

**NAME:** SELVA KARTHIK S

**REGISTER NO:** 126003238

**SECTION:** B

**ROLL NO:** 43

**COURSE CODE:** CSE308

**COURSE NAME:** OPERATING SYSTEM

**ASSIGNMENT NO:** 1

**PROBLEM STATEMENT:**

You are tasked with implementing a resource management system for a smart factory that operates with multiple robotic arms and conveyor belts. Each robot arm can access several shared resources such as tools (screwdrivers, hammers) and workstations. These resources are finite in number.

* ***OBJECTIVE:***

The resource management system for the smart factory involves four types of robotic arms, each requiring different combinations of shared resources (tools and workstations). The objective is to ensure efficient resource allocation while preventing deadlock and starvation, ensuring fair usage of shared resources.

* ***DESIGN DECISIONS:***

1. **Resource Allocation and Synchronization**

We will use mutexes of semaphores (Binary) to ensure that only one robot can access a tool or workstation at a time. Each robot type will have different resource requirements for performing its tasks, and these resources will be allocated based on the following principles:

* **Shared Resources:** Each type of robot will need tools (e.g., screwdrivers, hammers) and workstations.
* **Randomized Resource Requirements:** The number of resources required for each task will vary randomly, adding complexity to resource allocation.
* **Mutexes for Resource Locking:** Each resource will be protected by a mutex or semaphore, ensuring that only one robot can use it at a time.

**2. Deadlock Prevention**

To prevent deadlock, we can use one of the following strategies:

* **Resource Ordering:** Assign a global ordering to resources (e.g., screwdriver < hammer < workstation). Robots will acquire resources in a specific order to avoid circular wait conditions.
* **Banker’s Algorithm:** Keep track of each robot's maximum resource needs and the available resources, ensuring that robots can only acquire resources if the system remains in a safe state.
* **Timeout Mechanism:** If a robot cannot acquire all the necessary resources within a certain time, it will release any held resources and retry.

Let us first consider **resource ordering strategy** for simplicity and efficiency. This will ensure that robots acquire resources in a predefined order, thereby avoiding **circular wait and deadlock.**

***We will then make a comparative study on various AI tools used along with various mechanisms used above.***

***3. Starvation Prevention***

*To prevent starvation, we will implement a* ***fair scheduling mechanism*** *that guarantees that all robots will eventually get the resources they need after a certain wait time. This could involve:*

* ***Round-Robin Scheduling:*** *Each robot is given a turn to acquire resources, ensuring that no single robot monopolizes the resources.*
* ***FCFS:*** *Robots will be added to a queue when they request resources. The scheduler will process requests in the order they were made.*

*Let us consider* ***First-come First Server*** *to be used here to ensure fairness, where each robot gets a turn to acquire resources after previous robot’s tasks get completed.*

