EAST WEST UNIVERSITY

Department of Computer Science & Engineering

Semester: **Fall-2023**
Course Code: **CSE325**
Course Title: Operating System
Section: 01
Group: 07

Project Title (7): FIFA World Cup

Presented by:

Name:Kazi Minhajul Goni Sami
ID: 2021-2-60-020 (Roll-10)

Name: MD.Sajjad Hossain
ID: 2021-2-60-136 (Roll-21)

Name: Mir Azmain Wasif
ID: 2021-2-60-108(Section 2, Roll-17)

Presented to

Nawab Yousuf Ali
Professor
Department of Computer Science & Engineering
East West University

Date of Submission: 26/12/2023

Introduction

An operating system (OS) is software that manages computer hardware and software resources while also providing common functions to computer programs. Time-sharing operating systems plan tasks to make the most of the system's resources, and they may also contain accounting software for cost allocation of processor time, storage, printing, and other resources. Although application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it, the operating system acts as an intermediary between programs and the computer hardware for hardware functions such as input and output and memory allocation. From cellular phones and video game consoles to web servers and supercomputers, operating systems are found on many devices that incorporate a computer.
In our “burger buddies’ problem”, we will focus on the three main topics. Threads, process and semaphore.

Abstract

In our following project, we will implement and test a solution for the IPC (Inter Process Communication) problem of FIFA World Cup in which we will use semaphore and multi threads to execute our code to have a synchronization for the FIFA World Cup 2006, a fly-over has been constructed between the hotel where the teams are staying and the stadium. This fly-over will be used by the German team and the Italian team in the upcoming semifinal on Tuesday. A tram car is used to cross this fly-over, but it seats only four people, and must always carry a full load. We cannot put three Italians and one German in the same tramcar, because the Italians would be in majority and might try to intimidate the German. Similarly, we cannot put three Germans in the same tram car with one Italian. All other combinations are safe. In this program, we will mainly focus on three threads: function [German thread, Italian thread, and print line]. These three functions are related to each other with semaphore and created by threads. The following code will define some variables at the beginning of the program. But we can change it according to one’s choice.

Threads: Within a process, a thread is a path of execution. Multiple threads can exist in a process. The lightweight process is also known as a thread. By dividing a process into numerous threads, parallelism can be achieved. Multiple tabs in a browser, for example, can represent different threads. MS Word makes use of numerous threads: one to format the text, another to receive inputs, and so on. Below are some more advantages of multithreading.

Process: A process is essential for running software. The execution of a process must be done in a specific order. To put it another way, we write our computer programs in a text file, and when we run them, they turn into a process that completes all the duties specified in the program. A program can be separated into four components when it is put into memory and becomes a

process: stack, heap, text, and data. The diagram below depicts a simplified structure of a process in main memory.

Semaphore: Dijkstra proposed the semaphore in 1965, which is a very important technique for managing concurrent activities using a basic integer value called a semaphore. A semaphore is just an integer variable shared by many threads. In a multiprocessing context, this variable is utilized to solve the critical section problem and establish process synchronization.
There are two types of semaphores:

1. **Binary Semaphore –**
   This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.
2. **Counting Semaphore –**
   Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Similar Work:

- **Dining Philosophers' Problems**
- **Producer Consumer Problem**
- **Barber Shop Problem**

Proposed Work

1.Code Documentation:

- Add comprehensive code comments and documentation to explain the purpose and functioning of each function and data structure.
- Provide details about the synchronization mechanism used (mutexes and condition variables) and how they ensure safe tram boarding.

2.Error Handling and Input Validation:

- Implement error handling and input validation to handle unexpected inputs gracefully. For example, validate user input for the number of players to ensure it's greater than or equal to 1.

3.Modularization:

- Consider breaking down the code into smaller, modular functions to improve code readability and maintainability.
- Create separate headers and source files for functions and data structures used in the simulation.

4.Configuration Options:

- Allow for configuration options such as the tram capacity, the maximum number of players per team, and the sleep duration between actions. This flexibility can make the simulation more adaptable.

