OSL Assignment 2

Roll No. – 33255

1) 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 sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

```
#include <sys/types.h> //Importing required libraries
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>

void merge(int arr[], int l, int mid, int h) //Function to merge divided array
{
	int temp[h-l+1];
	int i = l;
	int j = mid+1;
	int k = 0;

	while(i<=mid && j<=h)
	{
		if(arr[i]<=arr[j])
		{
			temp[k] = arr[i];
			i++;
		}
		else
		{
			temp[k] = arr[j];
			j++;
		}
		k++;
	}

	while(i<=mid)
	{
		temp[k] = arr[i];
		i++;
		k++;
	}

	while(j<=h)
	{
		temp[k] = arr[j];
		j++;
		k++;
```

```
	}

	for(int x=l;x<=h;x++)
	{
		arr[x] = temp[x-l];
	}
}

void mergeSort(int arr[], int l, int h) //Merge sort function
{
	if(l>=h)
		return;

	int mid = (l+h)/2;
	mergeSort(arr, l, mid);
	mergeSort(arr, mid+1, h);
	merge(arr, l, mid, h);
}

int main() //main function
{
	int choice; //Variable to store choice of user

	printf("\n**********MENU**********\n"); //Displaying menu
	printf("1.Normal Execution (Sorting) \n");
	printf("2.Demonstrate Zombie State \n");
	printf("3.Demonstrate Orphan State \n");
	printf("\n");
	printf("Enter Your Choice : ");
	scanf("%d", &choice); //Taking input of choice from the user

	switch(choice) //Using switch-case on choice variable
	{
		case 1: //If choice==1, then normal execution
			int arr[5], x;
			printf("\n");
			printf("Enter Five Integers To Be Sorted \n");
			for(int i=0;i<5;i++) //Take five integers as input
			{
				printf("Enter Integer %d : ",i+1);
				scanf("%d", &x);
				arr[i] = x;
			}

			pid_t pid;
			pid = fork(); //Call fork() function and store returned value in a variable

			switch(pid) //Using switch-case on pid variable
			{
```

```
                case -1: //If pid==-1, then error occured while forking the process
                        printf("Error In Forking The Process! \n"); //Error occured while forking
the process
                        break;

                case 0: //If pid==0, then we are in child process
                        printf("\n");
                        printf("Inside Child Process \n"); //We are in the child process
                        mergeSort(arr, 0, 4); //Sort the given array
                        printf("Array Sorted By Child Process \n");
                        for(int i=0;i<5;i++) //Display sorted array
                        {
                                printf("%d ",arr[i]);
                        }
                        printf("\n");
                        exit(0);
                        break;

                default: //Else if pid>0, then we are in parent process
                        printf("\n");
                        printf("Inside Parent Process \n"); //We are inside parent process
                        mergeSort(arr, 0, 4); //Sort the given array
                        printf("Array Sorted By Parent Process \n");
                        for(int i=0;i<5;i++) //Display sorted array
                        {
                                printf("%d ",arr[i]);
                        }
                        printf("\n");
                        printf("Waiting For Child Process To Terminate..... \n \n");
                        wait(NULL); //Call wait() function to wait till the child has terminated
                        printf("\n");
                        printf("Child Process Has Been Terminated \n");
                        printf("Terminating Parent Process Now \n");
                        break;
        }
        break;

case 2: //If choice==2, then demonstrate zombie process
        pid_t pid1;
        pid1 = fork();

        switch(pid1)
        {
                case -1:
                        printf("Error In Forking The Process! \n");
                        break;

                case 0:
                        printf("\n");
```

```
                        printf("Inside Child Process \n");
                        printf("My Process ID = %d \n", getpid()); //Print PID of child
                        printf("My Parent's Process ID = %d \n", getppid()); //Print PID of parent
                        exit(0);
                        break;

                default:
                        sleep(5); //Putting parent process to sleep so that child process
terminates before it
                        printf("\n");
                        printf("Inside Parent Process \n");
                        printf("My Process ID = %d \n", getpid()); //Print PID of parent
                        printf("Demonstrating Zombie Process \n");
                        system("ps | grep a.out"); //Displaying current processes
                        printf("Child Process Is Dead And Has Become A Zombie (defunct) \n\n");
                        break;
        }
        break;

    case 3: //If choice==3, then demonstrate orphan process
        pid_t pid2;
        pid2 = fork();

        switch(pid2)
        {
                case -1:
                        printf("Error In Forking The Process! \n");
                        break;

                case 0:
                        sleep(2); //Putting child process to sleep so that parent process
terminates before it. This makes the child orphan and it is adopted by some other process.
                        printf("\n");
                        printf("Inside Child Process \n");
                        printf("My Process ID = %d \n", getpid()); //Print PID of child
                        printf("My Parent's Process ID = %d \n", getppid()); //Print PID of parent
                        system("ps | grep a.out");
                        printf("This Process Has Been Adopted By Process %d \n", getppid());
//Print PID of parent that has adopted this child process
                        printf("Its Original Parent Process %d Has Been Terminated \n", (getpid()-
1)); //Print PID of original parent
                        exit(0);
                        break;

                default:
                        printf("\n");
                        printf("Inside Parent Process \n");
                        printf("My Process ID = %d \n", getpid()); //Print PID of original parent
                        system("ps | grep a.out");
```

```
                                        printf("Process %d Is My Child Process \n", (getpid()+1)); //Print PID of
child
                                        printf("\n");
                                        break;
                        }
                        break;
        }
}
```

