**\*\*\*\*\*\*execv**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(){

pid\_t cpid = fork();

char \*args[]={"./fact","./p",NULL};

if(cpid==-1){

printf("FOrk failed");

exit(EXIT\_FAILURE);

}

else if(cpid==0){

printf("Process ID of child : %d\n",getpid());

execv(args[0],args);

}

else{

printf("Process ID of parent : %d\n",getpid());

execv(args[1],args);

}

printf("ENd of fork system call");

}

**\*\*\*factorial.c**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(){

int n,i,f=1;

printf("Enter a number:");

scanf("%d",&n);

printf("Process ID of child process---executing factorial:%d\n\n",getpid());

if(n==0 || n==1){

printf("Factorial of %d is 1",n);

}

else if(n<0){

printf("Factorial doesnt exists for a negative number");

}

else{

for(i=1;i<=n;i++){

f=f\*i;

}

printf("factorial of %d is %d",n,f);

}

}

**\*\*prime.c**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(){

int n,i;

printf("Enter a number:");

scanf("%d",&n);

printf("Process ID of parent process -- executing prime number program - %d\n\n",getpid());

int flag=0;

for (i=2;i<=n/2;i++){

if (n%i==0){

flag++;

break;

}

}

if(flag==0){

printf("%d is prime number\n\n",n);

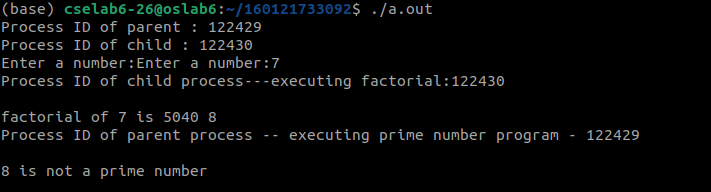
}

else{

printf("%d is not a prime number\n\n",n);

}

}



\*\*\*\*\***execvp**

**\*\*\*\*\*execve**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(){

printf("execve----\n");

pid\_t cpid = fork();

char \*args[]={"fact",NULL};

char \*args1[]={"p",NULL};

char \*envp[] = {"VAR=value", NULL};

if(cpid==-1){

printf("FOrk failed");

exit(EXIT\_FAILURE);

}

else if(cpid==0){

printf("Process ID of child : %d\n",getpid());

execve(args[0], args, envp);

}

else{

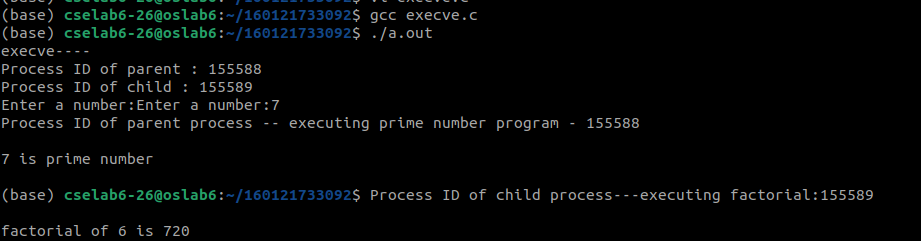
printf("Process ID of parent : %d\n",getpid());

execve(args1[0], args1, envp);

}

printf("End of fork system call");

}



**\*\*\*\*\*execl**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(){

printf("execvp----\n");

pid\_t cpid = fork();

char \*file1 = "./fact";

char \*file2 = "./p";

char \*arg1 = "Hello world!";

if(cpid==-1){

printf("FOrk failed");

}

else if(cpid==0){

printf("Process ID of child : %d\n",getpid());

execl(file1, arg1, NULL);

}

else{

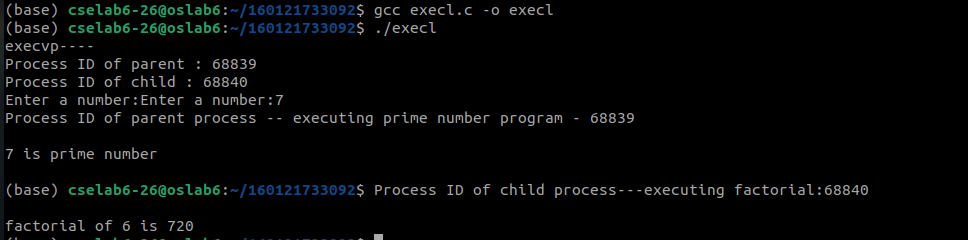
printf("Process ID of parent : %d\n",getpid());

execl(file2, arg1,NULL);

}

printf("ENd of fork system call");

}

****

**\*\*\*\*\*execlp**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(){

printf("execlp----\n");

pid\_t cpid = fork();

char \*file1 = "fact";

char \*file2 = "p";

char \*arg1 = "Hello world!";

if(cpid==-1){

printf("FOrk failed");

}

else if(cpid==0){

printf("Process ID of child : %d\n",getpid());

execl(file1, arg1, NULL);

}

else{

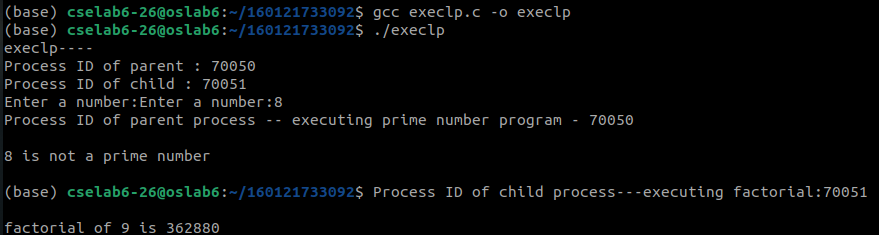
printf("Process ID of parent : %d\n",getpid());

execl(file2, arg1,NULL);

