EAST WEST UNIVERSITY

Semester: Fall 2025

Course Code: CSE325

Section: 05

Course title: Operating Systems

Project Report Name: Being Ted Mosby's

Submitted to:

Shatabdi Roy Moon

Lecturer

Department of Computer Science and Engineering

Submitted by: - Group: 2

Name	Id
Mahadi Prodhan	2023-2-60-033
Atiqur Rahman Miraj	2023-2-60-032
Md Saiful Islam Emon	2023-3-60-090
Shahriar Arefin	2023-3-60-517

Table of Contents

Project Description:

This project simulates an event planning company named “Ted’s Stories,” inspired by Ted Mosby’s romantic and serendipitous approach to life. The company manages multiple romantic events simultaneously, with limited event planners and shared resources such as proposal spots, decoration kits, and audiovisual equipment. The program uses threads to represent events, semaphores to manage resource availability and concurrency limits, and mutex locks to safely log activities. It demonstrates efficient synchronization and resource allocation in a multi-threaded environment.

Project Analysis:

The project models a real-world scenario where an event planning company must coordinate multiple events with limited staff and resources. Each event requires specific resources and runs for a random duration, simulating real-life event execution.

➢Strengths:

- Uses semaphores to control maximum concurrent events and shared resources.
- Implements mutex locks to protect shared logs and avoid race conditions.
- Demonstrates realistic event planning workflow with random resource requirements.
- Clear console output shows event states: waiting, acquired, running, completed.

➢Limitations:

- Fixed number of resources and events.
- Event planners are assigned randomly but not actively coordinated.
- No queue management for waiting events beyond semaphore limits.
- Console-based only; no GUI or persistent logging.

Overall Evaluation:

The project effectively demonstrates thread synchronization, resource management, and concurrent execution control in a multi-threaded C program. It serves as an educational tool for understanding semaphores, mutexes, and real-world concurrency problems.

Abstract:

The program simulates "Ted's Stories," an event planning company handling multiple romantic events. Each event is a thread that waits for an available slot, acquires necessary resources (proposal spots, decor kits, AV kits), executes for a random duration, and releases resources. Semaphores limit concurrent events and resource usage, while mutexes ensure safe logging. The simulation illustrates effective concurrency control in a resource-constrained environment.

Introduction:

The project simulates a romantic event planning company using multi-threading and synchronization primitives. It reflects real-world challenges such as resource sharing, scheduling, and concurrent execution.

➢Programming Language: C

C is chosen for its low-level control over threads and synchronization, making it ideal for concurrency simulations

➢ Data Structures:

- event_t struct to store event details (ID, planner, resource needs, duration).
- Arrays and semaphores for resource pools.

➢Functions (Modular Programming):

- oid* event_thread(void* arg) – Handles event lifecycle.
- main() – Initializes system, creates threads, and cleans up.

➢Console Input/Output (I/O):

printf() used to display customer actions and cafe status.