* ***SAMPLE CODE:***

|  |
| --- |
| #include <stdio.h>  #include <pthread.h>  #include <stdlib.h>  #include <unistd.h>  #include <time.h>  #define NUM\_ROBOTS 4  #define NUM\_ROUNDS 10  // Mutex for each resource  *pthread\_mutex\_t* screwdriver;  *pthread\_mutex\_t* hammer;  *pthread\_mutex\_t* workstation;  // Enum to represent robot types  typedef *enum* {      ROBOT\_TYPE\_A, // Requires Screwdriver and Hammer      ROBOT\_TYPE\_B, // Requires Hammer and Workstation      ROBOT\_TYPE\_C, // Requires Screwdriver and Workstation      ROBOT\_TYPE\_D  // Requires all resources  } RobotType;  // Struct to store metrics for each robot  typedef *struct* {  *int* robot\_id;  *int* tasks\_completed;  *float* total\_execution\_time;  *int* resources\_used;  // Count of resources acquired  } RobotMetrics;  // Global metrics array  RobotMetrics metrics[NUM\_ROBOTS];  // Robot task  *void* \*robot(*void* \**arg*) {  *int* robot\_id = \*(*int* \*)*arg*;      RobotType robot\_type = robot\_id % NUM\_ROBOTS; // Assign type based on ID  *int* round\_time; // Variable to track time spent in each round      for (*int* round = 0; round < NUM\_ROUNDS; round++) {          printf("Robot %d (Type %d) is waiting for resources in round %d...\n", robot\_id, robot\_type, round + 1);          time\_t start\_time = time(NULL); // Start time for this round          // Introduce randomness in resource requirements  *int* require\_screwdriver = rand() % 2;  *int* require\_hammer = rand() % 2;  *int* require\_workstation = rand() % 2;          // Acquire resources based on robot type and random requirements          if (require\_screwdriver && robot\_type != ROBOT\_TYPE\_B) {              pthread\_mutex\_lock(&screwdriver);              printf("Robot %d acquired Screwdriver\n", robot\_id);              metrics[robot\_id - 1].resources\_used++; // Increment resources used          }          if (require\_hammer && robot\_type != ROBOT\_TYPE\_C) {              pthread\_mutex\_lock(&hammer);              printf("Robot %d acquired Hammer\n", robot\_id);              metrics[robot\_id - 1].resources\_used++; // Increment resources used          }          if (require\_workstation) {              pthread\_mutex\_lock(&workstation);              printf("Robot %d acquired Workstation\n", robot\_id);              metrics[robot\_id - 1].resources\_used++; // Increment resources used          }          // Simulate robot task          printf("Robot %d is performing its task.\n", robot\_id);          sleep(rand() % 2 + 1);  // Simulate some work being done          // Release resources after task completion          if (require\_workstation) {              pthread\_mutex\_unlock(&workstation);              printf("Robot %d released Workstation\n", robot\_id);          }          if (require\_hammer && robot\_type != ROBOT\_TYPE\_C) {              pthread\_mutex\_unlock(&hammer);              printf("Robot %d released Hammer\n", robot\_id);          }          if (require\_screwdriver && robot\_type != ROBOT\_TYPE\_B) {              pthread\_mutex\_unlock(&screwdriver);              printf("Robot %d released Screwdriver\n", robot\_id);          }          // Calculate execution time for this round          round\_time = time(NULL) - start\_time;          metrics[robot\_id - 1].total\_execution\_time += round\_time;          // Increment tasks completed          metrics[robot\_id - 1].tasks\_completed++;          // Simulate a break before the next round          sleep(1);      }      printf("Robot %d completed all rounds.\n", robot\_id);      return NULL;  }  *int* main() {  *pthread\_t* robot\_threads[NUM\_ROBOTS];  *int* robot\_ids[NUM\_ROBOTS];      // Initialize mutexes for resources      pthread\_mutex\_init(&screwdriver, NULL);      pthread\_mutex\_init(&hammer, NULL);      pthread\_mutex\_init(&workstation, NULL);      // Seed random number generator      srand(time(NULL));      // Create robot threads      for (*int* i = 0; i < NUM\_ROBOTS; i++) {          robot\_ids[i] = i + 1;          metrics[i].robot\_id = robot\_ids[i]; // Initialize metrics          pthread\_create(&robot\_threads[i], NULL, robot, &robot\_ids[i]);      }      // Wait for all robots to complete their tasks      for (*int* i = 0; i < NUM\_ROBOTS; i++) {          pthread\_join(robot\_threads[i], NULL);      }      // Destroy the mutexes after use      pthread\_mutex\_destroy(&screwdriver);      pthread\_mutex\_destroy(&hammer);      pthread\_mutex\_destroy(&workstation);      // Display metrics      printf("\n=== Metrics ===\n");      for (*int* i = 0; i < NUM\_ROBOTS; i++) {          printf("Robot %d:\n", metrics[i].robot\_id);          printf("  Tasks Completed: %d\n", metrics[i].tasks\_completed);          printf("  Total Execution Time: %.2f seconds\n", metrics[i].total\_execution\_time);          printf("  Resources Used: %d\n", metrics[i].resources\_used);      }      printf("All robots have finished their tasks.\n");      return 0;  } |

