

Project Report



Submitted By:

Muhammad Wakeel 2023-CS-601

Submitted By:

Asif Hussain 2023-CS-646

Submitted To:

Mam Mariyam Manzoor

Dated:

31th December 2025

Department of Computer Science

University of Engineering and Technology Lahore, New Campus

Project Title:

Hospital Resource Scheduler – A Linux-Based OS Project in C

Table of Content:

1. Introduction	3
2. Objectives	3
3. Implementation Details	3
3.1 CPU Scheduling Algorithms	3
3.2 Multithreaded Execution.....	5
3.3 Dynamic Memory Allocation	6
3.4 Process Creation.....	7
3.5 Inter-Process Communication (IPC)	8
3.6 Synchronization	9
5. Conclusion	10

1. Introduction

Hospitals receive many patients simultaneously, each requiring different services such as consultation, lab tests, or treatment. Due to limited resources (doctors, machines, rooms), inefficient scheduling can cause delays. This project simulates a hospital environment using Operating System concepts in C on Linux, where patient requests are treated as jobs and scheduled using CPU scheduling algorithms.

2. Objectives

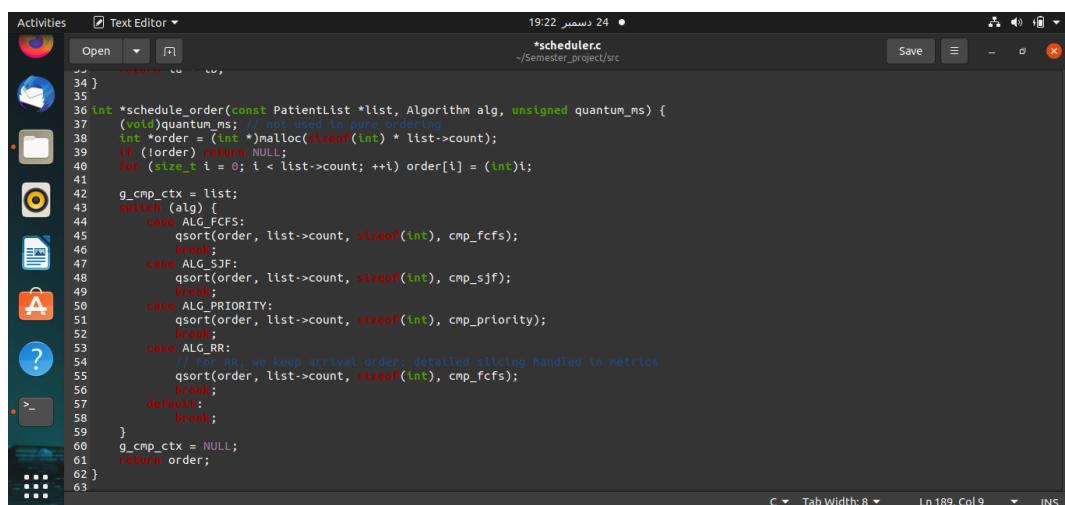
- Simulate patient requests as jobs
- Implement and compare CPU scheduling algorithms
- Execute patient jobs using multithreading
- Demonstrate process creation using fork() and exec()
- Implement IPC between scheduler and logger
- Ensure synchronization using semaphores and mutexes
- Use dynamic memory allocation for patient data

3. Implementation Details

3.1 CPU Scheduling Algorithms

The scheduler determines the execution order of patient requests.

Algorithms Implemented: FCFS, SJF, Priority, Round Robin



A screenshot of a Linux desktop environment showing a terminal window. The window title is "scheduler.c". The terminal displays the following C code:

```
Activities Text Editor 19:22 دسمبر 24 • *scheduler.c ~/Semester_project/src
Open F Save
34 }
35
36 int *schedule_order(const PatientList *list, Algorithm alg, unsigned quantum_ms) {
37     (void)quantum_ms; // not used in pure ordering
38     int *order = (int *)malloc(sizeof(int) * list->count);
39     if (!order) return NULL;
40     for (size_t i = 0; i < list->count; ++i) order[i] = (int)i;
41
42     g_cmp_ctx = list;
43     switch (alg) {
44         case ALG_FCFS:
45             qsort(order, list->count, sizeof(int), cmp_fcfs);
46             break;
47         case ALG_SJF:
48             qsort(order, list->count, sizeof(int), cmp_sjf);
49             break;
50         case ALG_PRIORITY:
51             qsort(order, list->count, sizeof(int), cmp_priority);
52             break;
53         case ALG_RR:
54             // For RR, we keep arrival order; detailed slicing handled in metrics
55             qsort(order, list->count, sizeof(int), cmp_fcfs);
56             break;
57         default:
58             break;
59     }
60     g_cmp_ctx = NULL;
61     return order;
62 }
```

