**Practical 7**

**Aim:** **Process control system calls:**

1. **The demonstration of fork ()**
2. **execve() and wait() system calls along with zombie and orphan states.**

**Code:**

**A) Demonstration of fork()**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

printf("Calling fork() to create child process\n");

int pid;

pid = fork();

if (pid == 0)

{

printf ( "\nParent Process id : %d \n",getpid() );

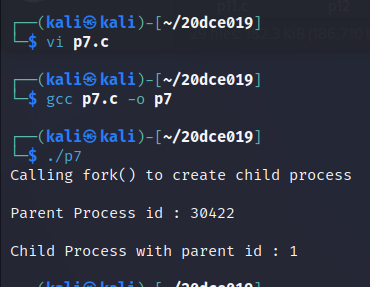
printf ( "\nChild Process with parent id : %d\n", getppid() );

}

return 0;

}

**Output:**

****

**B) execve() and wait() system calls along with zombie and orphan states.**

**Parent Process c code**

**Parent Process C code:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

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

{

int val[10],ele,status;

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]);

wait(&status);

printf("\nparrent process complete");

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)

{

execve(argv[1], cval, newenviron);

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

}

}

**Child process C code:**

#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)

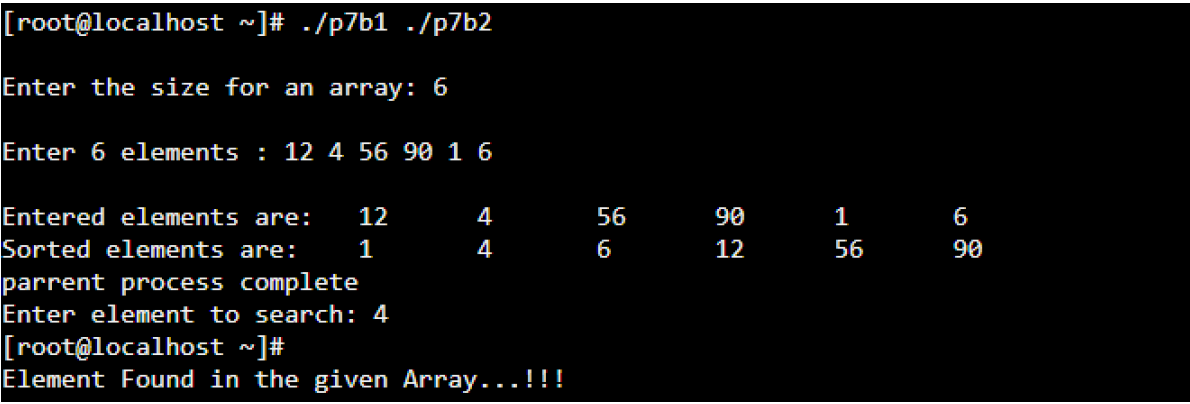
printf("\nElement Found in the given Array...!!!\n");

else

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

}

**OUTPUT:**

****

**CONCLUSION:**

In this practical I learnt about zombie and orphan processes. Zombie processes are created when the process dies but it isn’t removed from process table. Orphan processes are created when the parent for some reason is terminated and it dies, leaving the child process as an orphan process.