Operating System

Project

Operating system for the multi-agent system

Submitted by:

I210844 (Abdullah Chaudhary)

I210492 (Zainab Ali Khan)

I210420 (Aqib Ansari)

Submitted To:

Javeria Zia

Table of Contents

Introduction

The design and implementation of an operating system for a multi-agent system involving robots in a confined space poses interesting challenges and opportunities. In this scenario, 50 robots are randomly placed within a 100 by 100 room, with the goal of collectively estimating the width of a single exit. To achieve this, the operating system must facilitate communication, synchronization, and coordination among the individual robot processes.

Key aspects of the operating system design include process synchronization, memory management, inter-process communication (IPC), thread management, and process scheduling. These components are essential for ensuring efficient and reliable operation in a multi-agent system where robots must collaborate to achieve a common objective.

Project Requirements

Process Synchronization:

Synchronization primitives such as semaphores or locks will be utilized to ensure exclusive access to shared resources. This prevents race conditions and ensures that only one process accesses critical resources at a time.

Memory Management:

The operating system will provide mechanisms for allocating and deallocating memory for processes and threads. Each process and thread will have its own memory space to prevent conflicts and ensure data integrity.

Inter-Process Communication (IPC):

A robust IPC mechanism is crucial for facilitating seamless information exchange between robot processes. This will involve functions and methods for communication, allowing robots to share their estimated exit widths.

Thread Management:

Efficient thread management using the pthread library will be implemented. This includes creating, managing, and synchronizing threads to enable concurrent execution and optimal resource allocation.

Process Scheduling:

The operating system will employ an efficient process scheduling mechanism. This ensures that processes and threads are scheduled optimally to maintain smooth and efficient execution, considering the time-sensitive nature of the robot coordination.

By incorporating these elements into the operating system design, we aim to create a reliable and efficient environment for the robots to collaboratively estimate the exit width. The success of the system will depend on the effective coordination of processes, threads, and communication channels, emphasizing the principles of concurrency and synchronization.

Logic Approach

- **Initialize Shared Memory and Message Queue:**
- Generate a key for creating or accessing shared memory and a message queue.
- Create or access the shared memory segment for messages.
- Initialize variables for room dimension, exit width, and the number of robots.
- **Generate Robot Positions:**
- Generate random positions for 50 robots within the 100 by 100 room.
- **Calculate and Send Estimated Width:**
- Create a child process using **fork()** to handle the calculations.
- For each robot:
- Calculate the distance from the exit and estimate the exit width based on the provided conditions.
- Print the robot's estimated width.
- Calculate and print the estimated width.
- Send the robot's estimated width, robot number, and message type to the message queue.
- **Read Data from Shared Memory:**
- In the parent process:
- Wait for each child process to finish.
- Read data from the message queue for each robot.
- Print robot number, estimated width, true exit width, and the difference between estimated and true width.
- **Thread Creation for Neighbor Communication:**
- Initialize a semaphore for synchronizing access to the total width variable.
- Create an array of threads to represent each robot.
- For each robot:
- Create a thread to calculate the average width based on communication with neighbors.
- Pass the robot's ID to the thread.
- **Calculate Average Exit Width:**
- In each thread:

- Acquire the semaphore to ensure exclusive access to the total width variable.
- Calculate the average width considering neighbors' estimates.
- Update the total width variable.
- Release the semaphore.
- **Join Threads and Print Results:**
- Join all threads to wait for their completion.
- Calculate the final average exit width by dividing the total width by the number of robots.
- Print the average exit width calculated by all robots and the true exit width.
- **Cleanup:**
- Destroy the semaphore.
- Return 0 to indicate successful execution.

This algorithm outlines the major steps and flow of the provided code, including processes, threads, and communication mechanisms. It aims to estimate the exit width collaboratively among robots and demonstrate the use of shared memory, message queues, and synchronization mechanisms.

Algorithm

1. Initialize Shared Memory and Message Queue:

- key = generate_key("/tmp", 'D')

- msgid = create_or_access_msg_queue(key)

2. Generate Robot Positions:

- room = create_100_by_100_room()

- exitWidth = generate_random_exit_width()

- generate_random_robot_positions()

3. Calculate and Send Estimated Width (in a loop for each robot):

- for i in 1 to 50:

  - create_child_process()

  - result = calculate_distance_from_exit(robots[i], exitWidth)

  - estimatedWidth = (result * exitWidth) / 100

  - print("Robot i Estimated Width:", estimatedWidth)

  - calculate_and_send_to_shared_memory(i, estimatedWidth)