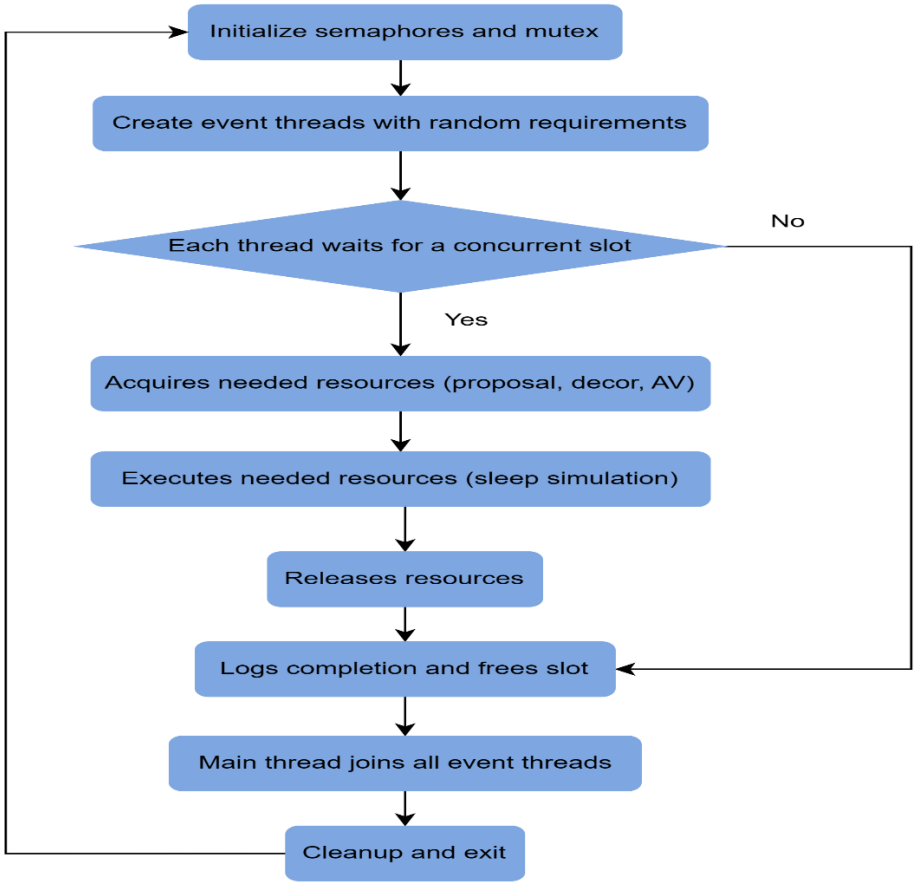
➢Control Structures:

- for loops for thread creation.
- if statements for resource allocation.
- sleep() to simulates event duration

➢Synchronization Techniques:

- Semaphores: Control concurrent events and resource pools.
- Mutex Lock: Protects shared log output

Flowchart:



The program logic follows this sequence:

1. Initialize semaphores and mutex.
2. Create event threads with random requirements.
3. Each thread waits for a concurrent slot.
4. Acquires needed resources (proposal, decor, AV).
5. Executes event (sleep simulation).

6. Releases resources.
7. Logs completion and frees slot.
8. Main thread joins all event threads.
9. Cleanup and exit.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <time.h>

#define NUM_EVENTS 8
#define MAX_CONCURRENT_EVENTS 3

#define PROPOSAL_SPOTS 3
#define DECOR_KITS 3
#define AV_KITS 4

typedef struct {
    int id;
    int planner_id;
    int need_proposal;
    int need_decor;
    int need_av;
    int duration;
} event_t;

event_t events[NUM_EVENTS];
```

```
sem_t sem_proposal, sem_decor, sem_av;
sem_t concurrent_events_sem;
pthread_mutex_t log_mutex;
int active_events = 0;

void* event_thread(void* arg) {
    event_t* ev = (event_t*)arg;

    pthread_mutex_lock(&log_mutex);
    printf("[WAITING] Event %d (Planner %d) - needs: P=%d, D=%d, AV=%d,
dur=%ds\n",
           ev->id, ev->planner_id, ev->need_proposal,
           ev->need_decor, ev->need_av, ev->duration);
    pthread_mutex_unlock(&log_mutex);

    sem_wait(&concurrent_events_sem);

    pthread_mutex_lock(&log_mutex);
    active_events++;
    printf("[ACQUIRED] Event %d got slot! Active events: %d/%d\n",
           ev->id, active_events, MAX_CONCURRENT_EVENTS);
    pthread_mutex_unlock(&log_mutex);


    if (ev->need_proposal) {
        sem_wait(&sem_proposal);
        pthread_mutex_lock(&log_mutex);
        printf("  Event %d acquired proposal spot\n", ev->id);
        pthread_mutex_unlock(&log_mutex);
    }

    for (int d = 0; d < ev->need_decor; d++) {
        sem_wait(&sem_decor);
        pthread_mutex_lock(&log_mutex);
```

