated is : %d\n", process_id[i]);
 }
}
int main()
{
 fillprocess();
 return 0;
}
```

Output 1:-

```
Process terminated is : 428
Process terminated is : 429
Process terminated is : 430
Process terminated is : 432
Process terminated is : 431


...Program finished with exit code 0
Press ENTER to exit console.
```

Output 2:-

```
Process terminated is : 583
Process terminated is : 586
Process terminated is : 587
Process terminated is : 584
Process terminated is : 585


...Program finished with exit code 0
Press ENTER to exit console.
```

4. Write a C program to create the multiple processes using fork() and display the process IDs and their parent process IDs in hierarchy.

Ans;-

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
 int temp1=fork();
 int temp2=fork();
 int temp3=fork();
 if (temp1 > 0 && temp2 > 0 && temp3 > 0)
```

```
 {
 printf(" PARENT");
 printf(" process id is %d \n", getpid());
 printf(" parent id is %d \n", getppid());
 }
 else if (temp1 == 0 && temp2 > 0 && temp3 > 0)
 {
 printf("FIRST CHILD");
 printf(" process id is %d \n", getpid());
 printf(" parent id is %d \n", getppid());
 }
 else if (temp1 > 0 && temp2 == 0 && temp3 > 0)
 {
 printf("SECOND CHILD");
 printf(" process id is %d \n", getpid());
 printf(" parent id is %d \n", getppid());
 }
 else if (temp1 > 0 && temp2 > 0 && temp3 == 0)
 {
 printf("THIRD CHILD");
 printf(" process id is %d \n", getpid());
 printf(" parent id is %d \n", getppid());
 }
 else
 {
 printf("FOURTH CHILD");
 printf(" process id is %d \n", getpid());
 printf(" parent id is %d \n", getppid());
 }

 return 0;
}
```

```
 PARENT process id is 803
 parent id is 798
FIRST CHILD process id is 804
 parent id is 1
SECOND CHILD process id is 805
 parent id is 1
FOURTH CHILD process id is 809
 parent id is 805
FOURTH CHILD process id is 808
 parent id is 1


...Program finished with exit code 0
Press ENTER to exit console.
```

5. Write a c program to interrupt and terminate the current process using signal handlers. (use SIGINT i.e., Ctrl-C or 2 from keyboard to interrupt and Ctrl-\ or 9 from keyborad to kill the process)

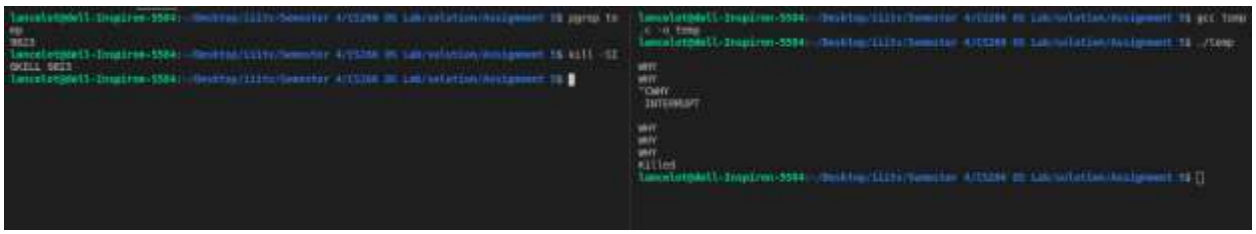
Ans:-

```c
#include<stdio.h>
#include <signal.h>
#include<sys/types.h>
#include<unistd.h>
void handle_sigcommand(int num)
{
 if(num==SIGINT)
 {
 printf("\n INTERRUPT \n");
 sleep(10);
 }
 else if(num==SIGKILL)
 {
 printf("\n KILLED \n");
 int temp=getpid();
 kill(temp,SIGKILL);

 }
}
int main()
{
 signal(SIGINT,handle_sigcommand);
 signal(SIGKILL,handle_sigcommand);
```

```
 while(1)
 {
 printf("WHY");
 sleep(2);
 }
 return 0;
}
```

Output:-



In the output we can clearly see that when ctr+c action is performed(denoted by “^C” in the output) the process is interrupted and when ctr+\ action is performed (denoted by ^\ in the output) the process is terminated.