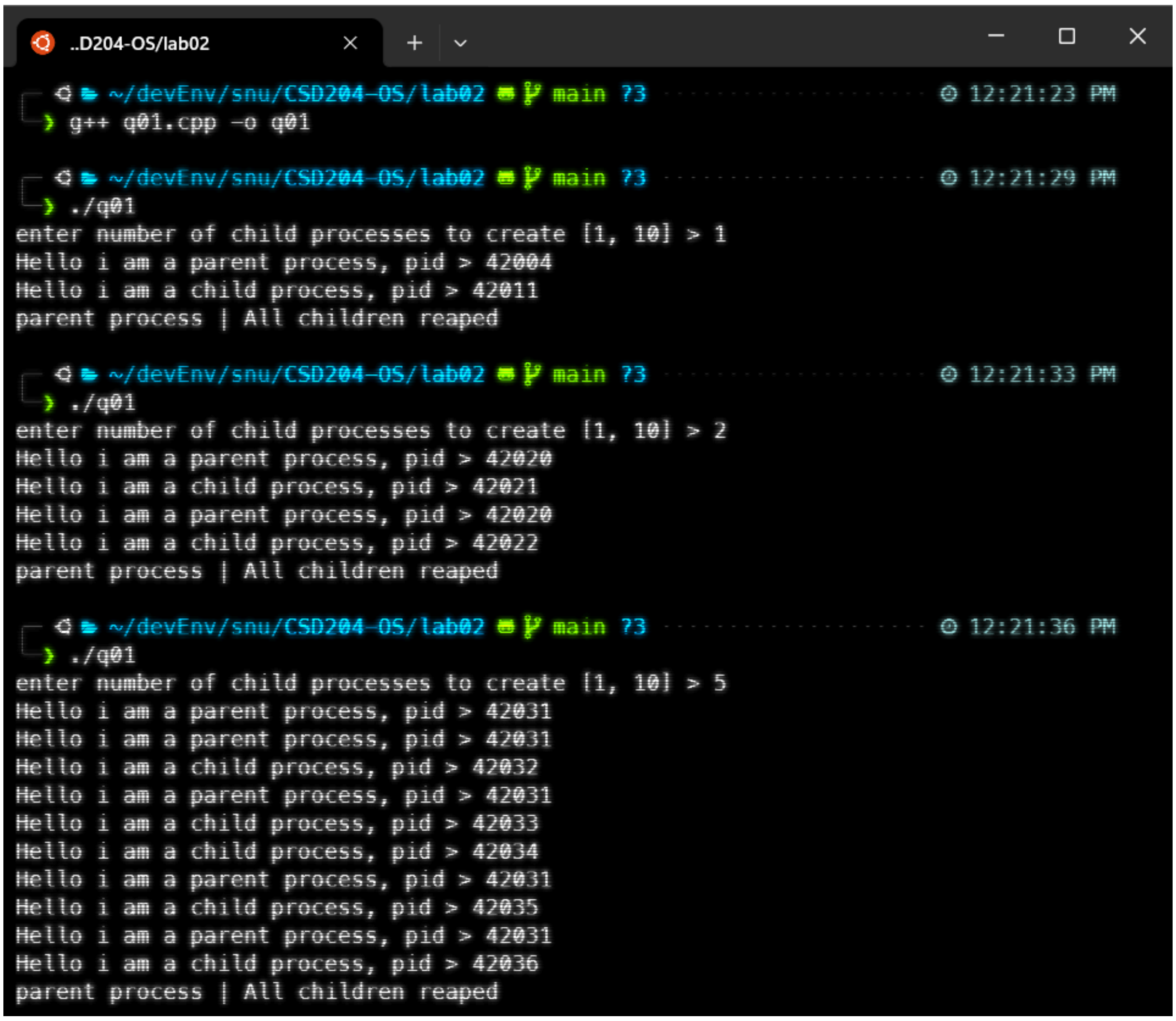
CSD204 - OS - Lab02

Lalit Maurya, 2310110164

Question 01



..D204-OS/lab02

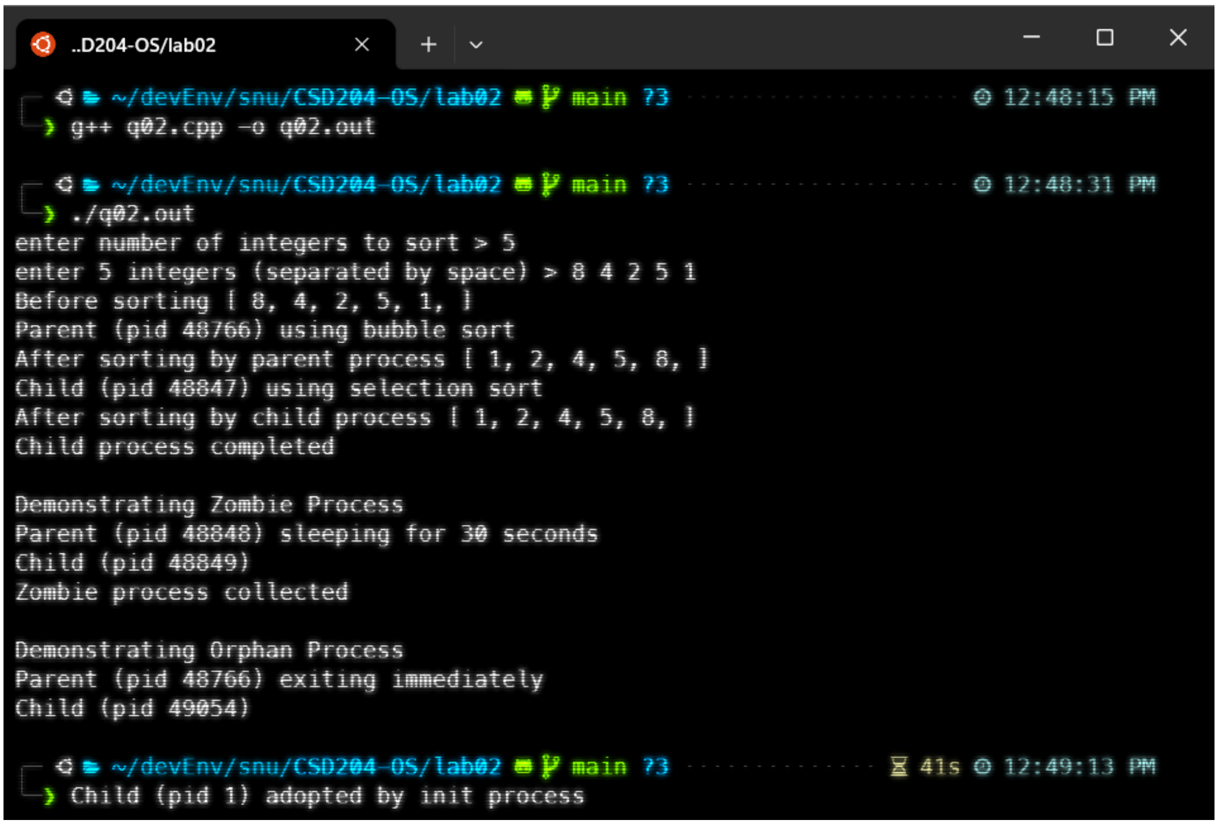
```
~/devEnv/snu/CSD204-OS/lab02  main ?3                              12:21:23 PM
❯ g++ q01.cpp -o q01

~/devEnv/snu/CSD204-OS/lab02  main ?3                              12:21:29 PM
❯ ./q01
enter number of child processes to create [1, 10] > 1
Hello i am a parent process, pid > 42004
Hello i am a child process, pid > 42011
parent process | All children reaped

~/devEnv/snu/CSD204-OS/lab02  main ?3                              12:21:33 PM
❯ ./q01
enter number of child processes to create [1, 10] > 2
Hello i am a parent process, pid > 42020
Hello i am a child process, pid > 42021
Hello i am a parent process, pid > 42020
Hello i am a child process, pid > 42022
parent process | All children reaped

~/devEnv/snu/CSD204-OS/lab02  main ?3                              12:21:36 PM
❯ ./q01
enter number of child processes to create [1, 10] > 5
Hello i am a parent process, pid > 42031
Hello i am a parent process, pid > 42031
Hello i am a child process, pid > 42032
Hello i am a parent process, pid > 42031
Hello i am a child process, pid > 42033
Hello i am a child process, pid > 42034
Hello i am a parent process, pid > 42031
Hello i am a child process, pid > 42035
Hello i am a parent process, pid > 42031
Hello i am a child process, pid > 42036
parent process | All children reaped
```

