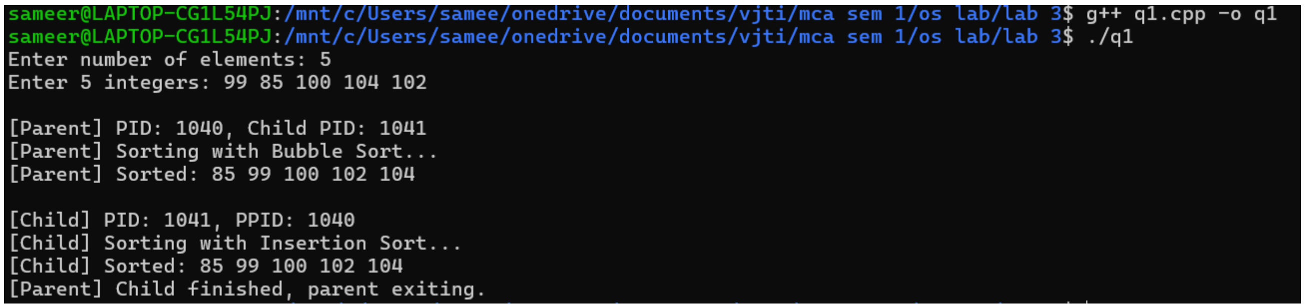
Sameer Gupta
OS Lab
October 7, 2025

Practical 3

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

Question 1. Implement the CPP 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.

```cpp
#include <iostream>
#include <vector>
#include <unistd.h>
#include <sys/wait.h>
#include <cstdlib>
using namespace std;
void bubbleSort(vector<int>& arr) {
    for (int i = 0; i < arr.size() - 1; i++)
        for (int j = 0; j < arr.size() - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
void insertionSort(vector<int>& arr) {
    for (int i = 1; i < arr.size(); i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter " << n << " integers: ";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    pid_t pid = fork();

    if (pid < 0) {
        cerr << "Fork failed!" << endl;
        exit(1);
    }
    else if (pid == 0) {
        // ---------- CHILD ----------
        cout << "\n[Child] PID: " << getpid()
             << ", PPID: " << getppid() << endl;
        cout << "[Child] Sorting with Insertion Sort..." << endl;
        insertionSort(arr);
        cout << "[Child] Sorted: ";
        for (int x : arr) cout << x << " ";
        cout << endl;
        exit(0);
    }
    else {
        // ---------- PARENT ----------
        cout << "\n[Parent] PID: " << getpid()
             << ", Child PID: " << pid << endl;
        cout << "[Parent] Sorting with Bubble Sort..." << endl;
        bubbleSort(arr);
        cout << "[Parent] Sorted: ";
        for (int x : arr) cout << x << " ";
        cout << endl;
        wait(NULL);
        cout << "[Parent] Child finished, parent exiting." << endl;
    }
}

```

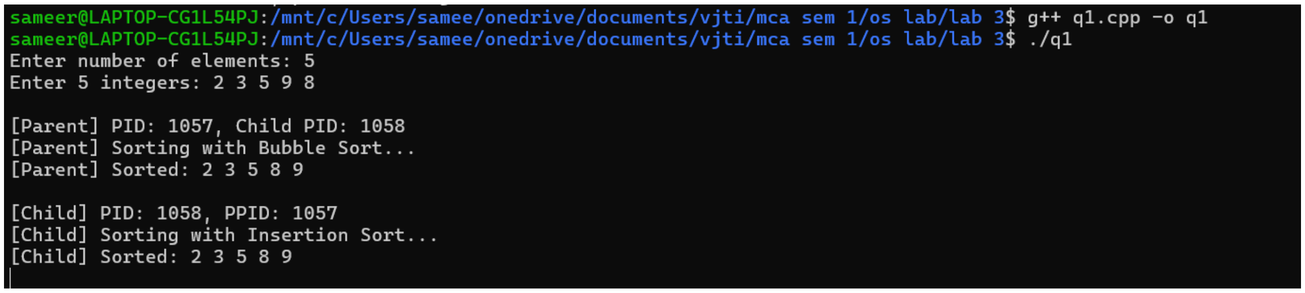


```
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ g++ q1.cpp -o q1
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ ./q1
Enter number of elements: 5
Enter 5 integers: 99 85 100 104 102

[Parent] PID: 1040, Child PID: 1041
[Parent] Sorting with Bubble Sort...
[Parent] Sorted: 85 99 100 102 104

[Child] PID: 1041, PPID: 1040
[Child] Sorting with Insertion Sort...
[Child] Sorted: 85 99 100 102 104
[Parent] Child finished, parent exiting.
```

