

OS Lab – 7
Darshan J Patel
21BCE6061

1.

Code-

```
#include<stdio.h>
```

```
int main()
{
    int lanes[4][3] = {{3,5}, {1,3}, {2,4}, {0,2}}; // Arrival and Crossing time for each lane
    int current_lane = 0, total_wait_time = 0;

    printf("Lane\tAverage Waiting Time\n");

    for(int i=0; i<4; i++)
    {
        int waiting_time = 0;
        if(i!=0) {
            waiting_time += (lanes[i-1][1]-lanes[i][0]); // Calculate waiting time before starting the
signal.
            if(current_lane != i-1) {
                waiting_time += 180; // If the previous lane is not adjacent to current lane, add 3 minutes
to the waiting time.
            }
        }

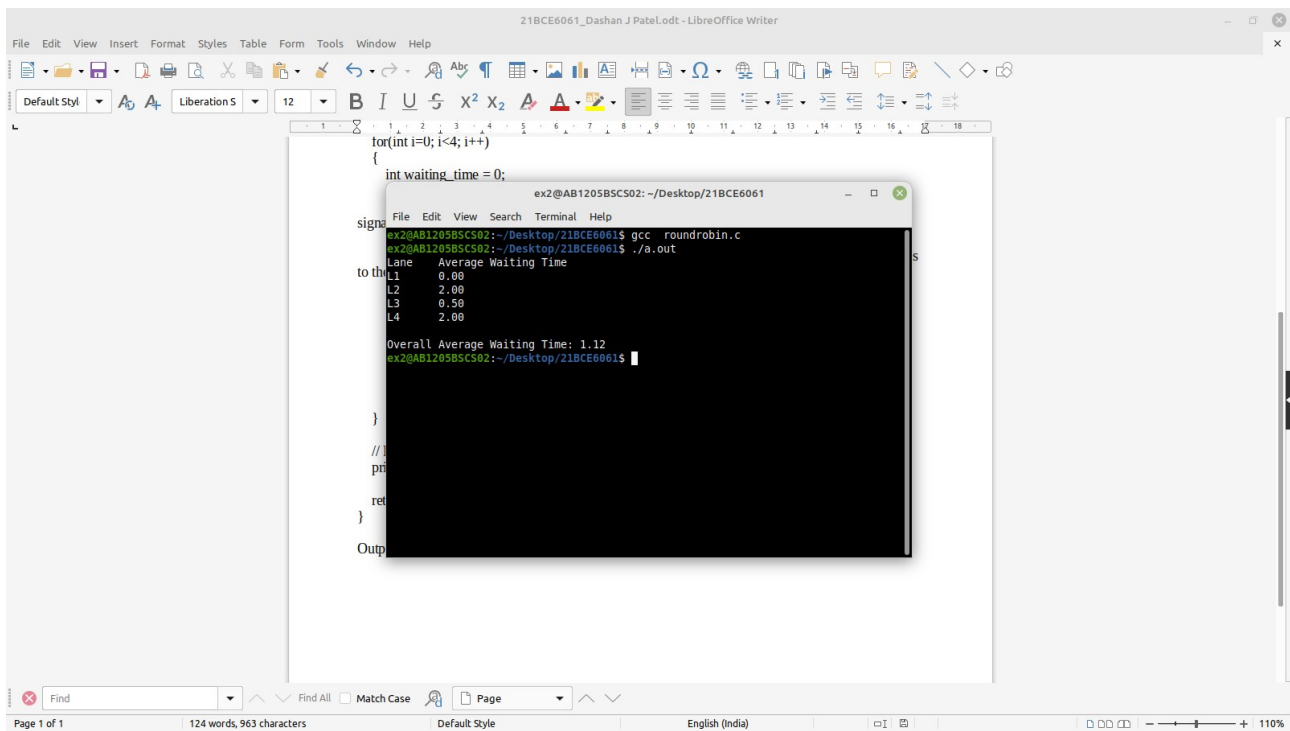
        // Print the average waiting time for the current lane
        printf("L%d\t%.2f\n", i+1, waiting_time/2.0);

        current_lane = i;
        total_wait_time += waiting_time;
    }

    // Print the overall average waiting time
    printf("\nOverall Average Waiting Time: %.2f\n", total_wait_time/8.0);

    return 0;
}
```