* ***SAMPLE INPUT/OUTPUT:***

|  |
| --- |
| root@Dell-Inspiron-5593:~/Code# gcc SC.c  root@Dell-Inspiron-5593:~/Code# ./a.out  Robot 1 (Type 1) is waiting for resources in round 1...  Robot 1 acquired Workstation  Robot 1 is performing its task.  Robot 2 (Type 2) is waiting for resources in round 1...  Robot 2 is performing its task.  Robot 4 (Type 0) is waiting for resources in round 1...  Robot 4 acquired Screwdriver  Robot 3 (Type 3) is waiting for resources in round 1...  Robot 1 released Workstation  Robot 4 acquired Workstation  Robot 4 is performing its task.  Robot 2 (Type 2) is waiting for resources in round 2...  Robot 1 (Type 1) is waiting for resources in round 2...  Robot 1 acquired Hammer  Robot 1 is performing its task.  Robot 4 released Workstation  Robot 4 released Screwdriver  Robot 3 acquired Screwdriver  Robot 4 (Type 0) is waiting for resources in round 2...  Robot 4 acquired Workstation  Robot 4 is performing its task.  Robot 1 released Hammer  Robot 3 acquired Hammer  Robot 4 released Workstation  Robot 3 acquired Workstation  Robot 3 is performing its task.  Robot 1 (Type 1) is waiting for resources in round 3...  Robot 4 (Type 0) is waiting for resources in round 3...  Robot 3 released Workstation  Robot 3 released Hammer  Robot 3 released Screwdriver  Robot 1 acquired Hammer  Robot 2 acquired Screwdriver  Robot 2 is performing its task.  Robot 1 is performing its task.  Robot 3 (Type 3) is waiting for resources in round 2...  Robot 2 released Screwdriver  Robot 4 acquired Screwdriver  Robot 4 is performing its task.  Robot 1 released Hammer  Robot 2 (Type 2) is waiting for resources in round 3...  Robot 2 is performing its task.  Robot 4 released Screwdriver  Robot 1 (Type 1) is waiting for resources in round 4...  Robot 1 acquired Hammer  Robot 1 is performing its task.  Robot 3 acquired Screwdriver  Robot 3 acquired Workstation  Robot 3 is performing its task.  Robot 4 (Type 0) is waiting for resources in round 4...  Robot 1 released Hammer  Robot 4 acquired Workstation  Robot 4 is performing its task.  Robot 3 released Workstation  Robot 3 released Screwdriver  Robot 2 (Type 2) is waiting for resources in round 4...  Robot 2 acquired Screwdriver  Robot 1 (Type 1) is waiting for resources in round 5...  Robot 2 acquired Workstation  Robot 2 is performing its task.  Robot 1 is performing its task.  Robot 4 released Workstation  Robot 3 (Type 3) is waiting for resources in round 3...  Robot 3 acquired Hammer  Robot 2 released Workstation  Robot 2 released Screwdriver  Robot 4 (Type 0) is waiting for resources in round 5...  Robot 4 acquired Screwdriver  Robot 3 acquired Workstation  Robot 3 is performing its task.  Robot 2 (Type 2) is waiting for resources in round 5...  Robot 2 acquired Workstation  Robot 2 is performing its task.  Robot 3 released Workstation  Robot 3 released Hammer  Robot 4 acquired Hammer  Robot 4 is performing its task.  Robot 1 (Type 1) is waiting for resources in round 6...  Robot 2 released Workstation  Robot 3 (Type 3) is waiting for resources in round 4...  Robot 4 released Hammer  Robot 4 released Screwdriver  Robot 1 acquired Hammer  Robot 1 acquired Workstation  Robot 1 is performing its task.  Robot 3 acquired Screwdriver  Robot 2 (Type 2) is waiting for resources in round 6...  Robot 4 (Type 0) is waiting for resources in round 6...  Robot 1 released Workstation  Robot 1 released Hammer  Robot 3 acquired Hammer  Robot 3 is performing its task.  Robot 1 (Type 1) is waiting for resources in round 7...  Robot 3 released Hammer  Robot 3 released Screwdriver  Robot 4 acquired Hammer  Robot 4 acquired Workstation  Robot 4 is performing its task.  Robot 2 acquired Screwdriver  Robot 3 (Type 3) is waiting for resources in round 5...  