Operating Systems Lab

Lab works # 07

Submitted By
QASIM ALI (20P-0070)
Submitted to :Muhammad Ahsan
(INSTRUCTOR CS)

DEPARTMENT OF COMPUTER SCIENCE

FAST NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES, PESHAWAR

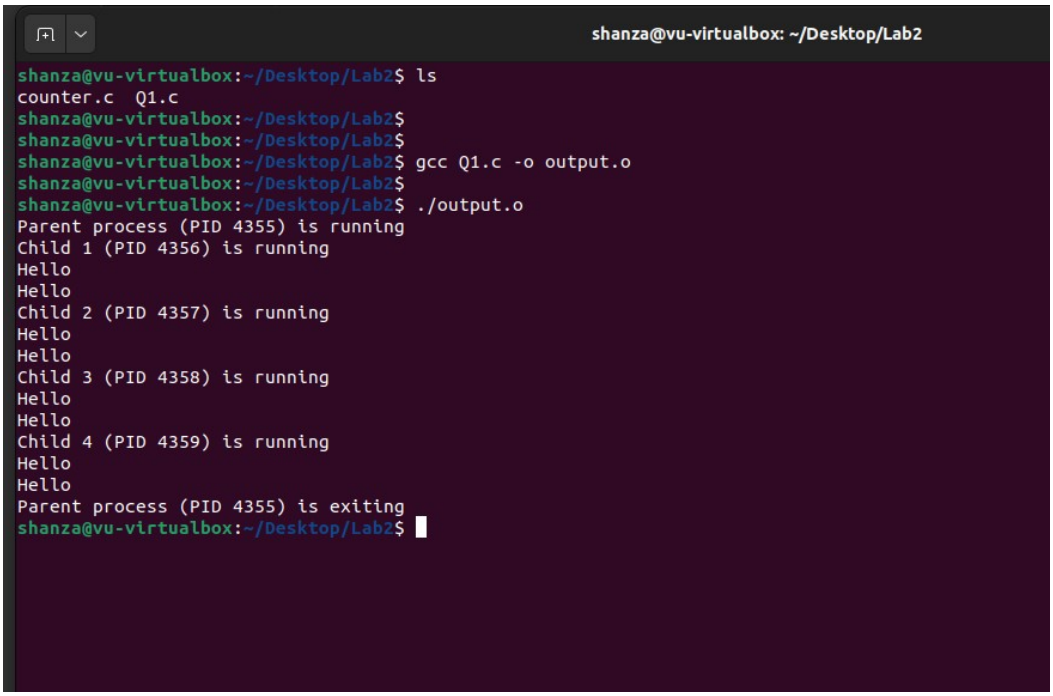
Session 2020-2024

Q1:

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

void printHello(int n) {
    int i = 0;
    while (i < n) {
        printf("Hello\n");
        i++;
    }
}

int main() {
    pid_t pid1, pid2, pid3, pid4;

    printf("Parent process (PID %d) is running\n", getpid());

    pid1 = fork();

    if (pid1 == 0) {
        // First child process
        printf("Child 1 (PID %d) is running\n", getpid());
        printHello(2);
    } else {
        // Parent process
        wait(NULL);
        pid2 = fork();

        if (pid2 == 0) {
            // Second child process
            printf("Child 2 (PID %d) is running\n", getpid());
            printHello(2);
        } else {
            // Parent process
            wait(NULL);
            pid3 = fork();

            if (pid3 == 0) {
                // Third child process
                printf("Child 3 (PID %d) is running\n", getpid());
                printHello(2);
            } else {
                // Parent process
                wait(NULL);
                pid4 = fork();

                if (pid4 == 0) {
                    // Fourth child process
```

```
            printf("Child 4 (PID %d) is running\n", getpid());
            printHello(2);
        } else {
            // Parent process
            wait(NULL);
            printf("Parent process (PID %d) is exiting\n", getpid());
        }
    }
  }
}

  return 0;
}
```

The output is :



Q2:

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

int main(int argc, char *argv[]) {
    // Check if a filename is provided as a command line argument
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // Step 1: Spawn a child process
    pid_t pid = fork();

    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Step 2: Child process calls exec to run with the command line argument
        execlp("cat", "cat", argv[1], NULL);
        // If execlp fails, print an error message and exit
        perror("exec");
        exit(EXIT_FAILURE);
    } else {
        // Step 3: Parent process calls wait to block until the child terminates
        int status;
        wait(&status);

        if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
            // Step 4: Parent process can execute some other command or perform additional actions
            printf("Parent process can execute additional actions here.\n");
        } else {
            // Parent termination status should be 0 if all has gone well
            printf("Parent process terminated successfully.\n");
        }
    }

    return 0;
}
```

The output is :
create also file of cat in folder

gcc counter.c -o counter
./counter cat.txt