```
        printf("  Event %d acquired decor kit %d/%d\n", ev->id, d+1, ev-
>need_decor);
        pthread_mutex_unlock(&log_mutex);
    }

    for (int a = 0; a < ev->need_av; a++) {
        sem_wait(&sem_av);
        pthread_mutex_lock(&log_mutex);
        printf("  Event %d acquired AV kit %d/%d\n", ev->id, a+1, ev->need_av);
        pthread_mutex_unlock(&log_mutex);
    }



    pthread_mutex_lock(&log_mutex);
    printf("[RUNNING] Event %d (Planner %d) - All resources acquired! Executing
for %d seconds...\n",
           ev->id, ev->planner_id, ev->duration);
    pthread_mutex_unlock(&log_mutex);

    sleep(ev->duration);

    if (ev->need_proposal) {
        sem_post(&sem_proposal);
        pthread_mutex_lock(&log_mutex);
        printf("  Event %d released proposal spot\n", ev->id);
        pthread_mutex_unlock(&log_mutex);
    }

    for (int d = 0; d < ev->need_decor; d++) {
        sem_post(&sem_decor);
        pthread_mutex_lock(&log_mutex);
        printf("  Event %d released decor kit %d/%d\n", ev->id, d+1, ev-
>need_decor);
        pthread_mutex_unlock(&log_mutex);
    }
```

```
    for (int a = 0; a < ev->need_av; a++) {
        sem_post(&sem_av);
        pthread_mutex_lock(&log_mutex);
        printf("  Event %d released AV kit %d/%d\n", ev->id, a+1, ev->need_av);
        pthread_mutex_unlock(&log_mutex);
    }

    pthread_mutex_lock(&log_mutex);
    active_events--;
    printf("[COMPLETED] Event %d finished! Active events now: %d/%d\n",
           ev->id, active_events, MAX_CONCURRENT_EVENTS);
    pthread_mutex_unlock(&log_mutex);

    sem_post(&concurrent_events_sem);

    return NULL;
}

int main() {
    srand(time(NULL));
    pthread_t event_threads[NUM_EVENTS];

    sem_init(&sem_proposal, 0, PROPOSAL_SPOTS);
    sem_init(&sem_decor, 0, DECOR_KITS);
    sem_init(&sem_av, 0, AV_KITS);
    sem_init(&concurrent_events_sem, 0, MAX_CONCURRENT_EVENTS);
    pthread_mutex_init(&log_mutex, NULL);

    printf("============================================\n");
    printf("   EVENT-THREAD MANAGEMENT SYSTEM\n");
    printf("============================================\n\n");

    printf("SYSTEM CAPACITY:\n");
    printf(" Maximum concurrent events: %d\n", MAX_CONCURRENT_EVENTS);
```

```
    printf(" Available resources:\n");
    printf("  - Proposal spots: %d\n", PROPOSAL_SPOTS);
    printf("  - Decor kits: %d\n", DECOR_KITS);
    printf("  - AV kits: %d\n\n", AV_KITS);

    printf("GENERATING %d EVENTS:\n", NUM_EVENTS);
    printf("--------------------\n");

    for (int i = 0; i < NUM_EVENTS; i++) {
        events[i].id = i + 1;
        events[i].planner_id = 1 + rand() % 2
        events[i].need_proposal = rand() % 2;
        events[i].need_decor = 1 + rand() % 2;
        events[i].need_av = 1 + rand() % 2;
        events[i].duration = 1 + rand() % 2;
    }

    printf("\n==========================================\n");
    printf("   EXECUTION START - Only %d at a time!\n", MAX_CONCURRENT_EVENTS);
    printf("==========================================\n\n");

    for (int i = 0; i < NUM_EVENTS; i++) {
        pthread_create(&event_threads[i], NULL, event_thread, &events[i]);
        usleep(500000);
    }

    for (int i = 0; i < NUM_EVENTS; i++) {
        pthread_join(event_threads[i], NULL);
    }

    sem_destroy(&sem_proposal);
    sem_destroy(&sem_decor);
    sem_destroy(&sem_av);
    sem_destroy(&concurrent_events_sem);
    pthread_mutex_destroy(&log_mutex);
```

```
    printf("\n=========================================\n");
    printf("   ALL %d EVENTS COMPLETED SUCCESSFULLY!\n", NUM_EVENTS);
    printf("=========================================\n");

    return 0;
}
```

Output:

```
PS D:\7th semester\325\Project> cd  d:\7th semester\325\Project\  ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }
==========================================
   EVENT-THREAD MANAGEMENT SYSTEM
==========================================

SYSTEM CAPACITY:
 Maximum concurrent events: 3
 Available resources:
  - Proposal spots: 3
  - Decor kits: 3
  - AV kits: 4

GENERATING 8 EVENTS:
--------------------

==========================================
   EXECUTION START - Only 3 at a time!
==========================================

[WAITING] Event 1 (Planner 2) - needs: P=0, D=2, AV=1, dur=1s
[ACQUIRED] Event 1 got slot! Active events: 1/3
  Event 1 acquired decor kit 1/2
  Event 1 acquired decor kit 2/2
  Event 1 acquired AV kit 1/1
[RUNNING] Event 1 (Planner 2) - All resources acquired! Executing for 1 seconds...
[WAITING] Event 2 (Planner 2) - needs: P=0, D=1, AV=2, dur=1s
[ACQUIRED] Event 2 got slot! Active events: 2/3
  Event 2 acquired decor kit 1/1
  Event 2 acquired AV kit 1/2
  Event 2 acquired AV kit 2/2
[RUNNING] Event 2 (Planner 2) - All resources acquired! Executing for 1 seconds...
  Event 1 released decor kit 1/2
  Event 1 released decor kit 2/2
  Event 1 released AV kit 1/1
[COMPLETED] Event 1 finished! Active events now: 1/3
[WAITING] Event 3 (Planner 1) - needs: P=0, D=2, AV=2, dur=2s
[ACQUIRED] Event 3 got slot! Active events: 2/3
  Event 3 acquired decor kit 1/2
  Event 3 acquired decor kit 2/2
  Event 3 acquired AV kit 1/2
  Event 3 acquired AV kit 2/2
[RUNNING] Event 3 (Planner 1) - All resources acquired! Executing for 2 seconds...
  Event 2 released decor kit 1/1
  Event 2 released AV kit 1/2
  Event 2 released AV kit 2/2
[COMPLETED] Event 2 finished! Active events now: 1/3
[WAITING] Event 4 (Planner 1) - needs: P=0, D=2, AV=2, dur=1s
[ACQUIRED] Event 4 got slot! Active events: 2/3
  Event 4 acquired decor kit 1/2
[WAITING] Event 5 (Planner 1) - needs: P=0, D=1, AV=2, dur=1s
[ACQUIRED] Event 5 got slot! Active events: 3/3
[WAITING] Event 6 (Planner 2) - needs: P=0, D=1, AV=2, dur=2s
```