5.Logging and Output Enhancement:

- Enhance the logging and output mechanism by providing timestamps for events or logging to a file.
- Add more detailed information about the simulation progress, such as the current state of players waiting to board and the status of the tram.

6.Testing and Validation:

- Develop test cases to validate the correctness of the simulation under various scenarios, including edge cases.
- Implement automated tests to ensure that the code continues to function correctly as it evolves.

7.Concurrency Improvements:

- Explore potential performance optimizations for the code, such as minimizing mutex contention or fine-tuning the synchronization logic.

8.User Interface (Optional):

- If applicable, create a simple user interface or configuration file support for users to specify the number of players interactively.

9.Resource Management:

- Ensure proper resource management, such as cleaning up threads and resources after the simulation has been completed.

10.Portability and Compatibility:

- Check the code for portability across different operating systems and platforms to ensure it runs consistently.

11.Version Control and Collaboration:

- Use version control systems (e.g., Git) to track changes and collaborate with others on the codebase.

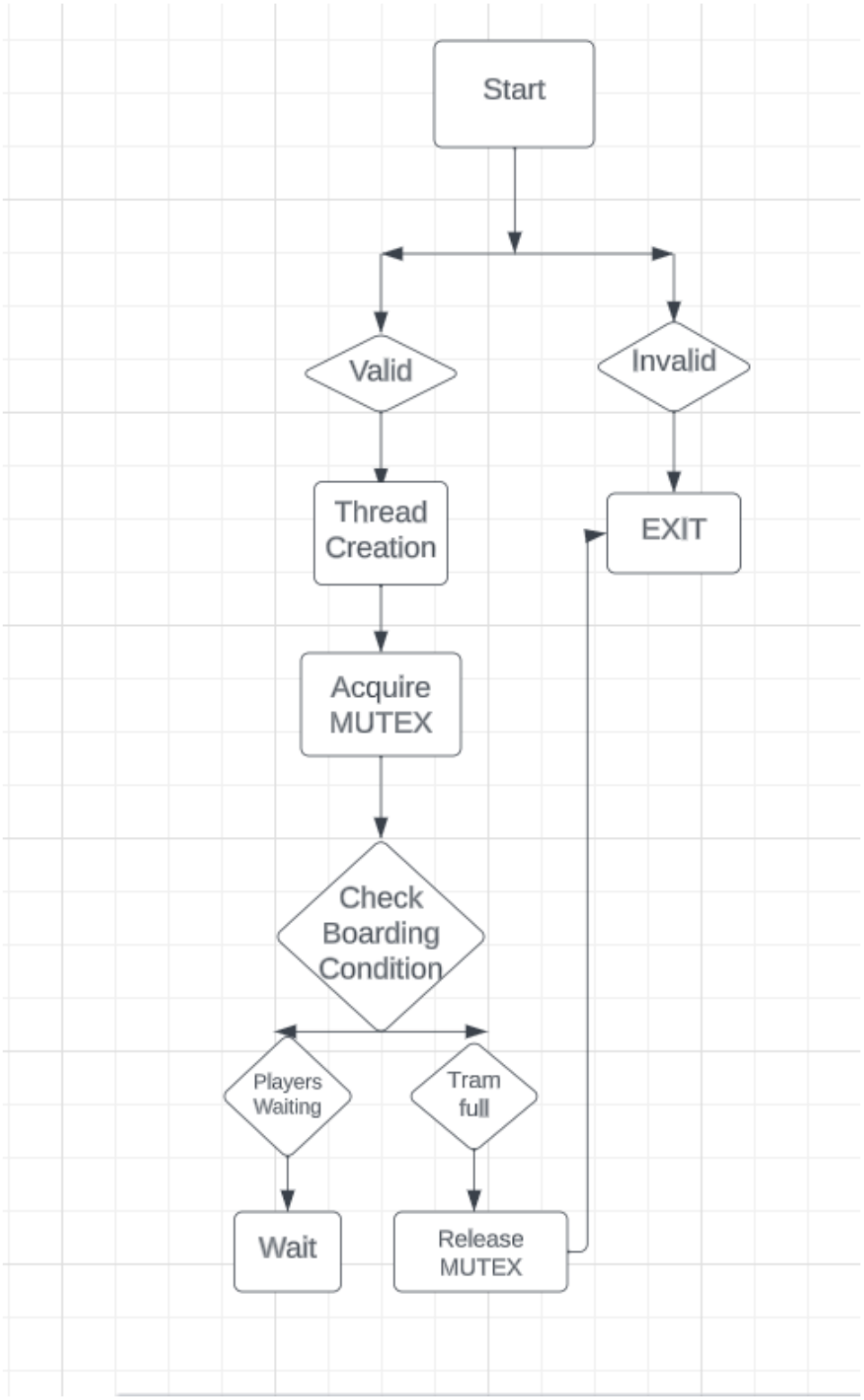
12.Security Considerations:

- If the code is used in a production environment, consider potential security vulnerabilities and implement necessary security measures.

13.Performance Profiling:

- Profile the code's performance using appropriate tools to identify bottlenecks or areas for optimization.

Flowchart:


Start
Valid
Invalid
Thread Creation
EXIT
Acquire MUTEX
Check Boarding Condition
Players Waiting
Tram full
Wait
Release MUTEX

C Program Code:

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

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_tram = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_board = PTHREAD_COND_INITIALIZER;

int num_germans_waiting = 0;
int num_italians_waiting = 0;
int num_boarded = 0;

void print_line(const char *message) {
    printf("%s\n", message);
    fflush(stdout);
    usleep(500000);
}

void *german_thread(void *arg) {
    int id = *(int *)arg;
    char message[50];
    sprintf(message, "German player %d has arrived.", id);
    print_line(message);

    pthread_mutex_lock(&mutex);
    num_germans_waiting++;

    while (num_boarded == 4 || (num_germans_waiting > 0 && num_italians_waiting >= 3)) {
        pthread_cond_wait(&cond_board, &mutex);
    }

    num_boarded++;
    if (num_boarded == 4) {
        sprintf(message, "Tram departing with Germans: %d, Italians: %d.", num_germans_waiting, num_italians_waiting);
        print_line(message);
        num_germans_waiting = 0;
        num_italians_waiting = 0;
        num_boarded = 0;
        pthread_cond_broadcast(&cond_tram);
    }

    pthread_mutex_unlock(&mutex);
```

```c
    pthread_mutex_unlock(&mutex);

    sprintf(message, "German player %d has crossed the fly-over.", id);
    print_line(message);

    return NULL;
}

void *italian_thread(void *arg) {
    int id = *(int *)arg;
    char message[50];
    sprintf(message, "Italian player %d has arrived.", id);
    print_line(message);

    pthread_mutex_lock(&mutex);
    num_italians_waiting++;

    while (num_boarded == 4 || (num_italians_waiting > 0 && num_germans_waiting >= 3)) {
        pthread_cond_wait(&cond_board, &mutex);
    }

    num_boarded++;
    if (num_boarded == 4) {
        sprintf(message, "Tram departing with Germans: %d, Italians: %d.", num_germans_waiting, num_italians_waiting);
        print_line(message);
        num_germans_waiting = 0;
        num_italians_waiting = 0;
        num_boarded = 0;
        pthread_cond_broadcast(&cond_tram);
    }

    pthread_mutex_unlock(&mutex);

    sprintf(message, "Italian player %d has crossed the fly-over.", id);
    print_line(message);

    return NULL;
}

int main() {
    int num_germans, num_italians;

    printf("Enter the number of German players: ");
```

```c
int main() {
    int num_germans, num_italians;

    printf("Enter the number of German players: ");
    scanf("%d", &num_germans);

    printf("Enter the number of Italian players: ");
    scanf("%d", &num_italians);

    if (num_germans < 1 || num_italians < 1) {
        printf("Invalid input. There must be at least 1 German and 1 Italian player.\n");
        return 1;
    }

    pthread_t german_threads[num_germans];
    pthread_t italian_threads[num_italians];
    int german_ids[num_germans];
    int italian_ids[num_italians];

    printf("FIFA World Cup Tram Simulation\n");

    for (int i = 0; i < num_germans; i++) {
        german_ids[i] = i + 1;
        pthread_create(&german_threads[i], NULL, german_thread, &german_ids[i]);
    }

    for (int i = 0; i < num_italians; i++) {
        italian_ids[i] = i + 1;
        pthread_create(&italian_threads[i], NULL, italian_thread, &italian_ids[i]);
    }

