Spring Workers - Synchronization Problem Simulation Report

Team Members:

Qurratulain	**CS-22018**
Malaika Mustafa	**CS-22006**
Sumaika Imran	**CS-22009**
Nensi Batra	**CS-21086**

Introduction:

This report documents the design, implementation, and testing of the Spring Workers simulation, a solution to a classical synchronization problem. The program simulates multiple picker threads collecting fruits from a tree and placing them in a shared crate, which is then transported by a loader thread. This activity was conducted to fulfil the requirements of a complex engineering problem (CEP) as part of the Operating Systems course.

Synchronization problems are common in concurrent and parallel systems. This simulation is inspired by the producer-consumer problem, where pickers act as producers and the loader acts as the consumer. Proper thread synchronization is crucial to avoid race conditions and ensure mutual exclusion.

Problem Statement:

Simulate a tree laden with fruits. Launch three picker processes and one loader process in parallel. If there is fruit on the tree, a picker picks it and places it in a slot of a crate with 12 slots. When a picker finds that the crate is full, it calls the loader. It waits for the loader to place this crate in a truck. Then, the loader furnishes a new crate for the pickers. We assume there is enough space in the truck for all crates. All pickers return to the main function when the tree is bare. In the end, the loader places any partially filled crate in the truck if present. If a picker is adding to the last crate, the loader waits for it to complete the action.

Implementation Details:

- The main function initializes the tree and creates an empty crate.
- Three picker threads are started and begin picking fruits from the tree.
- Each picker locks access to the tree to safely pick a fruit.
- Once a fruit is picked, the picker locks access to the crate and places the fruit in it.
- If the crate becomes full, the picker signals the loader and waits for the crate to be emptied.
- The loader waits for the signal, moves the full crate to the truck, and clears the crate.
- Pickers resume placing fruits once the crate is emptied.
- This cycle continues until all fruits are picked.
- At the end, the loader places any remaining fruits from the crate into the truck.
- All threads exit gracefully and the main function confirms completion.

Test Cases:

Test Case 1: Normal Scenario

- Fruits:30
- Crate_size:12
- Expected: Two full crates and one partial crate of 6 fruits moved. All pickers and loader exit correctly.

Test Case 2: Exact Fit

- Fruits: 24
- Crate size: 12
- Expected: Two full crates moved. No partial crate. Loader exits cleanly after all fruits are picked.

Test Case 3: Underfilled Crate

- Fruits: 10
- Crate size: 12
- Expected: One partial crate moved. Loader exits after moving the only crate. No deadlocks.

Code Attachment:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>  // For sleep and usleep

// Constants
#define MAX_FRUITS 30     // Total number of fruits on the tree
#define CRATE_SIZE 12     // Maximum size of crate
#define PICKER_COUNT 3    // Number of pickers (threads)

// Shared data
int tree[MAX_FRUITS];     // Tree as an array of fruits
int tree_size = MAX_FRUITS; // Number of fruits currently on the tree
int crate[CRATE_SIZE];    // Crate to store picked fruits
int crate_count = 0;      // How many fruits are currently in the crate

// Threading and synchronization variables
pthread_t pickers[PICKER_COUNT];    // Picker threads
pthread_t loader_thread;            // Loader thread

pthread_mutex_t tree_mutex = PTHREAD_MUTEX_INITIALIZER;     // Mutex for tree access
pthread_mutex_t crate_mutex = PTHREAD_MUTEX_INITIALIZER;    // Mutex for crate access
pthread_cond_t crate_full = PTHREAD_COND_INITIALIZER;       // Condition: crate is full
pthread_cond_t crate_emptied = PTHREAD_COND_INITIALIZER;    // Condition: crate is emptied
```

```
// Function to initialize the tree with fruits (fruit IDs)
void init_tree() {
    for (int i = 0; i < MAX_FRUITS; i++) {
        tree[i] = i + 1;  // Fruits are numbered 1 to MAX_FRUITS
    }
}

// Picker thread function
void* picker(void* arg) {
    int id = *((int*)arg);  // Picker ID

    while (1) {
        int fruit = -1;

        // ------------------ STEP 1: Pick a fruit from the tree ------------------
        pthread_mutex_lock(&tree_mutex);  // Lock the tree before picking
        if (tree_size > 0) {
            tree_size = tree_size - 1;       // Decrease the fruit count
            fruit = tree[tree_size];         // Take the last fruit
            printf("Picker %d picked fruit %d\n", id, fruit);
        }
        pthread_mutex_unlock(&tree_mutex);  // Unlock the tree

        if (fruit == -1) {
            break;  // No more fruits to pick
        }

        // ------------------ STEP 2: Add fruit to crate ------------------
        pthread_mutex_lock(&crate_mutex);  // Lock the crate

        // Wait if the crate is full and hasn't been emptied yet
        while (crate_count == CRATE_SIZE) {
            pthread_cond_wait(&crate_emptied, &crate_mutex);  // Wait for loader to empty
        }

        crate[crate_count] = fruit;         // Place fruit in crate
        crate_count = crate_count + 1;      // Update crate count
        printf("Picker %d placed fruit %d in crate slot %d\n", id, fruit, crate_count);

