ISA 3

Name: Gauresh Rekdo

Roll no: 17

# Robotic Arm Control System

In manufacturing and assembly lines, robotic arms are often used for tasks such as welding, painting, assembly, and material handling. When multiple robotic arms operate in the same area or on shared resources (like a conveyor belt or workspace), semaphores are essential for coordinating their movements and ensuring safe operation. Semaphores can control access to shared resources or coordinate tasks between different robotic arms, preventing collisions or interference during operation.

1. Multiple robotic arms are tasked with picking and placing items on a conveyor belt.
2. Each arm must wait for its turn to access a shared workspace to avoid collisions.
3. A semaphore is used to manage the sequence of movements, ensuring that only one arm operates in the critical section at a time.

**# C Code**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_ARMS 2

sem\_t semaphore; // Semaphore for controlling access

void\* robotic\_arm(void\* args) {

int arm\_id = \*((int\*)args);

// Simulate the robotic arm requesting access to the workspace

printf("Robotic Arm %d is waiting to access the workspace.\n", arm\_id);

sem\_wait(&semaphore); // Wait for access to the workspace

// Critical section - perform task

printf("Robotic Arm %d is now working...\n", arm\_id);

sleep(2); // Simulate work being done

printf("Robotic Arm %d has finished working.\n", arm\_id);

sem\_post(&semaphore); // Release the workspace

return NULL;

}

int main() {

pthread\_t arms[NUM\_ARMS];

int arm\_ids[NUM\_ARMS] = {1, 2};

// Initialize the semaphore with a value of 1 (binary semaphore)

sem\_init(&semaphore, 0, 1);

// Create robotic arm threads

for (int i = 0; i < NUM\_ARMS; i++) {

pthread\_create(&arms[i], NULL, robotic\_arm, &arm\_ids[i]);

}

// Wait for the threads to finish

for (int i = 0; i < NUM\_ARMS; i++) {

pthread\_join(arms[i], NULL);

}

// Clean up the semaphore

sem\_destroy(&semaphore);

return 0;

}

**# Python Code**

import threading

import time

NUM\_ARMS = 2

semaphore = threading.Semaphore(1) # Binary semaphore

def robotic\_arm(arm\_id):

print(f"Robotic Arm {arm\_id} is waiting to access the workspace.")

semaphore.acquire() # Wait for access to the workspace

# Critical section - perform task

print(f"Robotic Arm {arm\_id} is now working...")

time.sleep(2) # Simulate work being done

print(f"Robotic Arm {arm\_id} has finished working.")

semaphore.release() # Release the workspace

if \_\_name\_\_ == "\_\_main\_\_":

threads = []

# Create robotic arm threads

for i in range(1, NUM\_ARMS + 1):

thread = threading.Thread(target=robotic\_arm, args=(i,))

threads.append(thread)

thread.start()

# Wait for threads to finish

for thread in threads:

thread.join()

Test Case 1: Simultaneous Access Request

Two robotic arms request to pick an item from the same location simultaneously. The semaphore should ensure that only one arm can access that location.

Expected Result: One arm picks the item, while the other waits until the first is done.

Test Case 2: Sequential Task Execution

Robotic arms perform tasks that must be completed in a specific order (e.g., one arm assembles a part before another can paint it).

Expected Result: The first arm completes its task before the second arm begins.

Test Case 3: Collision Prevention

Two robotic arms are programmed to operate in overlapping workspaces. The semaphore should prevent them from entering a shared space simultaneously.

Expected Result: One arm is stopped by the semaphore until the other completes its task in the shared area.