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

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

    printf("Tram departing with Germans: %d, Italians: %d.\n", num_germans_waiting, num_italians_waiting);
    printf("All players have crossed the fly-over. Exiting simulation.\n");

    return 0;
}
```

Output:

```
sami@linux:~/Desktop$ gcc fifa.c -o fifa
sami@linux:~/Desktop$ ./fifa out
Enter the number of German players: 15
Enter the number of Italian players: 15
FIFA World Cup Tram Simulation
German player 2 has arrived.
German player 1 has arrived.
German player 4 has arrived.
German player 5 has arrived.
German player 6 has arrived.
German player 3 has arrived.
German player 7 has arrived.
German player 8 has arrived.
German player 9 has arrived.
German player 10 has arrived.
German player 11 has arrived.
German player 12 has arrived.
German player 14 has arrived.
German player 15 has arrived.
Italian player 1 has arrived.
Italian player 2 has arrived.
Italian player 3 has arrived.
Italian player 4 has arrived.
Italian player 5 has arrived.
German player 13 has arrived.
Italian player 6 has arrived.
Italian player 7 has arrived.
Italian player 9 has arrived.
Italian player 8 has arrived.
Italian player 10 has arrived.
Italian player 15 has arrived.
Italian player 13 has arrived.
Italian player 14 has arrived.
Italian player 11 has arrived.
Italian player 12 has arrived.
```

```
Italian player 12 has arrived.
German player 4 has crossed the fly-over.
German player 5 has crossed the fly-over.
German player 6 has crossed the fly-over.
Tram departing with Germans: 4, Italians: 0.
German player 3 has crossed the fly-over.
Italian player 14 has crossed the fly-over.
German player 2 has crossed the fly-over.
German player 1 has crossed the fly-over.
Tram departing with Germans: 2, Italians: 2.
Italian player 13 has crossed the fly-over.
Italian player 12 has crossed the fly-over.
Tram departing with Germans: 0, Italians: 4.
Italian player 8 has crossed the fly-over.
Italian player 11 has crossed the fly-over.
Italian player 9 has crossed the fly-over.
Italian player 7 has crossed the fly-over.
Italian player 6 has crossed the fly-over.
Italian player 4 has crossed the fly-over.
Tram departing with Germans: 0, Italians: 4.
Italian player 3 has crossed the fly-over.
German player 15 has crossed the fly-over.
German player 14 has crossed the fly-over.
German player 12 has crossed the fly-over.
Tram departing with Germans: 4, Italians: 0.
German player 11 has crossed the fly-over.
German player 10 has crossed the fly-over.
German player 9 has crossed the fly-over.
German player 8 has crossed the fly-over.
Tram departing with Germans: 4, Italians: 0.
German player 7 has crossed the fly-over.
Italian player 10 has crossed the fly-over.
Italian player 15 has crossed the fly-over.
Italian player 1 has crossed the fly-over.
Tram departing with Germans: 0, Italians: 4.
Italian player 2 has crossed the fly-over.
Italian player 5 has crossed the fly-over.
German player 13 has crossed the fly-over.
Tram departing with Germans: 1, Italians: 1.
All players have crossed the fly-over. Exiting simulation.
sami@linux:~/Desktop$
```

Conclusion:

Context switching is the process of storing a process's context or state such that it can be reloaded, and execution continued from the same point as before. A "Context Switch" is the act of transitioning from one process to another. A computer system often has multiple duties to complete. So, if one activity requires some I/O, we want to start the I/O operation before moving on to the next process. We'll go through it again later. We should pick up where we left off when we return to a process. For all intents and purposes, this process should never be aware of the switch, and it should appear as if it were the only one in the system.

The End