}

printf("ENd of fork system call");

}

****

**\*\*\*\*\*\*execlv**

#include <unistd.h>

#include<stdlib.h>

#include<stdio.h>

int main(void) {

char \*file = "/usr/bin/bash";

char \*arg1 = "-c";

char \*arg2 = "echo $ENV1 $ENV2!";

char \*const env[] = {"ENV1=Hello", "ENV2=World", NULL};

printf("execle----\n");

pid\_t cpid = fork();

char \*file1 = "./fact";

char \*file2 = "./p";

if(cpid==-1){

printf("FOrk failed");

}

else if(cpid==0){

printf("Process ID of child : %d\n",getpid());

execle(file1,file, arg1, NULL,env);

}

else{

printf("Process ID of parent : %d\n",getpid());

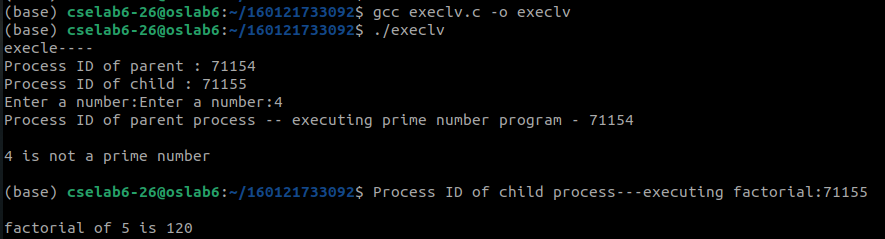
execle(file2,file, arg1,NULL,env);

}

printf("ENd of fork system call");

return 0;

}

****

**Threads**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

void \*fun1();

void \*fun2();

int shared = 1;

int main(){

pthread\_t thread1, thread2;

printf("Main thread process ID %d\n", getpid());

pthread\_create(&thread1, NULL, fun1, NULL);

pthread\_create(&thread2, NULL, fun2, NULL);

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

printf("In main thread\n");

}

void \*fun1(){

printf("Inside thread 1 with process ID: %d\n", getpid());

for(int i=0; i<5;i++){

printf("Thread one i = %d, Shared: %d\n", i, shared++);

sleep(1);

}

}

void \*fun2(){

printf("Inside thread 2 with process ID: %d\n", getpid());

for(int j=0; j<5;j++){

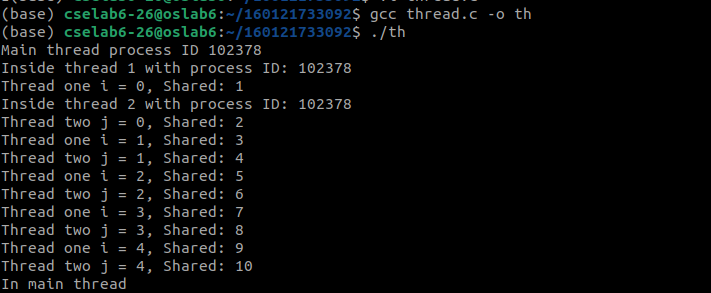
printf("Thread two j = %d, Shared: %d\n", j, shared++);

sleep(1);

}

}

~



**\*\*\*signalex.c**

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

void handle\_signal(int sig)// Handler

{

printf("\n Signal Caught %d \n",sig);

}

int main()

{

signal(SIGINT,handle\_signal);// Registration process

int i=0;

while(i<30)

{

printf("Hello World \n");

sleep(1);

i++;

}

return 0;

}



**\*\*\*sigchildex.c**

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<signal.h>

void handle\_signal(int sig)// Handler

{

printf("\n Inside Handler: child is terminated %d \n",sig);

}

int main()

{

signal(SIGCHLD,handle\_signal);// Registration process

int i=fork(), w;

if(i==0)

{

printf("Child process");

}

else

{

printf("Inside parent process \n");

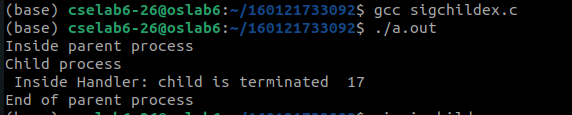
wait(&w);

printf("End of parent process \n");

}

return 0;

}



\*\*\*Paging

#include<stdio.h>

#include<stdlib.h>

#define MAX 50

int main()

{

int page[MAX],i,n,f,ps,off,pno;

int choice=0;

printf("\nEnter the no of pages in memory: ");

scanf("%d",&n);

printf("Enter page size: ");

scanf("%d",&ps);

printf("Enter no of frames: ");

scanf("%d",&f);

for(i=0;i<n;i++)

page[i]=-1;

printf("\nEnter the page table\n");

printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");

printf("\npageno\tframeno\n-------\t-------");

for(i=0;i<n;i++)

{

printf("\n%d\t",i);

scanf("%d",&page[i]);

if(page[i] >= f){

printf("Frame no. is greater than no. of frames\nexiting...");

exit(0);

}

}

do

{

printf("\nEnter the logical address(i.e,page no & offset):");

scanf("%d%d",&pno,&off);

if(page[pno]==-1)

printf("\nThe required page is not available in any of frames");

else

printf("\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);

printf("\nDo you want to continue(1/0)?:");

scanf("%d",&choice);

}while(choice==1);

return 1;

}