```
[WAITING] Event 6 (Planner 2) - needs: P=0, D=1, AV=2, dur=2s
  Event 3 released decor kit 1/2
  Event 3 released decor kit 2/2
  Event 3 released AV kit 1/2
  Event 3 released AV kit 2/2
[COMPLETED] Event 3 finished! Active events now: 2/3
  Event 5 acquired decor kit 1/1
  Event 5 acquired AV kit 1/2
  Event 5 acquired AV kit 2/2
[RUNNING] Event 5 (Planner 1) - All resources acquired! Executing for 1 seconds...
[ACQUIRED] Event 6 got slot! Active events: 3/3
  Event 4 acquired decor kit 2/2
  Event 4 acquired AV kit 1/2
  Event 4 acquired AV kit 2/2
[RUNNING] Event 4 (Planner 1) - All resources acquired! Executing for 1 seconds...
[WAITING] Event 7 (Planner 2) - needs: P=1, D=2, AV=1, dur=2s
[WAITING] Event 8 (Planner 2) - needs: P=1, D=1, AV=2, dur=1s
  Event 5 released decor kit 1/1
  Event 5 released AV kit 1/2
  Event 5 released AV kit 2/2
[COMPLETED] Event 5 finished! Active events now: 2/3
  Event 4 released decor kit 1/2
  Event 4 released decor kit 2/2
  Event 4 released AV kit 1/2
  Event 4 released AV kit 2/2
[COMPLETED] Event 4 finished! Active events now: 1/3
  Event 6 acquired decor kit 1/1
  Event 6 acquired AV kit 1/2
  Event 6 acquired AV kit 2/2
[RUNNING] Event 6 (Planner 2) - All resources acquired! Executing for 2 seconds...
[ACQUIRED] Event 8 got slot! Active events: 2/3
  Event 8 acquired proposal spot
  Event 8 acquired decor kit 1/1
  Event 8 acquired AV kit 1/2
  Event 8 acquired AV kit 2/2
[RUNNING] Event 8 (Planner 2) - All resources acquired! Executing for 1 seconds...
[ACQUIRED] Event 7 got slot! Active events: 3/3
  Event 7 acquired proposal spot
  Event 7 acquired decor kit 1/2
  Event 8 released proposal spot
  Event 8 released decor kit 1/1
  Event 8 released AV kit 1/2
  Event 8 released AV kit 2/2
[COMPLETED] Event 8 finished! Active events now: 2/3
  Event 7 acquired decor kit 2/2
  Event 7 acquired AV kit 1/1
[RUNNING] Event 7 (Planner 2) - All resources acquired! Executing for 2 seconds...
  Event 6 released decor kit 1/1
  Event 6 released AV kit 1/2
  Event 6 released AV kit 2/2
[COMPLETED] Event 6 finished! Active events now: 1/3
  Event 7 released proposal spot
```

```
  Event 4 released decor kit 2/2
  Event 4 released AV kit 1/2
  Event 4 released AV kit 2/2
[COMPLETED] Event 4 finished! Active events now: 1/3
  Event 6 acquired decor kit 1/1
  Event 6 acquired AV kit 1/2
  Event 6 acquired AV kit 2/2
[RUNNING] Event 6 (Planner 2) - All resources acquired! Executing for 2 seconds...
[ACQUIRED] Event 8 got slot! Active events: 2/3
  Event 8 acquired proposal spot
  Event 8 acquired decor kit 1/1
  Event 8 acquired AV kit 1/2
  Event 8 acquired AV kit 2/2
[RUNNING] Event 8 (Planner 2) - All resources acquired! Executing for 1 seconds...
[ACQUIRED] Event 7 got slot! Active events: 3/3
  Event 7 acquired proposal spot
  Event 7 acquired decor kit 1/2
  Event 8 released proposal spot
  Event 8 released decor kit 1/1
  Event 8 released AV kit 1/2
  Event 8 released AV kit 2/2
[COMPLETED] Event 8 finished! Active events now: 2/3
  Event 7 acquired decor kit 2/2
  Event 7 acquired AV kit 1/1
[RUNNING] Event 7 (Planner 2) - All resources acquired! Executing for 2 seconds...
  Event 6 released decor kit 1/1
  Event 6 released AV kit 1/2
  Event 6 released AV kit 2/2
[COMPLETED] Event 6 finished! Active events now: 1/3
  Event 7 released proposal spot
  Event 7 released decor kit 1/2
  Event 7 released decor kit 2/2
  Event 7 released AV kit 1/1
[COMPLETED] Event 7 finished! Active events now: 0/3

========================================
   ALL 8 EVENTS COMPLETED SUCCESSFULLY!
========================================
PS D:\7th semester\325\Project>
```

Table:

Components used	Purpose of using
Semaphore	Limits concurrent events and manages resource pools (proposal, decor, AV)
Mutex Lock	Protects shared log output and active event count from race conditions.
pthreads	Represents each event as a thread for concurrent execution.
Sleep	Simulates event duration and staggered thread creation

Conclusion:

This program successfully simulates “Ted’s Stories,” a romantic event planning company, using multi-threading and synchronization mechanisms. Semaphores control the number of concurrent events and resource availability, while mutex locks ensure thread-safe logging. Each event thread acquires necessary resources, executes, and releases them, demonstrating efficient concurrency management in a resource-constrained environment. The project highlights key OS concepts such as thread synchronization, resource allocation, and deadlock prevention in a real-world inspired scenario