Name: Prathamesh Sudhir Paraswar

Class: TE9

Batch: L9

Roll NO:33155

Assignment 2A:

1)Fork: 1) Fork is basically used to create a new process from the parent process. The calling process becomes parent process and the called process becomes the child process.

2) The fork function copies the parent's memory image so that the new process receives a copy of the address space of the parent.

3) Two process execute separately after the fork call.

4) Ex. pid = fork(void);

The fork function returns 0 for child process. If child process is not created successfully then it returns -1.

2)Wait: 1) When a process creates a child, both parent and child proceed with execution from the point of the fork.

2) The parent can execute wait to block until the child finishes. The wait function causes the caller to suspend execution until a child's status becomes available or until the caller receives a signal.

3) To create a zombie process we call wait in parent process.

4) To create a orphan process we call wait in child process.

5) ex. Wait(time_to_wait);

3) Zombie: A zombie process is a process in its terminated state. This usually happens in a program that has parent-child functions. After a child function has finished execution, it sends an exit status to its parent function. Until the

parent function receives and acknowledges the message, the child function remains in a “zombie” state, meaning it has executed but not exited.

4) Orphan: A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

Code:

```
#include <unistd.h>

#include<stdio.h>

#include<stdlib.h>

#define n 5

int no[n];

void sort(void)

{

  int i,j,temp;

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

  {

    for(j=i+1;j<5;j++)

    {

        if(no[i]>no[j])

        {

            temp=no[i];

            no[i]=no[j];
```

```
            no[j]=temp;
        }
    }
  }
}
void sort1(void)
{
  int i,j,temp;
  for(i=0;i<5;i++)
  {
    for(j=i+1;j<5;j++)
    {
        if(no[i]<no[j])
        {
            temp=no[i];
            no[i]=no[j];
            no[j]=temp;
        }
    }
  }
}

void xyz()
```

