**Practical – 7**

**Part - A**

**AIM: The demonstration of fork () system call.**

1. Ex1

Code:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(void)

{

pid\_t pid=fork();

if(pid==0)

{

printf("Child => PPID: %d PID: %d\n",getppid(),getpid());

exit(EXIT\_SUCCESS);

}

else if(pid>0)

{

printf("Parent => PID: %d\n",getpid());

printf("Waiting for child process to finish.\n");

wait(NULL);

printf("Child process finished.\n");

}

else

{

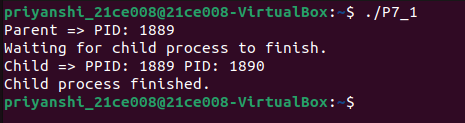
printf("Unable to create child process.\n");

}

return EXIT\_SUCCESS;

}

Output:



1. Ex2

Code:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

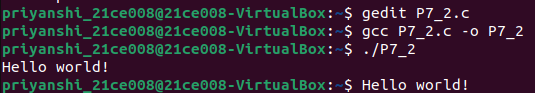
fork();

printf("Hello world!\n");

return 0;

}

Output:



1. Ex3

Code:

#include <stdio.h>

#include <sys/types.h>

int main()

{

fork();

fork();

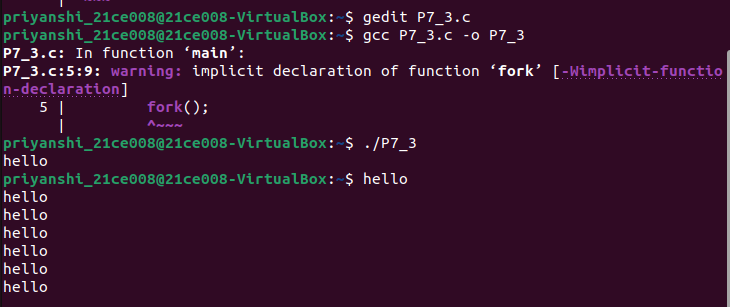
fork();

printf("hello\n");

return 0;

}

Output:



1. Ex4

Code:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void forkexample()

{

// child process because return value zero

if (fork() == 0)

printf("Child!\n");

// parent process because return value non-zero.

else

printf("Parent!\n");

}

int main()

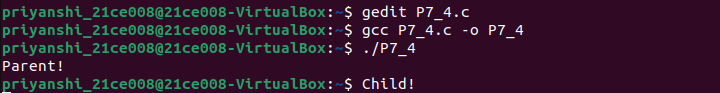
{

forkexample();

return 0;

}

Output:



**Part - B**

**AIM: Demonstration of execve () and wait () system calls along with zombie and orphan states.**

1. Zombie state (process)

Code:

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int main()

{

// Fork returns process id

// in parent process

pid\_t child\_pid = fork();

// Parent process

if (child\_pid > 0)

{

sleep(10);

printf("10 seconds waited..");

}

else

exit(0);

return 0;

}

Output:



1. Orphan state (process)

Code:

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Create a child process

int pid = fork();

if (pid > 0)

printf("in parent process \n");

// Note that pid is 0 in child process

// and negative if fork() fails

else if (pid == 0)

{

sleep(5);

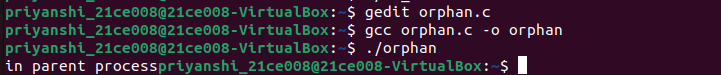
printf("in child process \n");

}

return 0;

}

Output:



1. Merge-Quick sort (use of execve() and wait())

Code:

/\*Implement the C program in which main program accepts the integers to be sorted Main program uses the fork system call to create a new process called a child process.Parent process sorts the integers using merge sort and waits for child process using wait system call to sort the integers using quick sort.Also demonstrate zombie and orphan states. \*/

# include<stdio.h>

# include <stdlib.h>

# include<sys/types.h>

# include<unistd.h>

int split ( int[], int , int );

void quickSort(int\* ,int, int);

void mergeSort(int arr[],int low,int mid,int high)

{

int i,j,k,l,b[20];

l=low;

i=low;

j=mid+1;

while((l<=mid)&&(j<=high)){

if(arr[l]<=arr[j])

{

b[i]=arr[l];

l++;