Figure 3.0: CPU scheduling logic using qsort() for multiple algorithms

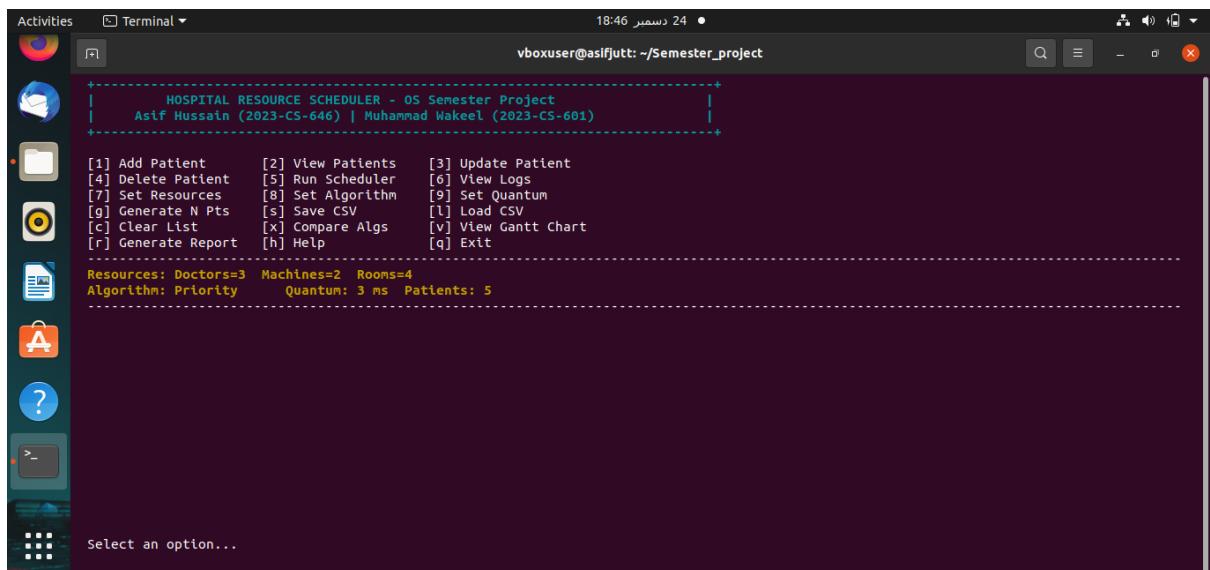


Figure 3.1: Main Manue Selections Interface

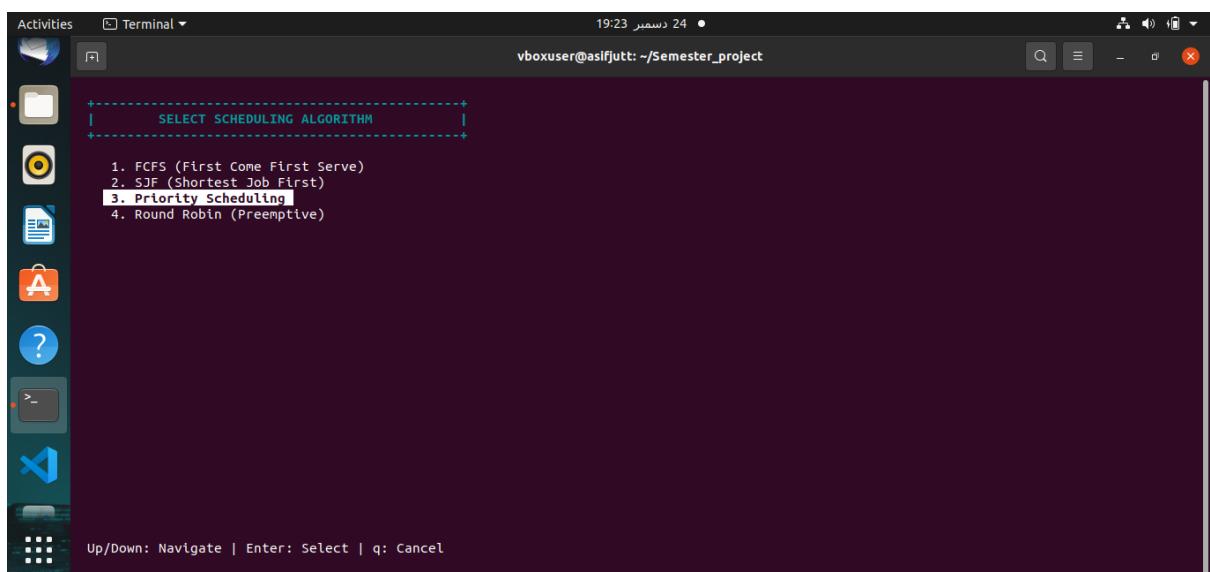


Figure 3.2: Scheduling algorithm selection interface

```

Activities Terminal 19:35 دسمبر 24 • vboxuser@asifjutt: ~/Semester_project
+-----+
| ALGORITHM COMPARISON REPORT |
+-----+
Patients: 5 | RR Quantum: 3 ms | Doctors: 3 | Machines: 2 | Rooms: 4
-----+
Algorithm      Avg Wait (ms)      Avg Turnaround (ms)      Winner
FCFS           1317.60            2094.20
SJF            1271.80            2048.40
Priority       1244.00            2020.60          BEST
Round Robin    2257.00            3034.40
-----+
ANALYSIS:
Best for Waiting Time: Priority (1244.00 ms)
Best for Turnaround Time: Priority (2020.60 ms)

Press any key to return...

```

Figure 3.3: Comparison of scheduling algorithms

3.2 Multithreaded Execution

Each patient is handled by a separate thread to simulate parallel hospital services.

```

Activities Text Editor 19:41 دسمبر 24 • vboxuser@asifjutt: ~/Semester_project/src
Open *scheduler.c thread_worker.c
scheduler.c
+-----+
16 void *patient_thread(void *arg) {
17     WorkerArgs *wa = (WorkerArgs *)arg;
18     Patient p = wa->patient;
19     sem_t *res = resource_for_service(wa->resources, p.service);
20
21     char buf[256];
22     snprintf(buf, sizeof(buf), "START id=%d name=%s service=%s\n", p.id, p.name, service_name(p.service));
23     write(wa->fifo_fd, buf, strlen(buf));
24
25     sem_wait(res);