Figure 1. Normal Execution

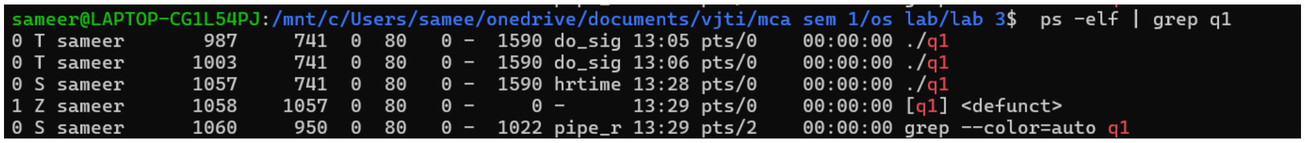
If we add `sleep(10)` and remove `wait(NULL)`, then the parent process keeps executing for a longer time. Thus child process keeps waiting resulting in the creation of a zombie process.

```
else {
    // ---------- PARENT ----------
    cout << "\n[Parent] PID: " << getpid()
         << ", Child PID: " << pid << endl;
    cout << "[Parent] Sorting with Bubble Sort..." << endl;
    bubbleSort(arr);
    cout << "[Parent] Sorted: ";
    for (int x : arr) cout << x << " ";
    cout << endl;
    sleep(10);
    cout << "[Parent] Child finished, parent exiting." << endl;
}
```



```
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ g++ q1.cpp -o q1
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ ./q1
Enter number of elements: 5
Enter 5 integers: 2 3 5 9 8

[Parent] PID: 1057, Child PID: 1058
[Parent] Sorting with Bubble Sort...
[Parent] Sorted: 2 3 5 8 9

[Child] PID: 1058, PPID: 1057
[Child] Sorting with Insertion Sort...
[Child] Sorted: 2 3 5 8 9
```

FIGURE 2. Parent Process still in execution

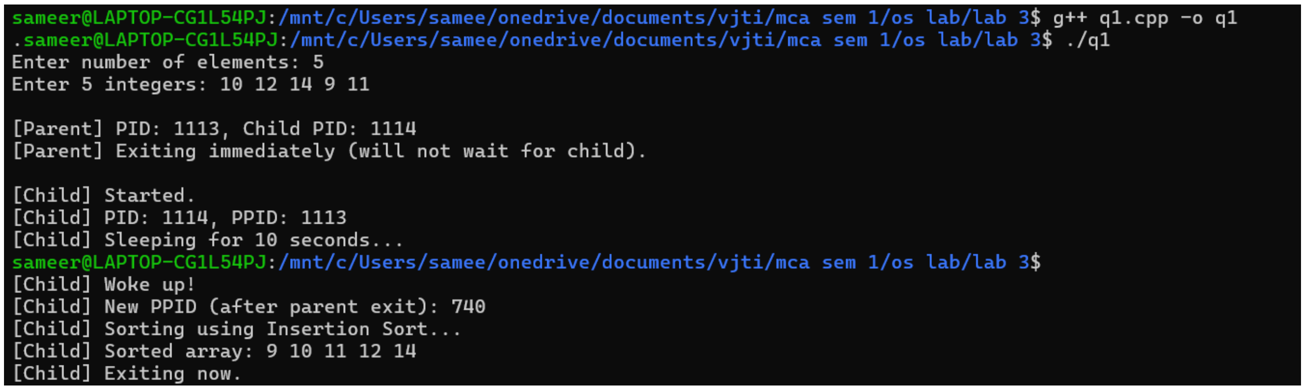


```
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$  ps -elf | grep q1
0 T sameer       987     741  0  80   0 -  1590 do_sig 13:05 pts/0    00:00:00 ./q1
0 T sameer      1003     741  0  80   0 -  1590 do_sig 13:06 pts/0    00:00:00 ./q1
0 S sameer      1057     741  0  80   0 -  1590 hrtime 13:28 pts/0    00:00:00 ./q1
1 Z sameer      1058    1057  0  80   0 -     0 -      13:29 pts/0    00:00:00 [q1] <defunct>
0 S sameer      1060     950  0  80   0 -  1022 pipe_r 13:29 pts/2    00:00:00 grep --color=auto q1
```

FIGURE 3. Zombie Process represented by 'Z'