}

else

{

b[i]=arr[j];

j++;

}

i++;

}

if(l>mid)

{

for(k=j;k<=high;k++)

{

b[i]=arr[k];

i++;

}

}

else

{

for(k=l;k<=mid;k++)

{

b[i]=arr[k];

i++;

}

}

for(k=low;k<=high;k++)

{

arr[k]=b[k];

}

}

void partition(int arr[],int low,int high)

{

int mid;

if(low<high)

{

double temp;

mid=(low+high)/2;

partition(arr,low,mid);

partition(arr,mid+1,high);

mergeSort(arr,low,mid,high);

}

}

void display(int a[],int size)

{

int i;

for(i=0;i<size;i++){

printf("%d\t\t",a[i]);

}

printf("\n");

}

int main()

{

int pid, child\_pid;

int size,i,status;

/\* Input the Integers to be sorted \*/

printf("Enter the number of Integers to Sort::::\t");

scanf("%d",&size);

int a[size];

int pArr[size];

int cArr[size];

for(i=0;i<size;i++){

printf("Enter number %d:",(i+1));

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

pArr[i]=a[i];

cArr[i]=a[i];

}

/\* Display the Enterd Integers \*/

printf("Your Entered Integers for Sorting\n");

display(a,size);

/\* Process ID of the Parent \*/

pid=getpid();

printf("Current Process ID is : %d\n",pid);

/\* Child Process Creation \*/

printf("[ Forking Child Process ... ] \n");

child\_pid=fork(); /\* This will Create Child Process and

Returns Child's PID \*/

if( child\_pid < 0){

/\* Process Creation Failed ... \*/

printf("\nChild Process Creation Failed!!!!\n");

exit(-1);

}

else if( child\_pid==0) {

/\* Child Process \*/

printf("\nThe Child Process\n");

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

printf("\nparent of child process is %d",getppid());

printf("Child is sorting the list of Integers by QUICK SORT::\n");

quickSort(cArr,0,size-1);

printf("The sorted List by Child::\n");

display(cArr,size);

printf("Child Process Completed ...\n");

sleep(10);

printf("\nparent of child process is %d",getppid());

}

else {

/\* Parent Process \*/

printf("parent process %d started\n",getpid());

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

sleep(30);

printf("The Parent Process\n");

printf("Parent %d is sorting the list of Integers by MERGE SORT\n",pid);

partition(pArr,0,size-1);

printf("The sorted List by Parent::\n");

display(pArr,size);

wait(&status);

printf("Parent Process Completed ...\n");

}

return 0;

}

int split ( int a[ ], int lower, int upper )

{

int i, p, q, t ;

p = lower + 1 ;

q = upper ;

i = a[lower] ;

while ( q >= p )

{

while ( a[p] < i )

p++ ;

while ( a[q] > i )

q-- ;

if ( q > p )

{

t = a[p] ;

a[p] = a[q] ;

a[q] = t ;

}

}

t = a[lower] ;

a[lower] = a[q] ;

a[q] = t ;

return q ;

}

void quickSort(int a[],int lower, int upper){

int i ;

if ( upper > lower )

{

i = split ( a, lower, upper ) ;

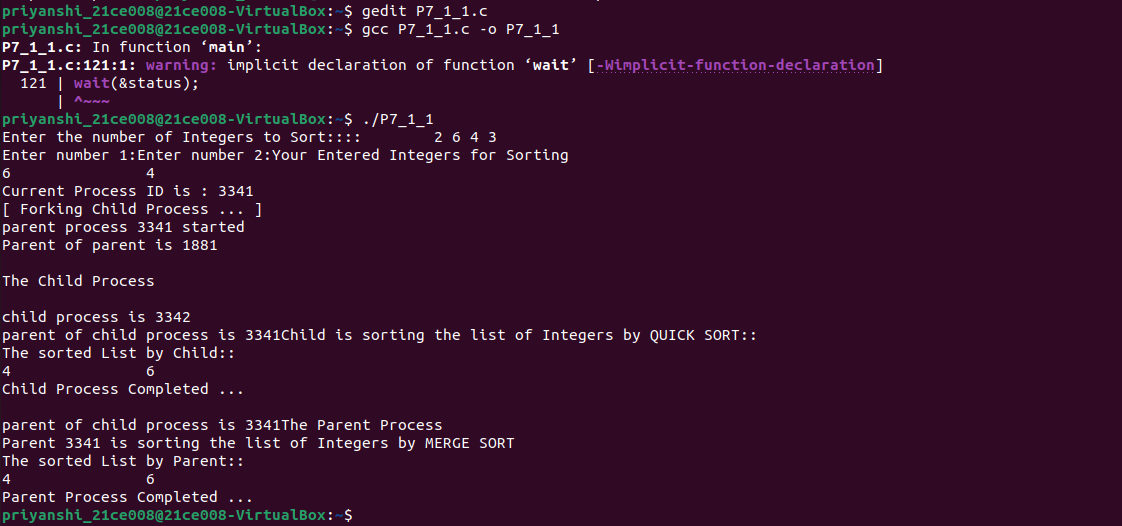
quickSort ( a, lower, i - 1 ) ;