Robot 4 released Workstation  Robot 4 released Hammer  Robot 2 acquired Workstation  Robot 2 is performing its task.  Robot 1 acquired Hammer  Robot 4 (Type 0) is waiting for resources in round 7...  Robot 2 released Workstation  Robot 2 released Screwdriver  Robot 4 is performing its task.  Robot 1 acquired Workstation  Robot 1 is performing its task.  Robot 2 (Type 2) is waiting for resources in round 7...  Robot 2 is performing its task.  Robot 1 released Workstation  Robot 1 released Hammer  Robot 3 acquired Hammer  Robot 3 is performing its task.  Robot 4 (Type 0) is waiting for resources in round 8...  Robot 4 acquired Screwdriver  Robot 1 (Type 1) is waiting for resources in round 8...  Robot 1 is performing its task.  Robot 2 (Type 2) is waiting for resources in round 8...  Robot 2 is performing its task.  Robot 3 released Hammer  Robot 4 acquired Hammer  Robot 4 is performing its task.  Robot 3 (Type 3) is waiting for resources in round 6...  Robot 1 (Type 1) is waiting for resources in round 9...  Robot 2 (Type 2) is waiting for resources in round 9...  Robot 2 is performing its task.  Robot 4 released Hammer  Robot 4 released Screwdriver  Robot 1 acquired Hammer  Robot 1 is performing its task.  Robot 3 acquired Screwdriver  Robot 4 (Type 0) is waiting for resources in round 9...  Robot 2 (Type 2) is waiting for resources in round 10...  Robot 3 acquired Hammer  Robot 3 is performing its task.  Robot 1 released Hammer  Robot 1 (Type 1) is waiting for resources in round 10...  Robot 3 released Hammer  Robot 3 released Screwdriver  Robot 1 acquired Hammer  Robot 1 acquired Workstation  Robot 1 is performing its task.  Robot 4 acquired Screwdriver  Robot 3 (Type 3) is waiting for resources in round 7...  Robot 1 released Workstation  Robot 1 released Hammer  Robot 4 acquired Hammer  Robot 4 acquired Workstation  Robot 4 is performing its task.  Robot 1 completed all rounds.  Robot 4 released Workstation  Robot 4 released Hammer  Robot 3 acquired Hammer  Robot 2 acquired Screwdriver  Robot 2 acquired Workstation  Robot 2 is performing its task.  Robot 3 is performing its task.  Robot 4 released Screwdriver  Robot 4 (Type 0) is waiting for resources in round 10...  Robot 3 released Hammer  Robot 2 released Workstation  Robot 2 released Screwdriver  Robot 4 acquired Screwdriver  Robot 4 acquired Hammer  Robot 4 acquired Workstation  Robot 4 is performing its task.  Robot 3 (Type 3) is waiting for resources in round 8...  Robot 2 completed all rounds.  Robot 4 released Workstation  Robot 4 released Hammer  Robot 4 released Screwdriver  Robot 3 acquired Screwdriver  Robot 3 acquired Hammer  Robot 3 acquired Workstation  Robot 3 is performing its task.  Robot 4 completed all rounds.  Robot 3 released Workstation  Robot 3 released Hammer  Robot 3 released Screwdriver  Robot 3 (Type 3) is waiting for resources in round 9...  Robot 3 acquired Screwdriver  Robot 3 acquired Hammer  Robot 3 acquired Workstation  Robot 3 is performing its task.  Robot 3 released Workstation  Robot 3 released Hammer  Robot 3 released Screwdriver  Robot 3 (Type 3) is waiting for resources in round 10...  Robot 3 acquired Screwdriver  Robot 3 acquired Hammer  Robot 3 acquired Workstation  Robot 3 is performing its task.  Robot 3 released Workstation  Robot 3 released Hammer  Robot 3 released Screwdriver  Robot 3 completed all rounds.  === Metrics ===  Robot 1:  Tasks Completed: 10  Total Execution Time: 24.00 seconds  Resources Used: 11  Robot 2:  Tasks Completed: 10  Total Execution Time: 27.00 seconds  Resources Used: 8  Robot 3:  Tasks Completed: 10  Total Execution Time: 33.00 seconds  Resources Used: 22  Robot 4:  Tasks Completed: 10  Total Execution Time: 28.00 seconds  Resources Used: 17  All robots have finished their tasks. |