26     ms_sleep(p.required_time_ms);
27     sem_post(res);
28
29     // Accumulate resource busy time (equals required time for non-preemptive service)
30     pthread_mutex_lock(&wa->resources->log_mutex);
31     switch (p.service) {
32         case SERVICE_CONSULTATION:
33             wa->resources->busy_doctors_ms += p.required_time_ms;
34             break;
35         case SERVICE_LAB_TEST:
36             wa->resources->busy_machines_ms += p.required_time_ms;
37             break;
38         case SERVICE_TREATMENT:
39             wa->resources->busy_rooms_ms += p.required_time_ms;
34             break;
40         default:
41             break;
42     }
43 }

```

Figure 3.4: Patient handling using POSIX threads

A screenshot of a Linux desktop environment. On the left, there's a vertical dock with icons for a browser, file manager, terminal, and other applications. The main window is a terminal window titled 'Activities' with the title bar 'Text Editor'. The terminal shows the path '/Semester_project/include' and the file 'thread_worker.h'. The code in the terminal is:

```
1 #ifndef THREAD_WORKER_H
2 #define THREAD_WORKER_H
3
4 #include <pthread.h>
5 #include "resources.h"
6 #include "ipc.h"
7
8 typedef struct {
9     Patient patient;
10    ResourcePool *resources;
11    int fifo_fd;
12 } WorkerArgs;
13
14 void *patient_thread(void *arg);
15
#endif // THREAD_WORKER_H
```

The status bar at the bottom of the terminal window shows 'C/Object Header' and 'Tab Width: 8'. The bottom right corner of the terminal window says 'Ln 16, Col 26' and 'INS'.

Figure 3.5: Thread creation for patient jobs

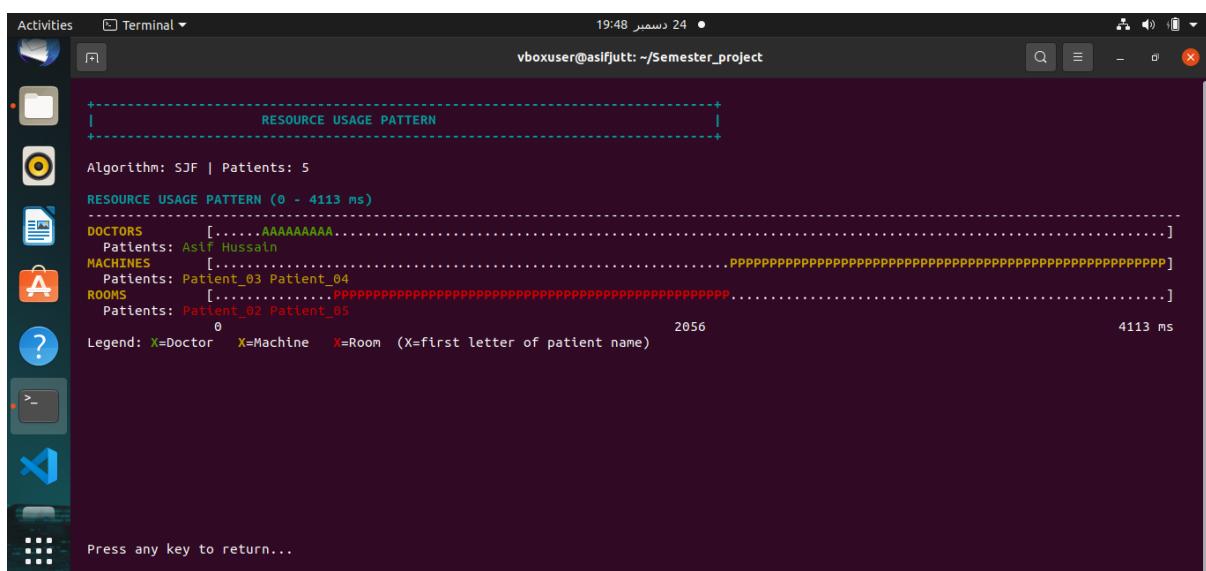


Figure 3.6: Parallel execution of patient threads

3.3 Dynamic Memory Allocation

Patient data is stored dynamically to support runtime changes.

The screenshot shows a terminal window titled "vboxuser@asifjutt: ~/Semester_project". The window displays a table titled "PATIENT QUEUE (5 patients)". The table has columns: ID, Name, Service, Priority, Req(ms), and Arrival(ms). The data is as follows:

ID	Name	Service	Priority	Req(ms)	Arrival(ms)
1	Asif Hussain	Consultation	2	300	230
2	Patient_02	Treatment	4	841	105
3	Patient_03	Lab Test	2	930	343
4	Patient_04	Lab Test	2	925	495
5	Patient_05	Treatment	1	887	40

At the bottom of the terminal, the message "Press any key to return..." is displayed.

Figure 3.7: Dynamic memory allocation for patient records

3.4 Process Creation

A separate logger process is created to record scheduling events.

The screenshot shows a code editor with multiple tabs open. The active tab is "ui.c". The code is as follows:

```

static void run_scheduler(uiState *st) {
    if (st->count == 0) {
        clear();
        mvprintw(3, 2, "No patients to schedule. Add patients first.");
        mvprintw(LINES-2, 2, "Press any key to return...");
        getch();
        return;
    }
    clear();
    mvprintw(2, 2, "Starting scheduler with %s algorithm...", alg_name(st->alg));
    refresh();
    // IPC setup
    if (ipc_setup_fifo() != 0) {
        clear(); mvprintw(3, 2, "Failed to setup FIFO."); getch(); return;
    }
    mqd_t mq = ipc_open_mq(1);
    int shm_fd = -1; Sharedstats *stats = NULL;
    int shm_ok = (ipc_setup_shm(&shm_fd, &stats, 1) == 0);
    if (!shm_ok) stats = NULL;
    pid_t pid = fork();
    if (pid == 0) {
        execl("bin/logger", "logger", (char *)NULL);
        _exit(127);
    }
}

```

Figure 3.9: Logger process creation using fork() and exec()

```

Activities Terminal 20:00 دسمبر 24 ●
vboxuser@asifjutt: ~/Semester_project
+-----+
| SCHEDULING RESULTS |
+-----+
Algorithm: SJF
PERFORMANCE METRICS:
Average Waiting Time: 1271.80 ms
Average Turnaround Time: 2048.40 ms
Total Execution Time: 984 ms
Completed Jobs: 5
RESOURCE UTILIZATION:
Doctors (3): 10.2%
Machines (2): 94.3%
Rooms (4): 43.9%
Logs saved to: logs/log.txt
Press any key to view RESOURCE USAGE PATTERN...