quickSort ( a, i + 1, upper ) ;

}

}

Output:



1. Sort-Search (Parent-child process)

Code:

main

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int main(int argc, char \*argv[])

{

int val[10],ele;

pid\_t pid;

char\* cval[10];

char \*newenviron[] = { NULL };

int i,j,n,temp;

printf("\nEnter the size for an array: ");

scanf("%d",&n);

printf("\nEnter %d elements : ", n);

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

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

printf("\nEntered elements are: ");

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

printf("\t%d",val[i]);

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

{

for(j=0;j<n-1;j++)

{

if(val[j]>val[j+1])

{

temp=val[j];

val[j]=val[j+1];

val[j+1]=temp;

}

}

}

printf("\nSorted elements are: ");

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

printf("\t%d",val[i]);

printf("\nEnter element to search: ");

scanf("%d",&ele);

val[i] = ele;

for (i=0; i < n+1; i++)

{

char a[sizeof(int)];

snprintf(a, sizeof(int), "%d", val[i]);

cval[i] = malloc(sizeof(a));

strcpy(cval[i], a);

}

cval[i]=NULL;

pid=fork();

if(pid==0)

{

char \*child\_executable = "./Child"; // Update the path if needed

execve(child\_executable, cval, newenviron);

perror("Error in execve call...");

}

}

Child

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*argv[],char \*en[])

{

int i,j,c,ele;

int arr[argc];

for (j = 0; j < argc-1; j++)

{

int n=atoi(argv[j]);

arr[j]=n;

}

ele=atoi(argv[j]);

i=0;

j=argc-1;

c=(i+j)/2;

while(arr[c]!=ele && i<=j)

{

if(ele > arr[c])

i = c+1;

else

j = c-1;

c = (i+j)/2;

}

if(i<=j)

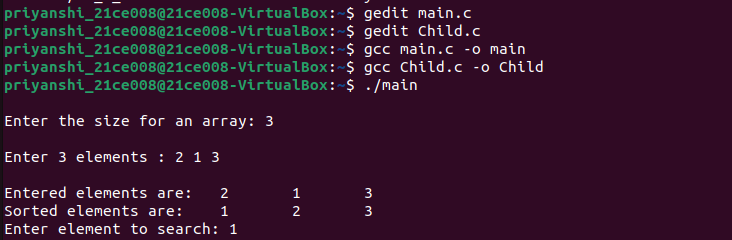
printf("\nElement Found in the given Array...!!!\n");

else

printf("\nElement Not Found in the given Array...!!!\n");

}

Output:



1. Execvparent and execvchild

Code:

execvParent

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc, char \*argv[]){

printf("\n-> PID of parent.c = %d ",getpid());

int parent;

parent = fork();

if(parent == -1)

{

printf("\n-> Some errors in calling ");

}

if(parent == 0)

{

printf("\n-> The child process is running.");

printf("\n-> Now execv will call child.c from child process.");

char \*args[]={"239",NULL};

execv("./execvchild",args);

}

else{

printf("\n-> Now parent is running \n");

}

return 0;

}

execvChild

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc,char \*argv[]){

printf("\n-> Now we are in child.c");

printf("\n-> The PID of child .c = %d",getpid());

printf("\n-> %s",\*argv);

return 0;

}

Output:

