**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");

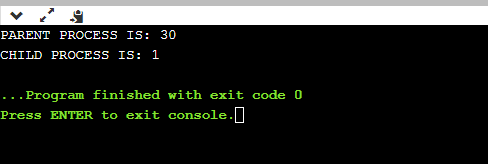
 printf("CHILD PROCESS IS: ");

 printf("%d", getppid());

 }

 return 0;

}



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

Ans:-

#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

 {

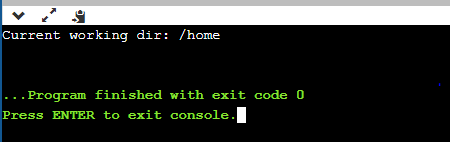
 printf("error running code");

 return 1;

 }

 return 0;

}



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

Ans:-

#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 .**

1. 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()

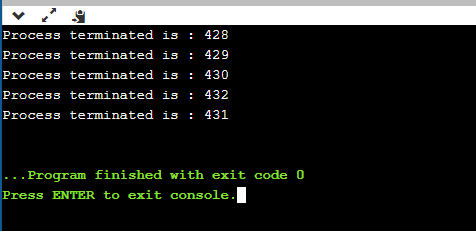
{

 fillprocess();

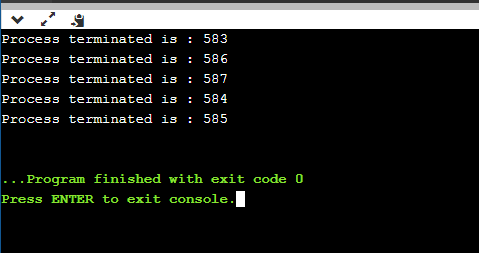
 return 0;

}

Output 1:-



Output 2:-



1. 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");

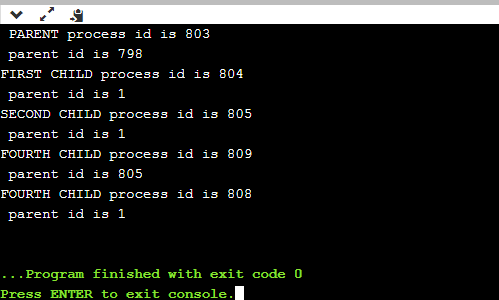
 printf(" process id is %d \n", getpid());

 printf(" parent id is %d \n", getppid());

 }

 return 0;

}



1. 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:-

#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");

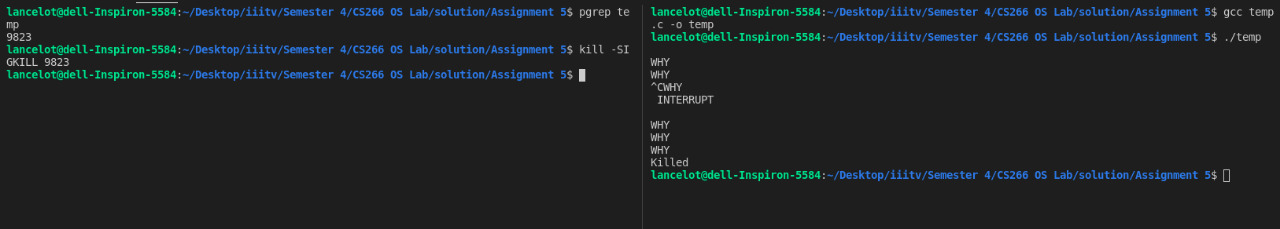
 sleep(2);

 }

 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.