Assignment No. 2

Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states

A. 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.

2_fork.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h> // For malloc and free
#include <sys/wait.h> // For wait()

// Function to perform bubble sort
void bubbleSort(int arr[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}


// Function to perform selection sort

void selectionSort(int arr[], int n) {
    int temp, small;
    for (int i = 0; i < n - 1; i++) {
        small = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[small] > arr[j]) {
                small = j;
            }
        }
        temp = arr[i];
        arr[i] = arr[small];
        arr[small] = temp;
    }
}
```

```
int main() {
    int n;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n]; // Original array
    printf("Enter array elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Allocate memory for the array copy
    int *arr_copy = (int *)malloc(n * sizeof(int));
    if (arr_copy == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Copy the original array
    for (int i = 0; i < n; i++) {
        arr_copy[i] = arr[i];
    }

    pid_t pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("fork failed");
        free(arr_copy);
        return 1;
    }

    if (pid == 0) {
        // Child process
        printf("This is the child process (PID: %d)\n", getpid());

        // Perform bubble sort
        bubbleSort(arr_copy, n);

        printf("The sorted array in the child process is:\n");
        for (int i = 0; i < n; i++) {
            printf("%d ", arr_copy[i]);
        }
        printf("\n");

        // Exit child process
        exit(0);
    } else {
        // Parent process
        printf("This is the parent process (PID: %d)\n", getpid());
        // Wait for the child process to complete
        printf("Parent process waiting for the child process to complete...\n");
        wait(NULL);

        // Perform selection sort
        selectionSort(arr, n);

        printf("The sorted array in the parent process is:\n");
        for (int i = 0; i < n; i++) {
            printf("%d ", arr[i]);
        }
        printf("\n");

        // Demonstrate the zombie state:
        printf("Parent process sleeping to demonstrate the zombie state...\n");
        sleep(15);  // Sleep to allow observation of the zombie process

        // After sleep, child process should be cleaned up by the kernel
        printf("Parent process waking up from sleep...\n");
    }

    // Free the allocated memory
    free(arr_copy);

    return 0;
}
```

OUTPUT:

```
pl-17@pl17-OptiPlex-3020:~/07IT$ gcc 2_fork.c
pl-17@pl17-OptiPlex-3020:~/07IT$ ./a.out
Enter the number of elements: 6
Enter array elements: 45
12
86
25
10
3
This is the parent process (PID: 4766)
Parent process waiting for the child process to complete...
This is the child process (PID: 4774)
The sorted array in the child process is:
3 10 12 25 45 86
The sorted array in the parent process is:
3 10 12 25 45 86
Parent process sleeping to demonstrate the zombie state...
Parent process waking up from sleep...
```

B. Implement the C program in which main program accepts an array. Main program uses the FORKsystem call to create a new process called a child process. Parent process sorts an array and passesthe sorted array to child process through the command line arguments of

EXECVE system call. Thechild process uses EXECVE system call to load new program which display array in reverse order.
2_b1.c

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

int main(int cnt,char *arr[])
{
	int n;

	char temp[10];

	for(int i=1;i<cnt-1;i++)
	{
		for(int j=1;j<cnt-i;j++)
		{
			if(strcmp(arr[j],arr[j+1])>0)   // order  // if arr[j] comes after arr[j+1] move forward
			{
				strcpy(temp,arr[j]);
				strcpy(arr[j],arr[j+1]);
				strcpy(arr[j+1],temp);
			}
		}
	}
		printf("\nSorted Array:");
		for(int i=1;i<cnt;i++)
		{
			printf("%s",arr[i]);
		}

		printf("\nENter integer to execute execv:");
		scanf("%d",&n);
		execv("./s.out",arr);
		//this execv is for accessing the o/p of the b2 file
}
```

2_b2.c

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

int main(int cnt,char *arr[])
{
	printf("This is second process %d",getpid());
	printf("\nReversed array:");
	for(int i=cnt-1;i>=1;i--){
		printf("\n%s\n",arr[i]);
```

```
        }
}
```

OUTPUT:

```
pl-17@pl17-OptiPlex-3020:~/07IT$ gcc 2_b1.c
pl-17@pl17-OptiPlex-3020:~/07IT$ gcc 2_b2.c -o s.out
pl-17@pl17-OptiPlex-3020:~/07IT$ ./a.out  b s e a

Sorted Array:abes
ENter integer to execute execv:1
This is second process 5488
Reversed array:
s
e
b
a
```