Output analysis:

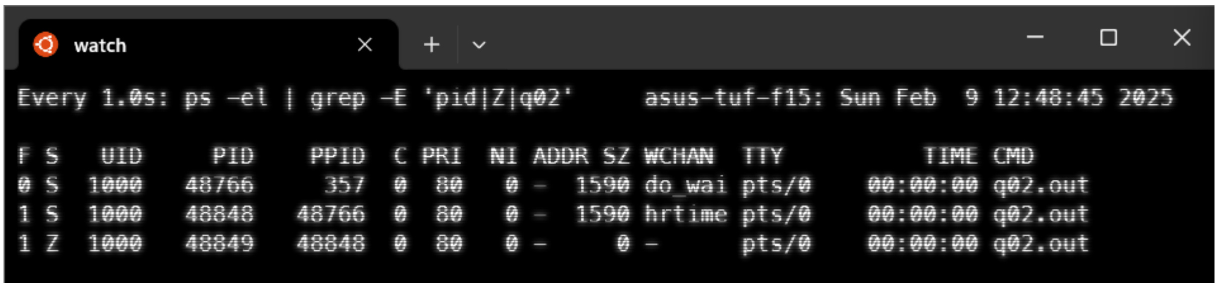
1. The parent process creates N child processes sequentially.
   - Each child is created after the previous one is reaped.
2. Each process prints its `PID`. The child processes exit after printing
3. After N forks, the parent uses `wait()` syscall to reap all its children

With N increasing, we can see the number of child processes increase. The `PID` of the parent does not change because fork sequentially instead of parallelly.

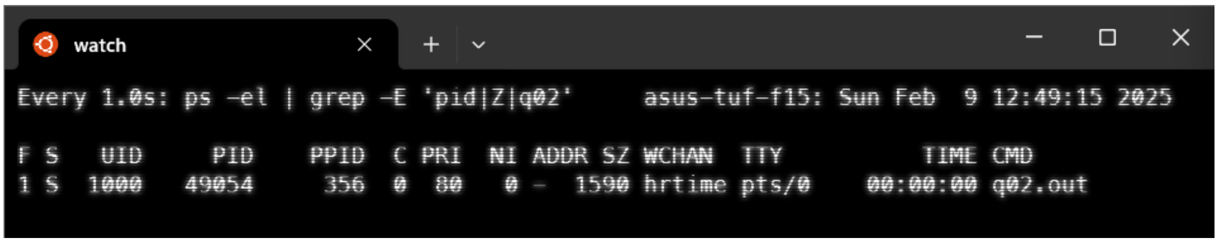
Question 02



```
~/devEnv/snu/CSD204-OS/lab02  main ?3                    12:48:15 PM
› g++ q02.cpp -o q02.out

~/devEnv/snu/CSD204-OS/lab02  main ?3                    12:48:31 PM
› ./q02.out
enter number of integers to sort > 5
enter 5 integers (separated by space) > 8 4 2 5 1
Before sorting [ 8, 4, 2, 5, 1, ]
Parent (pid 48766) using bubble sort
After sorting by parent process [ 1, 2, 4, 5, 8, ]
Child (pid 48847) using selection sort
After sorting by child process [ 1, 2, 4, 5, 8, ]
Child process completed

Demonstrating Zombie Process
Parent (pid 48848) sleeping for 30 seconds
Child (pid 48849)
Zombie process collected

Demonstrating Orphan Process
Parent (pid 48766) exiting immediately
Child (pid 49054)

~/devEnv/snu/CSD204-OS/lab02  main ?3              41s  12:49:13 PM
› Child (pid 1) adopted by init process
```



```
Every 1.0s: ps -el | grep -E 'pid|Z|q02'     asus-tuf-f15: Sun Feb  9 12:48:45 2025

F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   48766     357  0  80   0 -  1590 do_wai pts/0    00:00:00 q02.out
1 S  1000   48848   48766  0  80   0 -  1590 hrtime pts/0    00:00:00 q02.out
1 Z  1000   48849   48848  0  80   0 -     0 -      pts/0    00:00:00 q02.out
```



```
Every 1.0s: ps -el | grep -E 'pid|Z|q02'     asus-tuf-f15: Sun Feb  9 12:49:15 2025

F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
1 S  1000   49054     356  0  80   0 -  1590 hrtime pts/0    00:00:00 q02.out
```

