**Operating Systems**

**Project Report**

**Multi Agent Robot System**

Course Instructor: **Dr. Ahmad Din**

Submission Date: **7th December, 2023**

**Team Members**

Sarim Aeyzaz (i21-0328)  
Qasim Saeed (i21-0352)

Ehtsham Walidad (i21-0260)

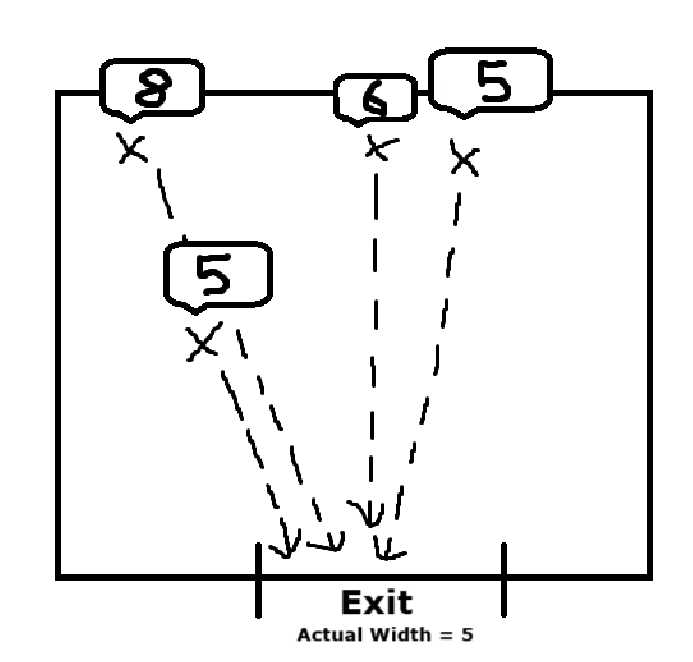
Hafiz Hammad Ahmed (i21-0343)

# **Table of Contents**

1. Problem Explanation
2. Flowchart
3. Step – by – Step Solution Explanation
4. Project Requirement Criteria
5. Output
6. Reflect