4. Read Data from Shared Memory (in the parent process):

- for i in 1 to 50:
  - wait_for_child_process_completion()
  - read_from_shared_memory_and_print_data()

5. Thread Creation for Neighbor Communication:

- totalWidthSemaphore = initialize_semaphore(1)
- for i in 1 to 50:
  - create_thread(calculateAverageWidthNeighbor, i)

6. Calculate Average Exit Width (in each thread):

- for i in 1 to 50:
  - acquire_semaphore(totalWidthSemaphore)
  - calculate_average_width_with_neighbors(i)
  - update_total_width_variable()
  - release_semaphore(totalWidthSemaphore)

7. Join Threads and Print Results:

- for i in 1 to 50:
  - join_thread(threads[i])
- calculate_final_average_exit_width()
- print("Average exit width calculated by all robots:", totalWidth / 50)
- print("True exit width:", exitWidth)

8. Cleanup:

- destroy_semaphore(totalWidthSemaphore)
- return 0

Code

```
#include <iostream>
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <pthread.h>
#include <cstdlib>
#include <ctime>
#include <sys/shm.h>
#include <semaphore.h>
#include <cmath>
#include <sys/wait.h>
using namespace std;

// Forward declaration
struct robot;

int exitWidth = 0;          //True exit
float totalWidth = 0;      //Robots will estimate

// Function to calculate distance
double calculateDistance(const robot &r1, const robot &r2);

// Message structure
struct Message {
   long messageType;      // Message type (can be used to distinguish messages)
   int robotNumber;       // Number of the robot
   float estimatedWidth;  // Estimated width information
};
```

```
// Robot data
struct robot {
   int x;
   int y;
   float exitWidthEstimation;

   robot()
   {
      x = 0;
      y = 0;
      exitWidthEstimation = 0;
   }
};

// Creating 50 robots
robot robots[50]; // Creating 50 robots.

// Function to calculate distance
double calculateDistance(const robot &r1, const robot &r2)
{
   return sqrt(pow(r2.x - r1.x, 2) + pow(r2.y - r1.y, 2));
}

double calculateDistanceFromTheExit(const robot &r1, int exitWidth)
{
   double robotPosition = sqrt(pow(r1.x, 2) + pow(r1.y, 2));
   double distanceFromExit = abs(exitWidth - robotPosition);
   int result = 0;
   if (distanceFromExit <= 5)
```

```
    {
        result = rand() % 10 + 95;
        return result;
    }
    if (distanceFromExit <= 10)
    {
        result = rand() % 24 + 88;
        return result;
    }
    if (distanceFromExit <= 15)
    {
        result = rand() % 30 + 80;
        return result;
    }
    if (distanceFromExit <= 25)
    {
        result = rand() % 60 + 70;
        return result;
    }
    if (distanceFromExit <= 35)
    {
        result = rand() % 80 + 60;
        return result;
    }
    result = rand() % 100 + 50;
    return result;
}

bool areNeighbors(const robot &robot1, const robot &robot2, double maxDistance)
```

```
{
    double distance = calculateDistance(robot1, robot2);
    return distance <= maxDistance;
}

sem_t totalWidthSemaphore;
void *calculateAverageWidthNeighbor(void *arg)
{
    int i = *((int *)arg);
    int count = 1;
    // Use semaphore while communication between robots
    sem_wait(&totalWidthSemaphore);
    for (int j = 0; j < 50; j++)
    {
        if (i != j && areNeighbors(robots[i], robots[j], 5))
        {
            count++;
            robots[i].exitWidthEstimation += robots[j].exitWidthEstimation;
        }
    }


    robots[i].exitWidthEstimation /= count;
    totalWidth += robots[i].exitWidthEstimation;
    sem_post(&totalWidthSemaphore);

    pthread_exit(NULL);
}
```

```
int main()
{
   int shmid;      //Shared memory segment
   // Program
   srand(time(0));
   int dimension = 100;
   int room[dimension][dimension]; // Creating a 100 by 100 room.
   exitWidth = rand() % 11 + 16; //16 to 26
   int numRobots = 50;

   // Giving random placement to robots in the room.
   for (int i = 0; i < 50; i++)
   {
      robots[i].x = rand() % 99 + 1;
      robots[i].y = rand() % 99 + 1;
   }

   // Create or access the shared memory segment for messages
   key_t key = ftok("/tmp", 'D');
   int msgid = msgget(key, 0666 | IPC_CREAT);

   // Sending data to shared memory
   for (int i = 0; i < 50; ++i)
   {
          pid_t childPid = fork();

      if (childPid == -1) {
         // Handle error
         cout << "Error in creating process" << endl;
```