        // If crate is full, notify the loader
        if (crate_count == CRATE_SIZE) {
            printf("Picker %d: Crate is full. Notifying loader...\n", id);
            pthread_cond_signal(&crate_full);  // Wake up loader
        }

        pthread_mutex_unlock(&crate_mutex);  // Unlock the crate

        usleep(100000);  // Simulate a delay (100 ms)
    }

    printf("Picker %d is done.\n", id);
```

```
    pthread_exit(NULL);
    return NULL;
}

// Loader thread function
void* loader(void* arg) {
    (void)arg; // Add this line at the top of the function

    while (1) {
        pthread_mutex_lock(&crate_mutex);

        // Wait for crate to be full OR all fruits are picked
        while (crate_count < CRATE_SIZE) {
            // Check if picking is done
            pthread_mutex_lock(&tree_mutex);
            int no_more_fruits = (tree_size == 0);
            pthread_mutex_unlock(&tree_mutex);

            if (no_more_fruits) {
                break;  // No need to wait anymore
            }

            pthread_cond_wait(&crate_full, &crate_mutex);  // Wait for crate to be full
        }

        // If there's something in the crate, move it to truck
        if (crate_count > 0) {
            printf("Loader: Moving crate to truck [ ");
            for (int i = 0; i < crate_count; i++) {
                printf("%d ", crate[i]);
                crate[i] = -1;  // Empty the crate slot
            }
            printf("]\n");

            crate_count = 0;  // Reset crate count
            pthread_cond_broadcast(&crate_emptied);  // Wake all pickers waiting for empty crate
        }

        pthread_mutex_unlock(&crate_mutex);

        // Check if everything is done
        pthread_mutex_lock(&tree_mutex);
        int done = (tree_size == 0);
        pthread_mutex_unlock(&tree_mutex);

        if (done && crate_count == 0) {
            break;  // Exit loader if all fruits picked and crate is empty
        }

        usleep(150000);  // Simulate loading delay (150 ms)
    }
```

```
    printf("Loader: All crates delivered. Exiting.\n");
    pthread_exit(NULL);
    return NULL;
}

// ------------------ MAIN FUNCTION ------------------
int main() {
    int picker_ids[PICKER_COUNT];  // Array to store picker IDs

    init_tree();  // Fill the tree with fruits

    // Initialize crate with empty slots
    for (int i = 0; i < CRATE_SIZE; i++) {
        crate[i] = -1;
    }

    // Create the loader thread
    pthread_create(&loader_thread, NULL, loader, NULL);

    // Create the picker threads
    for (int i = 0; i < PICKER_COUNT; i++) {
        picker_ids[i] = i + 1;  // Assign ID starting from 1
        pthread_create(&pickers[i], NULL, picker, &picker_ids[i]);
    }

    // Wait for all picker threads to finish
    for (int i = 0; i < PICKER_COUNT; i++) {
        pthread_join(pickers[i], NULL);
    }

    // Wake up loader in case it's waiting and pickers are done
    pthread_cond_signal(&crate_full);

    // Wait for the loader to finish
    pthread_join(loader_thread, NULL);

    // Clean up
    pthread_mutex_destroy(&tree_mutex);
    pthread_mutex_destroy(&crate_mutex);
    pthread_cond_destroy(&crate_full);
    pthread_cond_destroy(&crate_emptied);
    printf("Main: All work done. Simulation complete.\n");
    return 0;
}
```

Output Snapshots:

Test Case 1:

```
Output
Picker 1 picked fruit 30
Picker 1 placed fruit 30 in crate slot 1
Picker 3 picked fruit 29
Picker 3 placed fruit 29 in crate slot 2
Picker 2 picked fruit 28
Picker 2 placed fruit 28 in crate slot 3
Picker 1 picked fruit 27
Picker 1 placed fruit 27 in crate slot 4
Picker 2 picked fruit 26
Picker 2 placed fruit 26 in crate slot 5
Picker 3 picked fruit 25
Picker 3 placed fruit 25 in crate slot 6
Picker 1 picked fruit 24
Picker 1 placed fruit 24 in crate slot 7
Picker 3 picked fruit 23
Picker 3 placed fruit 23 in crate slot 8
Picker 2 picked fruit 22
Picker 2 placed fruit 22 in crate slot 9
Picker 1 picked fruit 21
Picker 1 placed fruit 21 in crate slot 10
Picker 2 picked fruit 20
Picker 2 placed fruit 20 in crate slot 11
Picker 3 picked fruit 19
Picker 3 placed fruit 19 in crate slot 12
Picker 3: Crate is full. Notifying loader...
Loader: Moving crate to truck [ 30 29 28 27 26 25 24 23 22 21 20 19 ]
Picker 1 picked fruit 18
Picker 1 placed fruit 18 in crate slot 1
Picker 3 picked fruit 17
Picker 3 placed fruit 17 in crate slot 2
Picker 2 picked fruit 16
Picker 2 placed fruit 16 in crate slot 3
Picker 1 picked fruit 15
Picker 1 placed fruit 15 in crate slot 4
Picker 3 picked fruit 14
Picker 3 placed fruit 14 in crate slot 5
Picker 2 picked fruit 13
Picker 2 placed fruit 13 in crate slot 6
Picker 1 picked fruit 12
Picker 1 placed fruit 12 in crate slot 7
Picker 3 picked fruit 11
Picker 3 placed fruit 11 in crate slot 8
Picker 2 picked fruit 10
Picker 2 placed fruit 10 in crate slot 9
```