Output-



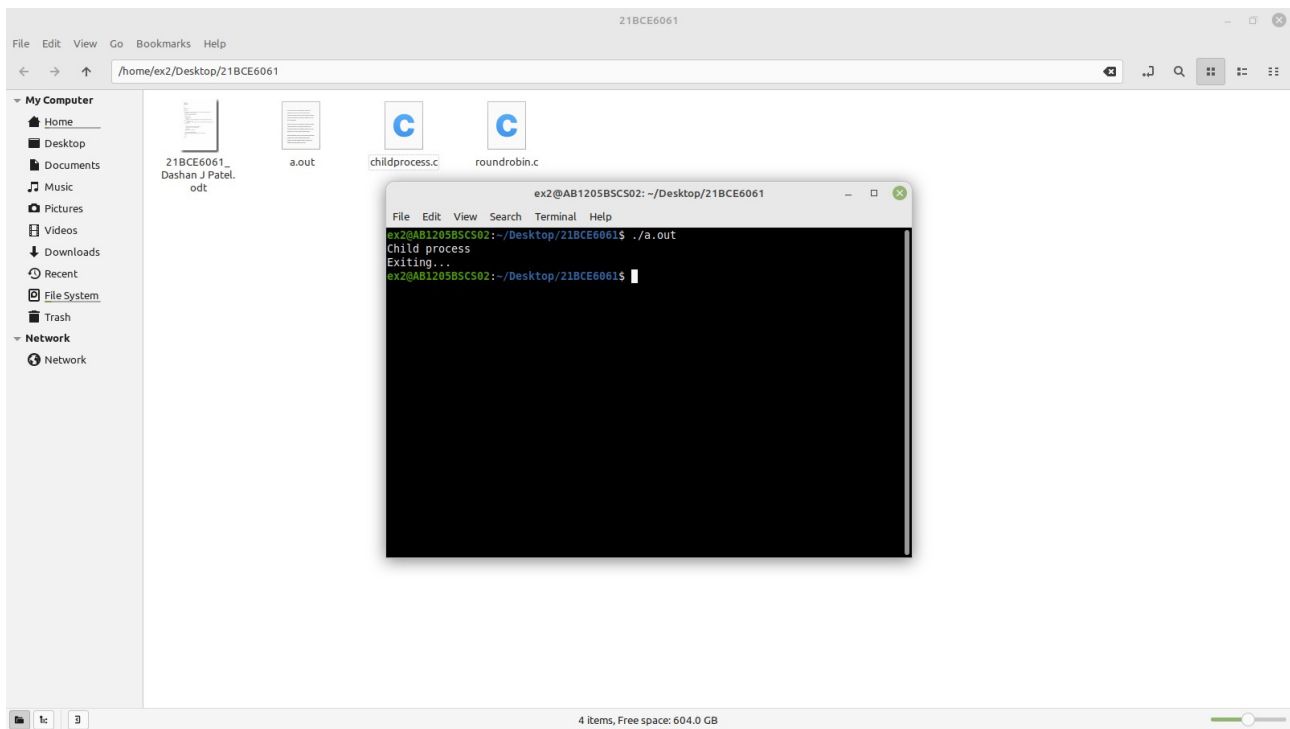
2.

```
#include<stdio.h>
#include<unistd.h>
```

```
int main() {
    pid_t pid = fork(); // Create a child process

    if(pid == 0) { // Child process
        printf("Child process\n");
        printf("Exiting...\n");
    }
    else { // Parent process
        sleep(5); // Wait for 5 seconds before terminating the parent process
    }

    return 0;
}
Output-
```



3.

```
#include <stdio.h>
#include <unistd.h>
```

```
int main() {
    pid_t pid = fork();

    if (pid == 0) { // child process
        sleep(5);
        printf("Child process %d\n", getpid());
        printf("Parent process %d\n", getppid());
    } else { // parent process
        printf("Parent process %d\n", getpid());
        sleep(2); // wait for 2 seconds
        printf("Orphan process %d\n", getpid());
    }
}
```

```
    return 0;
}
```

Output-