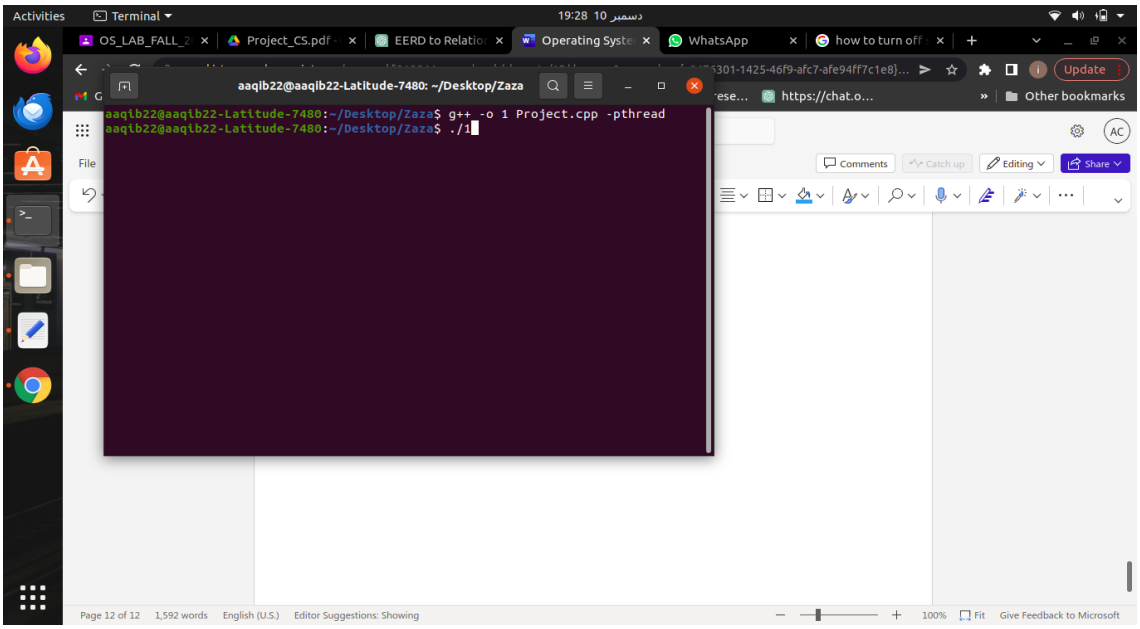
```
        return 1;
    }

    if (childPid == 0) {
    double result = calculateDistanceFromTheExit(robots[i], exitWidth);
    float estimatedWidth = (result * exitWidth) / 100;
    robots[i].exitWidthEstimation = estimatedWidth;

    cout << "Robot " << i + 1 << " Estimated Width: " << robots[i].exitWidthEstimation << endl;
    cout << "Sending data to shared memory." << endl;
    Message msg;
    msg.messageType = i + 1; // Use i + 1 as the message type
    msg.robotNumber = i + 1;
    msg.estimatedWidth = robots[i].exitWidthEstimation;
    msgsnd(msgid, &msg, sizeof(msg.robotNumber) + sizeof(msg.estimatedWidth), 0);
    }
    else{
    wait(NULL);
    exit(0);
    }
}

// Close the shared memory segment after writing
// shmctl(shmid, IPC_RMID, NULL);

// Reopen the shared memory segment for writing
// shmid = shmget(key, sizeof(Message), 0666 | IPC_CREAT);

// Reading data from the shared memory
```

```
   float robotNum[50], width[50];
   for (int i = 0; i < 50; i++)
   {
      Message receiveMsg;
      msgrcv(msgid, &receiveMsg, sizeof(receiveMsg.robotNumber) +
sizeof(receiveMsg.estimatedWidth), i + 1, 0);

      robotNum[i] = receiveMsg.robotNumber;
      width[i] = receiveMsg.estimatedWidth;
   }

   sem_init(&totalWidthSemaphore, 0, 1); // Initialize the semaphore

   pthread_t threads[50];
   //Neighbors sharing data with each other through threads
   for (int i = 0; i < 50; i++)
   {
      int *threadId = static_cast<int *>(malloc(sizeof(int)));
      *threadId = i;
      pthread_create(&threads[i], NULL, calculateAverageWidthNeighbor, (void *)threadId);
   }

   // Join threads
   for (int i = 0; i < 50; i++)
   {
      pthread_join(threads[i], NULL);
   }
   for (int i=0; i<50; i++){
```

```
        cout <<"Robot: " <<robotNum[i] << " " <<" Approximated value: " <<
robots[i].exitWidthEstimation <<" True Exit Width: "<<exitWidth<<" Difference:
"<<robots[i].exitWidthEstimation - exitWidth <<endl;

   }


   totalWidth /= numRobots;

   cout << "Average exit width calculated by all the robots is: " << totalWidth << ".\nWhile the truth
value is: " << exitWidth << endl;


   cout << endl;

   sem_destroy(&totalWidthSemaphore); // Destroy the semaphore

   return 0;

}
```