Output

```
nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$ gcc Assignment2a.c
nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$ ./a.out

**********MENU**********
1.Normal Execution (Sorting)
2.Demonstrate Zombie State
3.Demonstrate Orphan State

Enter Your Choice : 1

Enter Five Integers To Be Sorted
Enter Integer 1 : 3
Enter Integer 2 : 7
Enter Integer 3 : 4
Enter Integer 4 : 1
Enter Integer 5 : 9

Inside Parent Process
Array Sorted By Parent Process
1 3 4 7 9
Waiting For Child Process To Terminate......


Inside Child Process
Array Sorted By Child Process
1 3 4 7 9

Child Process Has Been Terminated
Terminating Parent Process Now
```

```
nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$ ./a.out

**********MENU**********
1.Normal Execution (Sorting)
2.Demonstrate Zombie State
3.Demonstrate Orphan State

Enter Your Choice : 2

Inside Child Process
My Process ID = 2874
My Parent's Process ID = 2873

Inside Parent Process
My Process ID = 2873
Demonstrating Zombie Process
   2873 pts/0    00:00:00 a.out
   2874 pts/0    00:00:00 a.out <defunct>
Child Process Is Dead And Has Become A Zombie (defunct)

nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$ ./a.out

**********MENU**********
1.Normal Execution (Sorting)
2.Demonstrate Zombie State
3.Demonstrate Orphan State

Enter Your Choice : 3

Inside Parent Process
My Process ID = 2886
   2886 pts/0    00:00:00 a.out
   2887 pts/0    00:00:00 a.out
Process 2887 Is My Child Process

nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$
Inside Child Process
My Process ID = 2887
My Parent's Process ID = 1183
   2887 pts/0    00:00:00 a.out
This Process Has Been Adopted By Process 1183
Its Original Parent Process 2886 Has Been Terminated
```

2) Implement the C program in which main program accepts an integer array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an integer array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void merge(int arr[], int l, int mid, int h) //Function to merge divided array
{
```

```
	int temp[h-l+1];
	int i = l;
	int j = mid+1;
	int k = 0;

	while(i<=mid && j<=h)
	{
		if(arr[i]<=arr[j])
		{
			temp[k] = arr[i];
			i++;
		}
		else
		{
			temp[k] = arr[j];
			j++;
		}
		k++;
	}

	while(i<=mid)
	{
		temp[k] = arr[i];
		i++;
		k++;
	}

	while(j<=h)
	{
		temp[k] = arr[j];
		j++;
		k++;
	}

	for(int x=l;x<=h;x++)
	{
		arr[x] = temp[x-l];
	}
}

void mergeSort(int arr[], int l, int h) //Merge sort function
{
	if(l>=h)
		return;

	int mid = (l+h)/2;
	mergeSort(arr, l, mid);
	mergeSort(arr, mid+1, h);
	merge(arr, l, mid, h);
```

```
}

int main()
{
	int arr[5], x;

	printf("\n");
	printf("Enter Five Integers To Be Sorted \n");
	for(int i=0;i<5;i++) //Take five integers as input
	{
		printf("Enter Integer %d : ",i+1);
		scanf("%d", &x);
		arr[i] = x;
	}

	pid_t pid;
	pid = fork(); //Call fork() function and store returned value in a variable

	switch(pid)
	{
		case -1: //If pid==-1, then error occured while forking the process
			printf("Error In Forking The Process! \n"); //Error occured while forking the process
			break;

		case 0: //If pid==0, then we are in child process
			printf("\n");
			printf("Inside Child Process \n"); //We are in the child process
			mergeSort(arr, 0, 4); //Sort the given array
			printf("Array Sorted By Child Process \n");
			for(int i=0;i<5;i++) //Display sorted array
			{
				printf("%d ",arr[i]);
			}
			printf("\n");
			char** sarr = (char**)malloc(7*sizeof(char*)); //Declaring array of strings
			sarr[0] = "./reverse";
			sarr[6] = NULL;
			int j=0;
			for(int i=1;i<6;i++)
			{
				sarr[i] = (char*)malloc(12*sizeof(char)); //Allocating memory for a string
				sprintf(sarr[i], "%d", arr[j++]); //Converting integer to string
			}
			execv("./reverse", sarr); //Calling exec function to replace current process by reverse
program
			break;


		default: //Else if pid>0, then we are in parent process
```

```
                printf("\n");
                printf("Inside Parent Process \n"); //We are inside parent process
                printf("Waiting For Child Process To Terminate..... \n");
                wait(NULL);
                printf("Execution Of Child Process Has Been Completed \n");
                printf("Ending Parent Process Now \n\n");
                break;
        }
}
```

Reverse.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char* argv[])
{
        printf("\n");
        printf("Inside Reverse Program \n");
        printf("Printing Sorted Array In Reverse Order \n");
        for(int i=5;i>=1;i--)
        {
                int x = atoi(argv[i]);
                printf("%d ",x);
        }
        printf("\n\n");
}
```

```
nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$ gcc Assignment2b.c
nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$ ./a.out

Enter Five Integers To Be Sorted
Enter Integer 1 : 8
Enter Integer 2 : 3
Enter Integer 3 : 6
Enter Integer 4 : 9
Enter Integer 5 : 4

Inside Parent Process
Waiting For Child Process To Terminate.....

Inside Child Process
Array Sorted By Child Process
3 4 6 8 9

Inside Reverse Program
Printing Sorted Array In Reverse Order
9 8 6 4 3

Execution Of Child Process Has Been Completed
Ending Parent Process Now

nehul_nikude@Nehul-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/OS/Assignment 2$
```