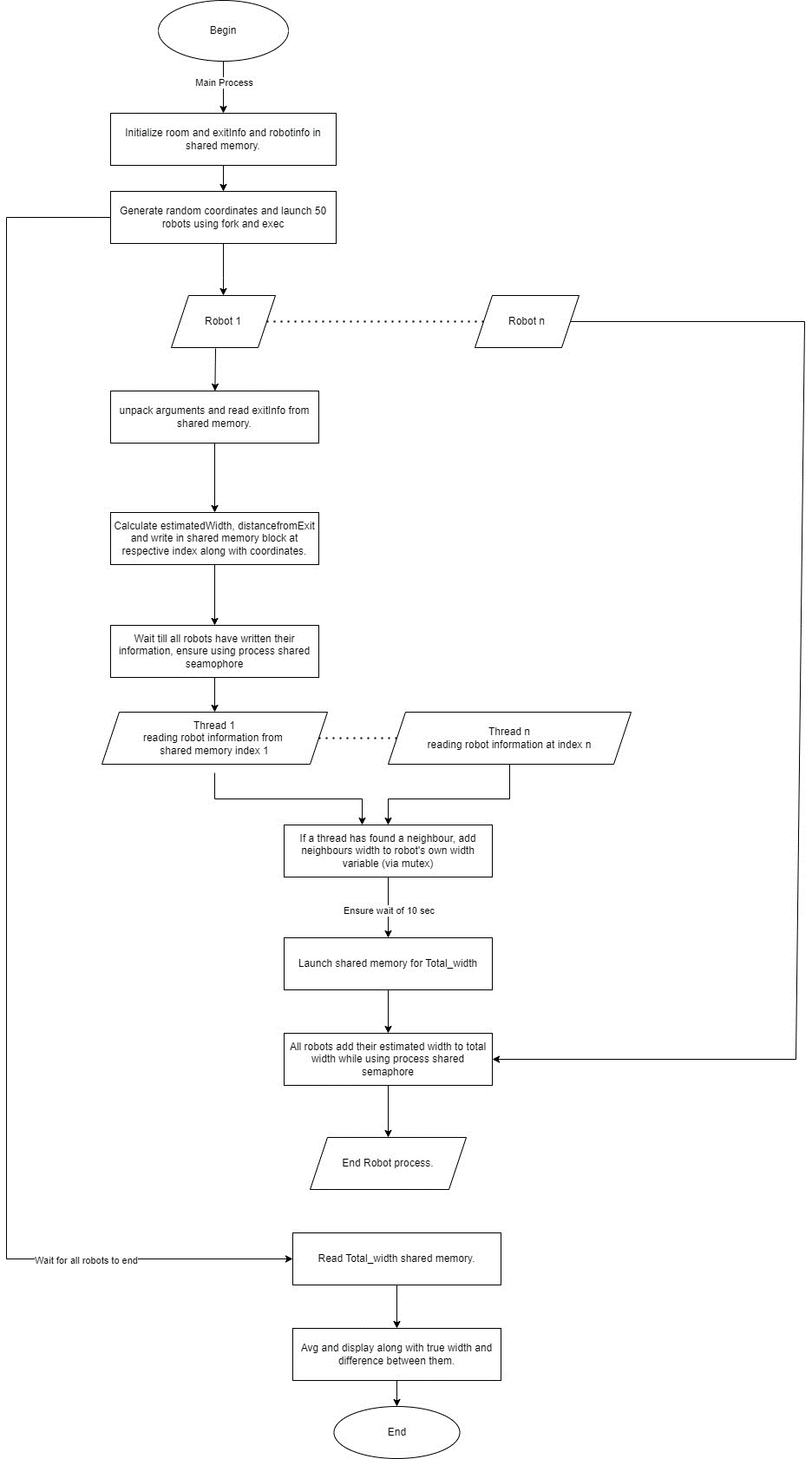
# **Problem Explanation**

There are 50 robots in a 100x100 room with an exit ranging from 16 units to 26 units. Every robot has a line of sight to the distance and the robot estimates the width of the exit based on how far away it is from it. After estimating its width, it also gathers the width estimate of its neighbours (those which are less than 5 units from the Robot) and averages the width. Finally, all robots send their average width to a global variable where all of the robot’s widths are averaged and compared with the true width of the exit.



# **Flowchart**

A flowchart showcasing our solution is available on the next page.



# **Step – By – Step Solution Explanation**

## Shared Memory Creation

**Step #1:** Create shared memory section storing exit information (exitInfo), store the exit’s X coordinate, Y coordinate and generated width onto this shared memory and detach it.

**Step #2:** Create shared memory section storing robot information (robotInfo) but during the initialization step, flush the shared memory with -1’s everywhere and detach it.

**Step #3:** Create shared memory section storing the final estimated width (estimatedGlobalWidth), initialize this as 0 and detach it.

## Semaphore and Process Creation

**Step #4:** Create two semaphores (globalWidthSemaphore and waitTill50). Initialize each semaphore as 1 and make them shareable between processes.

**Step #5:** Use fork to create a child process for each robot and each process will generate its own ID, (x,y) coordinate and run a file called robot.cpp. The ID, coordinates, along with totalRobot count and verbose are passed as arguments for the file.

## Robot File

**Step #6:** Each robot notes down the start time and target time (10 seconds from creation)

**Step #7:** Each robot opens the exitInfo shared memory and robotInfo shared memory. Afterwards, each robot would then store its own information (from the arguments), distance calculated (with information from exitInfo), estimated width (calculated via function) and store all of this information on its own section in the shared memory (this section is identified by the robot ID given from the file’s arguments.

**Note (How the accuracy function works):**

The accuracy requirements were that if a robot is 5 units or less, it will have an accuracy of 95% or greater. If the robot is between 5 and 10 units, the accuracy will range from 88% to 95% and so on.

To solve this, we divided distances into units. 1 unit = 5 blocks away from exit. For each unit being away from the exit, find the minimum accuracy range and the maximum accuracy range. Select an accuracy value from this range and calculate the reduction based on distance (Max -10 to 10 units of deviation from True width). Add or subtract this reduction on the true width to return the estimated width.

**Step #8:** Each robot will wait at a place until all other robots are done with their information uploading (This is done to synchronize them to make sure no robot would be reading -1 -1 -1 -1 information from shared memory).

**Step #9:** Each robot launches a thread containing ID of another robot on the board. Within this thread, the robot access information about its neighbor from the shared memory and calculates Euclidean distance. If the distance is less than 5 units then using a mutex, it will add the neighbors estimated width to its own width.

**Step #10:** After each thread is finished and is joined, each robot will calculate the average width and wait until the initial 10 seconds have passed (target time).

**Step #11:** When 10 seconds have passed, all robots will add their own width to the estimatedGlobalWidth shared memory while ensuring no race conditions through process-shared semaphore.

## Back to Main File

**Step #12:** As such, the parent process waits for all of the robot processes to finish. When it is done, it will open the estimatedGlobalWidth shared memory, take out the total width value from all of the robots, divide it by total number of robots and output the True Width of the exit, this averaged estimated width of all robots and the difference between them in the terminal

# Project Requirement Criteria

**Process Synchronization:** For resources where multiple writes are done on the same place in a shared memory. Semaphores and Mutex locks are used to ensure exclusive access (Width variable, globalWidth shared memory).

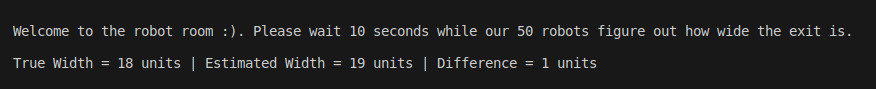
**Memory Management:** Shared memory detach was used everywhere when it wasn’t needed. Additionally, all shared memory resources, semaphores and mutexes created were destroyed appropriately.

**Interprocess Communication:** Shared memory was used to share and communicate exit information, robot information and global width between all processes.

**Thread Management:** Each robot has 50 threads where they read information of other robots and calculates neighbouring robots and add their widths to robot’s own width. All threads were joined properly and it was made sure that there was no dangling/zombie thread in the system.

**Process Scheduling:** Each robot would wait for all robots to finish writing their coordinates to the shared memory. Only after that, they would start the reading operations. While each robot was adjusting its width based on detecting neighbours in different threads, a mutex was used to ensure proper scheduling and avoiding race conditions on width variable. Additionally, after the 10 seconds wait, all processes were scheduled to add their width to the global width shared variable one by one by using a semaphore.

# Output (Verbose logging level 0)



# Thank You!