How to Run

- g++ -o 1 Project.cpp **-pthread**
- ./1



Output

aaqib22@aaqib22-Latitude-7480:~/Desktop/Zaza$ g++ -o 1 Project.cpp -pthread

aaqib22@aaqib22-Latitude-7480:~/Desktop/Zaza$ ./1

Output:

Robot 1 Estimated Width: 22.56

Sending data to shared memory.

Robot 2 Estimated Width: 21.6

Sending data to shared memory.

Robot 3 Estimated Width: 21.84

Sending data to shared memory.

Robot 4 Estimated Width: 24

Sending data to shared memory.

Robot 5 Estimated Width: 17.28

Sending data to shared memory.

Robot 6 Estimated Width: 17.28

Sending data to shared memory.

Robot 7 Estimated Width: 23.76

Sending data to shared memory.

Robot 8 Estimated Width: 30.48

Sending data to shared memory.

Robot 9 Estimated Width: 34.8

Sending data to shared memory.

Robot 10 Estimated Width: 26.88

Sending data to shared memory.

Robot 11 Estimated Width: 23.76

Sending data to shared memory.

Robot 12 Estimated Width: 34.32

Sending data to shared memory.

Robot 13 Estimated Width: 15.84

Sending data to shared memory.

Robot 14 Estimated Width: 26.4

Sending data to shared memory.

Robot 15 Estimated Width: 29.04

Sending data to shared memory.

Robot 16 Estimated Width: 27.84

Sending data to shared memory.

Robot 17 Estimated Width: 22.8

Sending data to shared memory.

Robot 18 Estimated Width: 31.92

Sending data to shared memory.

Robot 19 Estimated Width: 25.2

Sending data to shared memory.

Robot 20 Estimated Width: 19.2

Sending data to shared memory.

Robot 21 Estimated Width: 26.16

Sending data to shared memory.

Robot 22 Estimated Width: 31.44

Sending data to shared memory.

Robot 23 Estimated Width: 30.72

Sending data to shared memory.

Robot 24 Estimated Width: 27.6

Sending data to shared memory.

Robot 25 Estimated Width: 13.68

Sending data to shared memory.

Robot 26 Estimated Width: 14.88

Sending data to shared memory.

Robot 27 Estimated Width: 18.24

Sending data to shared memory.

Robot 28 Estimated Width: 24.48

Sending data to shared memory.

Robot 29 Estimated Width: 29.52

Sending data to shared memory.

Robot 30 Estimated Width: 17.52

Sending data to shared memory.

Robot 31 Estimated Width: 24.48

Sending data to shared memory.

Robot 32 Estimated Width: 23.76

Sending data to shared memory.

Robot 33 Estimated Width: 27.12

Sending data to shared memory.

Robot 34 Estimated Width: 15.6

Sending data to shared memory.

Robot 35 Estimated Width: 31.44

Sending data to shared memory.

Robot 36 Estimated Width: 32.4

Sending data to shared memory.

Robot 37 Estimated Width: 25.68

Sending data to shared memory.

Robot 38 Estimated Width: 26.4

Sending data to shared memory.

Robot 39 Estimated Width: 24.48

Sending data to shared memory.

Robot 40 Estimated Width: 17.52

Sending data to shared memory.

Robot 41 Estimated Width: 22.8

Sending data to shared memory.

Robot 42 Estimated Width: 15.84

Sending data to shared memory.

Robot 43 Estimated Width: 35.52

Sending data to shared memory.

Robot 44 Estimated Width: 15.12

Sending data to shared memory.

Robot 45 Estimated Width: 30.24

Sending data to shared memory.

Robot 46 Estimated Width: 28.56

Sending data to shared memory.

Robot 47 Estimated Width: 21.36

Sending data to shared memory.

Robot 48 Estimated Width: 31.92

Sending data to shared memory.

Robot 49 Estimated Width: 24.48

Sending data to shared memory.

Robot 50 Estimated Width: 23.28

Sending data to shared memory.