* ***COMPARITIVE ANALYSIS:***

|  |  |
| --- | --- |
| ***Resource Ordering Strategy & First-Come First-Server*** | ***Resource Ordering Strategy & Round-Robin Scheduling*** |
| **Robot 1:**  Tasks Completed: 10  Total Execution Time: 24.00 seconds  Resources Used: 11  **Robot 2:**  Tasks Completed: 10  Total Execution Time: 27.00 seconds  Resources Used: 8  **Robot 3:**  Tasks Completed: 10  Total Execution Time: 33.00 seconds  Resources Used: 22  **Robot 4:**  Tasks Completed: 10  Total Execution Time: 28.00 seconds  Resources Used: 17 | **Robot 1:**  Tasks Completed: 10  Total Execution Time: 18.00 seconds  Resources Used: 8  **Robot 2:**  Tasks Completed: 10  Total Execution Time: 15.00 seconds  Resources Used: 10  **Robot 3:**  Tasks Completed: 10  Total Execution Time: 19.00 seconds  Resources Used: 14  **Robot 4:**  Tasks Completed: 10  Total Execution Time: 14.00 seconds  Resources Used: 19 |

|  |  |
| --- | --- |
| ***Bankers Algorithm & First-Come First-Server*** | ***Bankers Algorithm & Round-Robin Scheduling*** |
| **Robot 1:**  Tasks Completed: 10  Total Execution Time: 13.00 seconds  Resources Used: 8  **Robot 2:**  Tasks Completed: 10,  Total Execution Time: 14.00 seconds  Resources Used: 9  **Robot 3:**  Tasks Completed: 10  Total Execution Time: 13.00 seconds  Resources Used: 10  **Robot 4:**  Tasks Completed: 10  Total Execution Time: 9.00 seconds  Resources Used: 6 | **Robot 1:**  Tasks Completed: 10  Total Execution Time: 21.00 seconds  Resources Used: 10  **Robot 2:**  Tasks Completed: 10  Total Execution Time: 19.00 seconds  Resources Used: 11  **Robot 3:**  Tasks Completed: 10  Total Execution Time: 16.00 seconds  Resources Used: 12  **Robot 4:**  Tasks Completed: 10  Total Execution Time: 22.00 seconds  Resources Used: 9 |

|  |  |
| --- | --- |
| ***Timeout Mechanism & First-Come First-Server*** | ***Timeout Mechanism & Round-Robin Scheduling*** |
| **Robot 1:**  Tasks Completed: 1  Total Execution Time: 10.00 seconds  Resources Used: 2  **Robot 2:**  Tasks Completed: 1  Total Execution Time: 10.00 seconds  Resources Used: 1  **Robot 3:**  Tasks Completed: 1  Total Execution Time: 10.00 seconds  Resources Used: 1  **Robot 4:**  Tasks Completed: 1  Total Execution Time: 10.00 seconds  Resources Used: 1 | **Robot 1:**  Tasks Completed: 8  Total Execution Time: 33.00 seconds  Resources Used: 8  **Robot 2:**  Tasks Completed: 8  Total Execution Time: 36.00 seconds  Resources Used: 8  **Robot 3:**  Tasks Completed: 9  Total Execution Time: 43.00 seconds  Resources Used: 9  **Robot 4:**  Tasks Completed: 7  Total Execution Time: 33.00 seconds  Resources Used: 7 |

* ***CONCLUSIONS:***

Given the above analysis, the best strategy to choose is:

***Banker's Algorithm & First-Come First-Server***

**Efficiency**: It has the lowest total execution times across all robots while still completing the maximum number of tasks.

**Resource Management**: It shows effective resource usage, minimizing the total resources needed compared to the other strategies.

**Consistency**: All robots completed the same number of tasks with lower execution times, indicating that this strategy allows for reliable performance without overloading the system.

* ***REFERENCES AND ADDITIONAL CODE:***
* ***CHATGPT***
* ***GEMINI***
* ***REVOLUTIONARY AI***
* ***REPLIT AI***

The Source Code for all the Models:

<https://github.com/SelvaKarthik01/OS-Assignment>

X---------------------------------------X