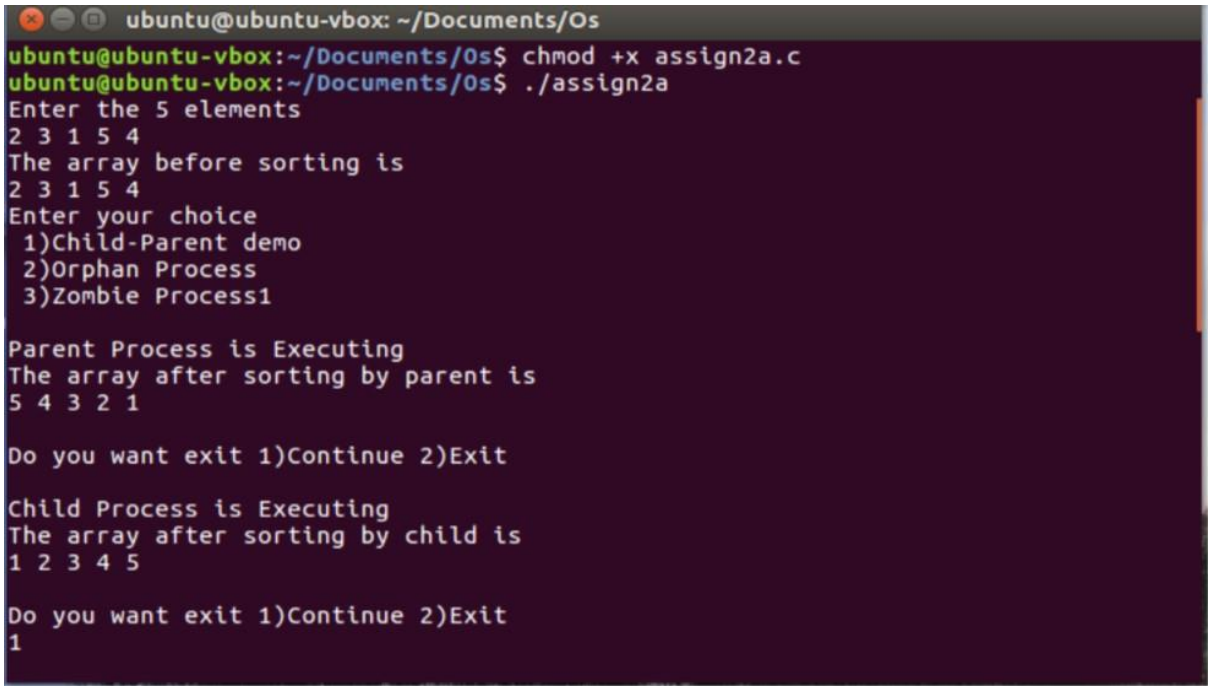
```
{
        if(fork()==0)
        {
                printf("\nChild Process is Executing\n");
                sort();
                printf("The array after sorting by child is\n");
                int j;
                for(j=0;j<5;j++)
                {
                        printf("%d",no[j]);
                        printf(" ");
                }
                printf("\n");
        }
        else{
                int status =0;
                printf("\nParent Process is Executing\n");
                sort1();
                printf("The array after sorting by parent is\n");
                int j;
                for(j=0;j<5;j++)
                {
                        printf("%d",no[j]);
```

```
                printf(" ");
            }
            printf("\n");
        }
}
void orphan()
{
        if(fork()==0)
        {
        printf("In child process\n");
        sleep(5);
        printf("Child Process ID: %d\n", getpid());
        printf("Parent Process ID: %d\n", getppid());
        printf("Child process is orphan now\n");
        printf("Parent Process ID: %d\n", getppid());
        printf("Child process completed\n");
        }
        else
        {
        printf("In parent Process\n");
        printf("Parent Process ID: %d\n", getpid());
        printf("Parent Process completed\n");
        }
```

```
}
void zombie()
{
        if(fork()==0)
        {
        printf("In child process\n");
        printf("Child process completed\n");
        printf("Child Process ID: %d\n", getpid());
        printf("Parent Process ID: %d\n", getppid());
        printf("Child process is zombie now\n");
        }
        else
        {
        printf("In parent Process\n");
        sleep(4);
        printf("Parent Process ID: %d\n", getpid());
        printf("Parent Process completed\n");
        }
}
int main(void)
{
 printf("Enter the 5 elements\n");
 for(int i=0;i<n;i++)
```

```
    scanf("%d",&no[i]);
 printf("The array before sorting is\n");
 int i;
 for(i=0;i<5;i++)
 {
        printf("%d",no[i]);
        printf(" ");
 }
int k;
do
{
printf("\nEnter your choice \n 1)Child-Parent demo \n 2)Orphan Process \n
3)Zombie Process");
int choice;
scanf("%d",&choice);
        if(choice==1)
        {
        xyz();
        }
        else if(choice==2)
        {
        orphan();
        }
        else if(choice==3)
```

```
        {
        zombie();
        }
        else
        {
        break;
        }
printf("\nDo you want exit 1)Continue 2)Exit\n");
scanf("%d",&k);
} while(k==1);
return 0;
}
```



```
ubuntu@ubuntu-vbox: ~/Documents/Os
Enter your choice
 1)Child-Parent demo
 2)Orphan Process
 3)Zombie Process2
In parent Process
Parent Process ID: 2172
Parent Process completed

Do you want exit 1)Continue 2)Exit
In child process
Child Process ID: 2181
Parent Process ID: 2172
Child process is orphan now
Child process completed

Do you want exit 1)Continue 2)Exit
1

Enter your choice
 1)Child-Parent demo
 2)Orphan Process
 3)Zombie Process3
In parent Process
In child process
```

```
ubuntu@ubuntu-vbox: ~/Documents/Os
Child process is orphan now
Child process completed

Do you want exit 1)Continue 2)Exit
1

Enter your choice
 1)Child-Parent demo
 2)Orphan Process
 3)Zombie Process3
In parent Process
In child process
Child process completed
Child Process ID: 2229
Parent Process ID: 2179
Child process is zombie now

Do you want exit 1)Continue 2)Exit
Parent Process ID: 2179
Parent Process completed

Do you want exit 1)Continue 2)Exit
2
ubuntu@ubuntu-vbox:~/Documents/Os$
```