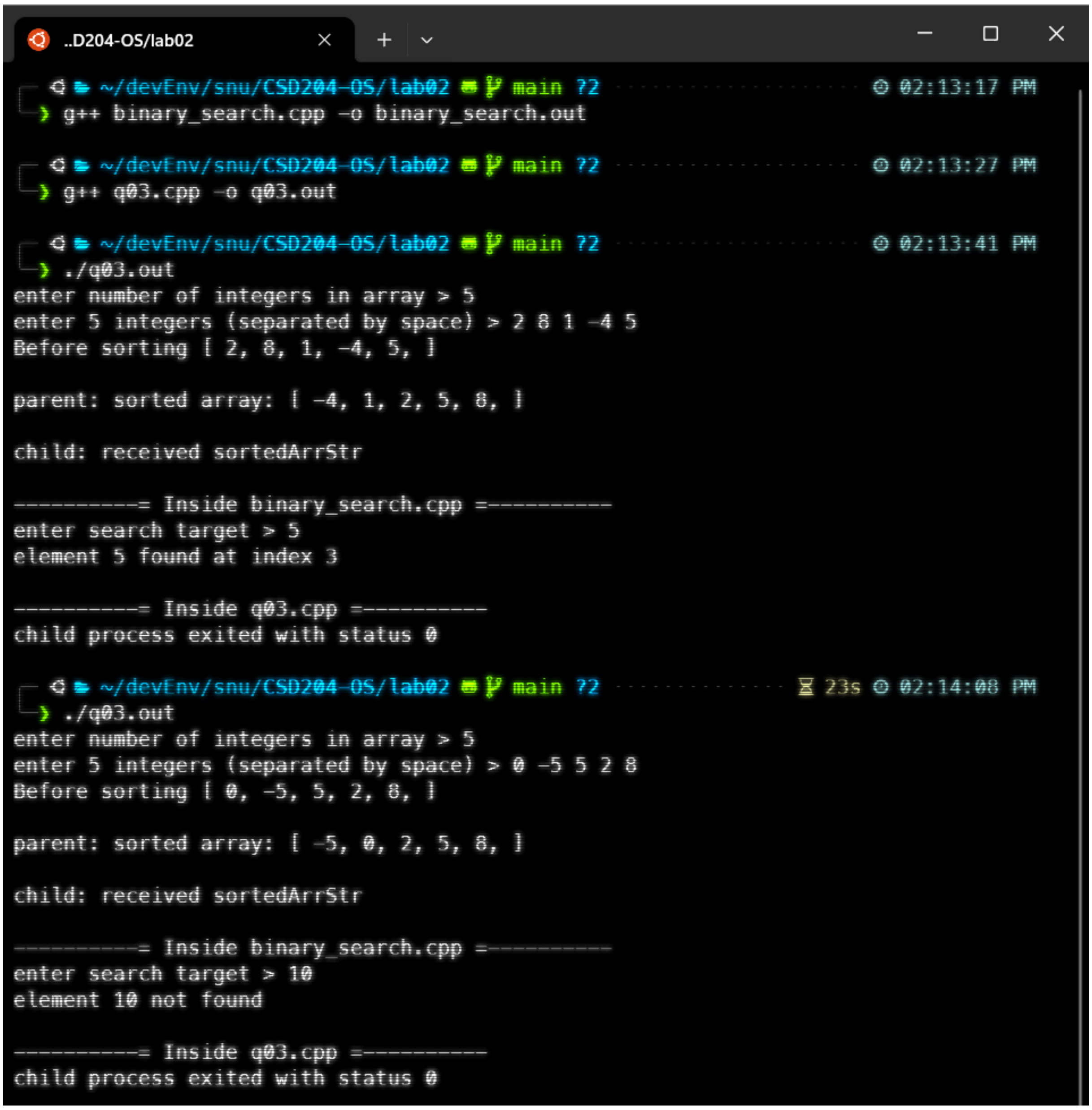
Output analysis:

- Sorting:
  1. We take user input for a vector of integers.
  2. We use the `fork()` syscall and pass the vector and a copy of it to the parent and child process respectively.
  3. Both processes list their `PID` and sorting method, and subsequently print their sorted vectors.
- Zombie Process:
  A zombie process is one that has finished executing but still has an entry in the process table.

We demonstrate that here by having the child process quit immediately while the parent process is sleeping for 30 seconds. In this time, the `wait()` syscall is not executed and the child process still exists even though it has finished execution. This process is marked with `'Z'`.

- Orphan Process:
  An orphan process is one whose parent no longer exists.
  - We demonstrate that here by having the parent quit immediately while the child process sleeps for 30 seconds.
  - When the parent process quits, the child is inherited by process with `PID` 356 which is a Relay to the `PID` 357 which is the shell process.
  - On a traditional linux system, the orphaned process would be inherited by `init`.
  - Note that I am using WSL which is why the inheritance of orphaned processes goes to the shell instead of the `init` process.

Question 03



```
~/devEnv/snu/CSD204-OS/lab02  main ?2                                    02:13:17 PM
❯ g++ binary_search.cpp -o binary_search.out

~/devEnv/snu/CSD204-OS/lab02  main ?2                                    02:13:27 PM
❯ g++ q03.cpp -o q03.out

~/devEnv/snu/CSD204-OS/lab02  main ?2                                    02:13:41 PM
❯ ./q03.out
enter number of integers in array > 5
enter 5 integers (separated by space) > 2 8 1 -4 5
Before sorting [ 2, 8, 1, -4, 5, ]

parent: sorted array: [ -4, 1, 2, 5, 8, ]

child: received sortedArrStr

----------= Inside binary_search.cpp =----------
enter search target > 5
element 5 found at index 3

----------= Inside q03.cpp =----------
child process exited with status 0

~/devEnv/snu/CSD204-OS/lab02  main ?2                              23s   02:14:08 PM
❯ ./q03.out
enter number of integers in array > 5
enter 5 integers (separated by space) > 0 -5 5 2 8
Before sorting [ 0, -5, 5, 2, 8, ]

parent: sorted array: [ -5, 0, 2, 5, 8, ]

child: received sortedArrStr

----------= Inside binary_search.cpp =----------
enter search target > 10
element 10 not found

----------= Inside q03.cpp =----------
child process exited with status 0
```

1. The main function of `q03.cpp` takes user input for a vector of integers.
2. It then creates uses the `pipe()` syscall for inter process communication.
3. The main function then uses the `fork()` syscall to create a parent and child process.
   - **Parent Process:**
     - The parent process takes the user input vector and sorts it using bubble sort
     - It then serializes it into a string which it writes the write end of the pipe we created earlier.
   - **Child Process:**
     - The child process reads from the read end of the pipe created earlier to receive the serialized sorted vector.
     - It then calls `binary_search.cpp` using the `exec()` syscall.
4. Inside `binary_search.cpp`
   - We first deserialize the array to convert it back into a vector.
   - We then run a simple binary search on said vector.
5. After running `binary_search.cpp`, we log exit status of child to ensure no errors occurred and quit.

Note: I made the assumption of using forks for parent child communication as I could not find any resources or guides for doing so using `exec()` syscalls. The `binary_search.cpp` program however, is called via the `exec()` syscalls.