If we remove `wait(NULL)` and force the parent process to exit immediately after execution, and add `sleep(10)` in the child process, then the child process becomes orphan and gets adopted by init.

```cpp
 else if (pid == 0) {
        // ---------------- CHILD PROCESS ----------------
        cout << "\n[Child] Started." << endl;
        cout << "[Child] PID: " << getpid()
             << ", PPID: " << getppid() << endl;
        cout << "[Child] Sleeping for 10 seconds..." << endl;
        sleep(10);
        cout << "\n[Child] Woke up!" << endl;
        cout << "[Child] New PPID (after parent exit): " << getppid() << endl;
        cout << "[Child] Sorting using Insertion Sort..." << endl;
        insertionSort(arr);
        cout << "[Child] Sorted array: ";
        for (int num : arr) cout << num << " ";
        cout << endl;
        cout << "[Child] Exiting now." << endl;
        exit(0);
    }
else {
    // ---------------- PARENT PROCESS ----------------
    cout << "\n[Parent] PID: " << getpid()
         << ", Child PID: " << pid << endl;
    cout << "[Parent] Exiting immediately (will not wait for child)." << endl;
    exit(0);
}
```



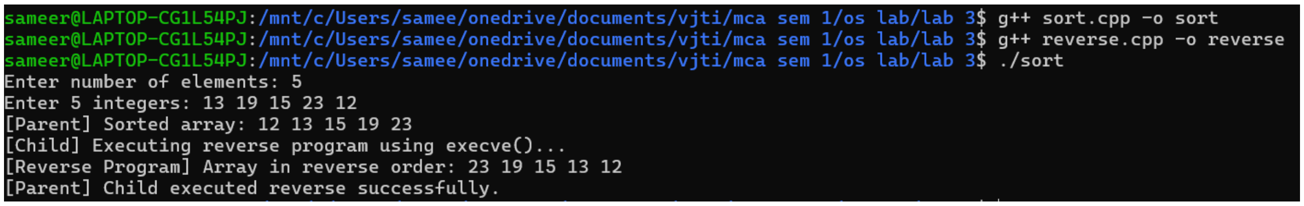
```
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ g++ q1.cpp -o q1
.sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ ./q1
Enter number of elements: 5
Enter 5 integers: 10 12 14 9 11

[Parent] PID: 1113, Child PID: 1114
[Parent] Exiting immediately (will not wait for child).

[Child] Started.
[Child] PID: 1114, PPID: 1113
[Child] Sleeping for 10 seconds...
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$
[Child] Woke up!
[Child] New PPID (after parent exit): 740
[Child] Sorting using Insertion Sort...
[Child] Sorted array: 9 10 11 12 14
[Child] Exiting now.
```

FIGURE 4. Parent ID for child process changes in orphan state.

Question 2. Implement the CPP program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an 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 displays array in reverse order.

```
# reverse.cpp
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    cout << "[Reverse Program] Array in reverse order: ";
    for (int i = argc - 1; i > 0; i--) {
        cout << argv[i] << " ";
    }
    cout << endl;
    return 0;
}
```

```
# sort.cpp
#include <iostream>
#include <vector>
#include <unistd.h>
#include <sys/wait.h>
#include <cstring>
#include <cstdlib>

using namespace std;
// Bubble Sort function (for parent)
void bubbleSort(vector<int>& arr) {
    for (int i = 0; i < arr.size() - 1; i++) {
        for (int j = 0; j < arr.size() - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter " << n << " integers: ";
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    bubbleSort(arr);
    cout << "[Parent] Sorted array: ";
    for (int num : arr)
        cout << num << " ";
    cout << endl;
    pid_t pid = fork();
    if (pid < 0) {
        cerr << "Fork failed!" << endl;
        exit(1);
    }
    else if (pid == 0) {
        // ---------- CHILD PROCESS ----------
        cout << "[Child] Executing reverse program using execve()...\n";
        vector<char*> args;
        args.push_back(strdup("./reverse"));
        for (int num : arr) {
            string s = to_string(num);
            args.push_back(strdup(s.c_str()));
        }
        args.push_back(NULL);
        execve("./reverse", args.data(), NULL);
        perror("execve failed");
        exit(1);
    }
    else {
        // ---------- PARENT PROCESS ----------
        wait(NULL);
        cout << "[Parent] Child executed reverse successfully.\n";
    }

    return 0;
}
```



```
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ g++ sort.cpp -o sort
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ g++ reverse.cpp -o reverse
sameer@LAPTOP-CG1L54PJ:/mnt/c/Users/samee/onedrive/documents/vjti/mca sem 1/os lab/lab 3$ ./sort
Enter number of elements: 5
Enter 5 integers: 13 19 15 23 12
[Parent] Sorted array: 12 13 15 19 23
[Child] Executing reverse program using execve()...
[Reverse Program] Array in reverse order: 23 19 15 13 12
[Parent] Child executed reverse successfully.
```

Figure 5. Execution using execve system call