```

Figure 3.10: Parent (scheduler) and child (logger) processes

3.5 Inter-Process Communication (IPC)

Scheduler and logger communicate using FIFO, message queues, and shared memory.

```

Activities Text Editor 20:02 دسمبر 24 ●
ipc.c -~/Semester_project/src
Save
scheduler.c thread_worker.c patient.c thread_worker.h logger.c ui.c ipc.c
*ipc.c
55 int ipc_setup_shm(int *fd, SharedStats **stats_ptr, int create) {
56     int flags = create ? (O_CREAT | O_RDWR) : O_RDWR;
57     int shm_fd = shm_open(SHM_NAME, flags, 0666);
58     if (shm_fd == -1) {
59         perror("shm_open");
60         return -1;
61     }
62     if (create) {
63         if (ftruncate(shm_fd, (off_t)sizeof(SharedStats)) == -1) {
64             perror("ftruncate");
65             close(shm_fd);
66             return -1;
67         }
68     }
69     void *addr = mmap(NULL, sizeof(SharedStats), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
70     if (addr == MAP_FAILED) {
71         perror("mmap");
72         close(shm_fd);
73         return -1;
74     }
75     *fd = shm_fd;
76     *stats_ptr = (SharedStats *)addr;
77     return 0;
78 }
79
80 int ipc_cleanup_shm() {
81     shm_unlink(SHM_NAME);
82     return 0;
}

```

Figure 3.11: IPC implementation using FIFO and shared memory

```

Activities Terminal 20:05 دسمبر 24 ●
vboxuser@asifjutt: ~/Semester_project
EXECUTION LOGS

START id=5 name=Patient_05 service=Consultation
[MQ] STATS_READY

Final Report:
Average Waiting Time: 576.00 ms
Average Turnaround Time: 1089.60 ms
Completed Jobs: 5
START id=4 name=Patient_04 service=LabTest
START id=1 name=Patient_01 service=LabTest
START id=3 name=Patient_03 service=Treatment
START id=2 name=Patient_02 service=LabTest
FINISH id=5 name=Patient_05 service=Consultation
FINISH id=3 name=Patient_03 service=Treatment
FINISH id=4 name=Patient_04 service=LabTest
FINISH id=1 name=Patient_01 service=LabTest
FINISH id=2 name=Patient_02 service=LabTest
START id=5 name=Patient_05 service=Treatment
[MQ] STATS_READY

Final Report:
Average Waiting Time: 1244.00 ms
Average Turnaround Time: 2020.60 ms
Completed Jobs: 5

Press any key to return...

```

Figure 3.12: Inter-process communication in action

3.6 Synchronization

Semaphores and mutexes control access to limited resources.

```

Activities Text Editor 20:07 دسمبر 24 ●
resources.c -/Semester_project/src
Save
resources.c
resources.h
thread_worker.c
patient.c
thread_worker.h
logger.c
ui.c
ipc.c
resources.c

1 #include "resources.h"
2 #include <stdio.h>
3
4 int resources_init(ResourcePool *rp, int num_doctors, int num_machines, int num_rooms) {
5     if (sem_init(&rp->doctors, 0, (unsigned int)(num_doctors > 0 ? num_doctors : 1)) != 0) return -1;
6     if (sem_init(&rp->machines, 0, (unsigned int)(num_machines > 0 ? num_machines : 1)) != 0) return -1;
7     if (sem_init(&rp->rooms, 0, (unsigned int)(num_rooms > 0 ? num_rooms : 1)) != 0) return -1;
8     if (pthread_mutex_init(&rp->log_mutex, NULL) != 0) return -1;
9     rp->num_doctors = (num_doctors > 0 ? num_doctors : 1);
10    rp->num_machines = (num_machines > 0 ? num_machines : 1);
11    rp->num_rooms = (num_rooms > 0 ? num_rooms : 1);
12    rp->busy_doctors_ms = 0ULL;
13    rp->busy_machines_ms = 0ULL;
14    rp->busy_rooms_ms = 0ULL;
15    return 0;
16 }
17
18 void resources_destroy(ResourcePool *rp) {
19     sem_destroy(&rp->doctors);
20     sem_destroy(&rp->machines);
21     sem_destroy(&rp->rooms);
22     pthread_mutex_destroy(&rp->log_mutex);
23 }
24
25 sem_t *resource_for_service(ResourcePool *rp, ServiceType service) {
26     switch (service) {
27         case SERVICE_CONSULTATION: return &rp->doctors;
28         case SERVICE_LAB_TEST: return &rp->machines;
29     }
30 }

```

Figure 3.13: Resource initialization using semaphores

```
logger.c
sf_Shared /media/sf_Shared/Semester_project/src
12 static void *mq_reader(void *arg) {
13     FILE *out = (FILE *)arg;
14     mqd_t mq = ipc_open_mq(0);
15     if (mq == (mqd_t)-1) pthread_exit(NULL);
16
17     char msg[MQ_MSG_MAX];
18     unsigned prio;
19     while (1) {
20         ssize_t n = mq_receive(mq, msg, sizeof(msg), &prio);
21         if (n >= 0) {
22             msg[n] = '\0';
23             fprintf(out, "[MQ] %s\n", msg);
24             fflush(out);
25             if (strcmp(msg, "STATS_READY") == 0) {
26                 // Read shared memory stats
27                 int fd = -1; SharedStats *stats = NULL;
28                 if (ipc_setup_shm(&fd, &stats, 0) == 0) {
29                     if (fprintf(out, "\nFinal Report:\n") < 0);
30                     fprintf(out, "Average Waiting Time: %.2f ms\n", stats->avg_wait_ms);
31                     fprintf(out, "Average Turnaround Time: %.2f ms\n", stats->avg_turnaround_ms);
32                     fprintf(out, "Completed Jobs: %d\n", stats->completed_jobs);
33                     fflush(out);
34                     munmap(stats, sizeof(*stats));
35                     close(fd);
36                 }
37             } // After stats, continue reading until FIFO ends; do not exit here
38         }
39     }
40 }
```

Figure 3.14: Controlled access to shared resources

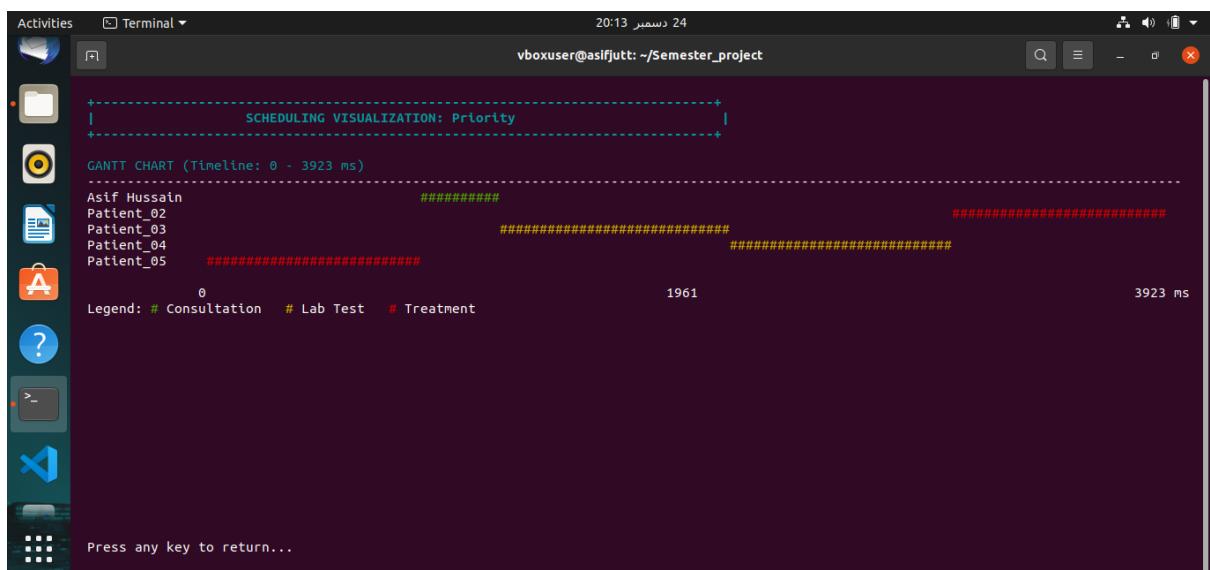


Figure 3.15: Gantt chart visualization of patient execution

5. Conclusion

This project demonstrates the practical application of Operating System concepts such as CPU scheduling, multithreading, IPC, synchronization, and dynamic memory management in a real-world hospital scenario. The system efficiently schedules patient requests while ensuring safe and synchronized resource usage.