Robot: 1 Approximated value: 22.68 True Exit Width: 24 Difference: -1.32

Robot: 2 Approximated value: 21.6 True Exit Width: 24 Difference: -2.4

Robot: 3 Approximated value: 21.84 True Exit Width: 24 Difference: -2.16

Robot: 4 Approximated value: 23.84 True Exit Width: 24 Difference: -0.160002

Robot: 5 Approximated value: 17.28 True Exit Width: 24 Difference: -6.72

Robot: 6 Approximated value: 17.28 True Exit Width: 24 Difference: -6.72

Robot: 7 Approximated value: 19.32 True Exit Width: 24 Difference: -4.68

Robot: 8 Approximated value: 33 True Exit Width: 24 Difference: 9

Robot: 9 Approximated value: 34.8 True Exit Width: 24 Difference: 10.8

Robot: 10 Approximated value: 26.88 True Exit Width: 24 Difference: 2.88

Robot: 11 Approximated value: 21 True Exit Width: 24 Difference: -3

Robot: 12 Approximated value: 34.32 True Exit Width: 24 Difference: 10.32

Robot: 13 Approximated value: 15.84 True Exit Width: 24 Difference: -8.16

Robot: 14 Approximated value: 27.96 True Exit Width: 24 Difference: 3.96

Robot: 15 Approximated value: 29.04 True Exit Width: 24 Difference: 5.04

Robot: 16 Approximated value: 27.84 True Exit Width: 24 Difference: 3.84

Robot: 17 Approximated value: 22.74 True Exit Width: 24 Difference: -1.26

Robot: 18 Approximated value: 31.92 True Exit Width: 24 Difference: 7.92

Robot: 19 Approximated value: 25.2 True Exit Width: 24 Difference: 1.2

Robot: 20 Approximated value: 21.84 True Exit Width: 24 Difference: -2.16

Robot: 21 Approximated value: 26.16 True Exit Width: 24 Difference: 2.16

Robot: 22 Approximated value: 31.44 True Exit Width: 24 Difference: 7.44

Robot: 23 Approximated value: 30.72 True Exit Width: 24 Difference: 6.72

Robot: 24 Approximated value: 27.6 True Exit Width: 24 Difference: 3.6

Robot: 25 Approximated value: 13.68 True Exit Width: 24 Difference: -10.32

Robot: 26 Approximated value: 17.1 True Exit Width: 24 Difference: -6.9

Robot: 27 Approximated value: 19.62 True Exit Width: 24 Difference: -4.38

Robot: 28 Approximated value: 24.48 True Exit Width: 24 Difference: 0.48

Robot: 29 Approximated value: 28.74 True Exit Width: 24 Difference: 4.74

Robot: 30 Approximated value: 17.52 True Exit Width: 24 Difference: -6.48

Robot: 31 Approximated value: 20.16 True Exit Width: 24 Difference: -3.84

Robot: 32 Approximated value: 23.76 True Exit Width: 24 Difference: -0.24

Robot: 33 Approximated value: 27.12 True Exit Width: 24 Difference: 3.12

Robot: 34 Approximated value: 19.72 True Exit Width: 24 Difference: -4.28

Robot: 35 Approximated value: 31.44 True Exit Width: 24 Difference: 7.44

Robot: 36 Approximated value: 32.4 True Exit Width: 24 Difference: 8.4

Robot: 37 Approximated value: 25.68 True Exit Width: 24 Difference: 1.68

Robot: 38 Approximated value: 26.4 True Exit Width: 24 Difference: 2.4

Robot: 39 Approximated value: 23.16 True Exit Width: 24 Difference: -0.84

Robot: 40 Approximated value: 17.52 True Exit Width: 24 Difference: -6.48

Robot: 41 Approximated value: 22.8 True Exit Width: 24 Difference: -1.2

Robot: 42 Approximated value: 18 True Exit Width: 24 Difference: -6

Robot: 43 Approximated value: 34.26 True Exit Width: 24 Difference: 10.26

Robot: 44 Approximated value: 21.84 True Exit Width: 24 Difference: -2.16

Robot: 45 Approximated value: 30.24 True Exit Width: 24 Difference: 6.24

Robot: 46 Approximated value: 25.2 True Exit Width: 24 Difference: 1.2

Robot: 47 Approximated value: 21.36 True Exit Width: 24 Difference: -2.64

Robot: 48 Approximated value: 27.88 True Exit Width: 24 Difference: 3.88

Robot: 49 Approximated value: 24.48 True Exit Width: 24 Difference: 0.48

Robot: 50 Approximated value: 23.28 True Exit Width: 24 Difference: -0.719999

Average exit width calculated by all the robots is: 24.5996.

While the truth value is: 24