Assignment 2: **The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states. Implement the C program in which the 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 the child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load a new program that uses.**

Name: **Chaitanya Suresh Uge**
Roll Number: **333064**
GR Number: **21910718**
Division: C **Batch: C3**

Source Code:

1. first.c

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include<sys/types.h>
4   #include<sys/wait.h>
5   #include<string.h>
6   #include <unistd.h>
7
8   void swap(int *x, int *y)
9   {
10   int temp = *x;
11   *x = *y;
12   *y = temp;
13  }
14
15  int partition(int arr[], int low, int high)
16  {
17   int i, j, pivot = arr[high];
18   i = (low - 1);
19
20   for(j = low; j <= high- 1; j++)
21   {
22    if(arr[j] < pivot)
23    {
```

```
24      i++;
25      swap(&arr[i], &arr[j]);
26    }
27   }
28
29   swap(&arr[i + 1], &arr[high]);
30   return (i + 1);
31  }
32
33  void quickSort(int arr[], int low, int high, int n)
34  {
35   if(low < high)
36   {
37    int pi = partition(arr, low, high);
38    quickSort(arr, low, pi - 1, n);
39    quickSort(arr, pi + 1, high, n);
40   }
41
42  }
43
44
45  int main(int argc, char *argv[])  // main fun takes three inputs: int main(int argc,
46  char *argv[], char *envp[])
47  {
48   pid_t pid;
49   int n, i;
50   char *arr[10];
51   printf("Enter the Size of Array: ");
52   scanf("%d", &n);
53   int a[n];
54   printf("\nEnter Array Elements: ");
55
56   for(i = 0; i < n; i++)
57   {
58     scanf("%d", &a[i]);
59   }
60
61   printf("\nEntered Array: ");
62
63   for(i = 0; i < n; i++)
64   {
65    printf("%d ", a[i]);
66   }
67
68   quickSort(a, 0, (n - 1), n); // calling quckSort function
69
70   printf("\nSorted Array: ");
71
72   for(i = 0; i < n; i++)
73   {
74    printf("%d ", a[i]);      // printing sorted array
75    char buf[sizeof(int)];    // converting int array to char pointer array because
```

```
76  execve fun requires char pointer array
77     snprintf(buf, sizeof(int), "%d", a[i]);
78     arr[i] = malloc(sizeof(buf));
79     strcpy(arr[i], buf);
80   }
81
82   arr[i] = NULL;              // end of string
83   pid = fork();
84
85   if(pid == 0)              // Child process
86   {
87     printf("\nInside the child Process, Parent pid = %d, Child pid = %d\n", getppid(),
88 getpid());
89
90     // execve function takes 3 args: int execve(const char *filename, char *const
91 argv[], char *const envp[])
92     if(execve(argv[1], arr, NULL) == -1)    // Calling execve function
93     {
94       printf("\nERROR\n");
95     }
96
97     printf("\nChild Process Terminating...\n");
98   }
99
100  else
101  {
102    printf("\nInside the Parent Process, ppid = %d\nExecuting and Terminating Child
103 Process (if any)\n", getpid());
104    wait(NULL);
       printf("\nParent Process Terminating...\n");
     }

     return 0;
   }
```

2. second.c

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include<sys/types.h>
4   #include<sys/wait.h>
5   #include <unistd.h>
6   #include <string.h>
7
8   int main(int argc, char *argv[])
9   {
10   int i, j, c, s, arr[argc];
11   printf("\nThe value of argc is %d", argc);
12   printf("\nSearch Element: ");
13   scanf("%d", &s);
14
15   for(i = 0; i < argc; i++)
16   {
17    arr[i] = atoi(argv[i]);  // char to int
18   }
19
20   i = 0;
21   j = argc - 1;
22   c = (i + j) / 2;
23
24   while(s != arr[c] && i <= j)
25   {
26    if(arr[c] < s)
27    {
28     i = c + 1;
29    }
30    else
31    {
32     j = c - 1;
33    }
34
35    c = (i + j) / 2;
36   }
37
38   if(i <= j)
39   {
40    printf("\nElement Present\n");
41   }
42
43   else
44   {
45    printf("\nElement Absent\n");
46   }
47
48   return 0;
49  }
```

Output:

Compilation:

```
[root@localhost mark2]# gcc first.c -o first.exe
[root@localhost mark2]# gcc second.c -o second.exe
[root@localhost mark2]# ls
first.c  first.exe  second.c  second.exe
[root@localhost mark2]#
```

Execution:

```
[root@localhost mark2]# ./first.exe ./second.exe
Enter the Size of Array: 6

Enter Array Elements: 4 3 2 5 1 6

Entered Array: 4 3 2 5 1 6
Sorted Array: 1 2 3 4 5 6
Inside the Parent Process, ppid = 5294
Executing and Terminating Child Process (if any)
Sorted Array: 1 2 3 4 5 6
Inside the child Process, Parent pid = 5294, Child pid = 5295

The value of argc is 6
Search Element:
```

```
Search Element: 4

Element Present

Parent Process Terminating...
[root@localhost mark2]#
```