```
Picker 1 picked fruit 9
Picker 1 placed fruit 9 in crate slot 10
Picker 3 picked fruit 8
Picker 3 placed fruit 8 in crate slot 11
Picker 2 picked fruit 7
Picker 2 placed fruit 7 in crate slot 12
Picker 2: Crate is full. Notifying loader...
Loader: Moving crate to truck [ 18 17 16 15 14 13 12 11 10 9 8 7 ]
Picker 3 picked fruit 6
Picker 3 placed fruit 6 in crate slot 1
Picker 1 picked fruit 5
Picker 1 placed fruit 5 in crate slot 2
Picker 2 picked fruit 4
Picker 2 placed fruit 4 in crate slot 3
Picker 3 picked fruit 3
Picker 3 placed fruit 3 in crate slot 4
Picker 2 picked fruit 2
Picker 2 placed fruit 2 in crate slot 5
Picker 1 picked fruit 1
Picker 1 placed fruit 1 in crate slot 6
Picker 3 is done.
```

```
Picker 2 is done.
Picker 1 is done.
Loader: Moving crate to truck [ 6 5 4 3 2 1 ]
Loader: All crates delivered. Exiting.
Main: All work done. Simulation complete.
```

Test Case 2:

Output

```
Picker 1 picked fruit 24
Picker 1 placed fruit 24 in crate slot 1
Picker 3 picked fruit 23
Picker 3 placed fruit 23 in crate slot 2
Picker 2 picked fruit 22
Picker 2 placed fruit 22 in crate slot 3
Picker 3 picked fruit 21
Picker 3 placed fruit 21 in crate slot 4
Picker 1 picked fruit 20
Picker 1 placed fruit 20 in crate slot 5
Picker 2 picked fruit 19
Picker 2 placed fruit 19 in crate slot 6
Picker 3 picked fruit 18
Picker 3 placed fruit 18 in crate slot 7
Picker 2 picked fruit 17
Picker 2 placed fruit 17 in crate slot 8
Picker 1 picked fruit 16
Picker 1 placed fruit 16 in crate slot 9
Picker 2 picked fruit 15
Picker 2 placed fruit 15 in crate slot 10
Picker 1 picked fruit 14
Picker 1 placed fruit 14 in crate slot 11
```

```
Picker 3 picked fruit 13
Picker 3 placed fruit 13 in crate slot 12
Picker 3: Crate is full. Notifying loader...
Loader: Moving crate to truck [ 24 23 22 21 20 19 18 17 16 15 14 13 ]
Picker 2 picked fruit 12
Picker 2 placed fruit 12 in crate slot 1
Picker 1 picked fruit 11
Picker 1 placed fruit 11 in crate slot 2
Picker 3 picked fruit 10
Picker 3 placed fruit 10 in crate slot 3
Picker 2 picked fruit 9
Picker 2 placed fruit 9 in crate slot 4
Picker 1 picked fruit 8
Picker 1 placed fruit 8 in crate slot 5
Picker 3 picked fruit 7
Picker 3 placed fruit 7 in crate slot 6
Picker 2 picked fruit 6
Picker 2 placed fruit 6 in crate slot 7
Picker 1 picked fruit 5
Picker 1 placed fruit 5 in crate slot 8
Picker 3 picked fruit 4
```

```
Picker 3 placed fruit 4 in crate slot 9
Picker 2 picked fruit 3
Picker 2 placed fruit 3 in crate slot 10
Picker 1 picked fruit 2
Picker 1 placed fruit 2 in crate slot 11
Picker 3 picked fruit 1
Picker 3 placed fruit 1 in crate slot 12
Picker 3: Crate is full. Notifying loader...
Loader: Moving crate to truck [ 12 11 10 9 8 7 6 5 4 3 2 1 ]
Loader: All crates delivered. Exiting.
Picker 2 is done.
Picker 1 is done.
Picker 3 is done.
Main: All work done. Simulation complete.
```

Test Case 3:

Output

```
Picker 1 picked fruit 10
Picker 1 placed fruit 10 in crate slot 1
Picker 3 picked fruit 9
Picker 3 placed fruit 9 in crate slot 2
Picker 2 picked fruit 8
Picker 2 placed fruit 8 in crate slot 3
Picker 3 picked fruit 7
Picker 3 placed fruit 7 in crate slot 4
Picker 1 picked fruit 6
Picker 1 placed fruit 6 in crate slot 5
Picker 2 picked fruit 5
Picker 2 placed fruit 5 in crate slot 6
Picker 3 picked fruit 4
Picker 3 placed fruit 4 in crate slot 7
Picker 1 picked fruit 3
Picker 1 placed fruit 3 in crate slot 8
Picker 2 picked fruit 2
Picker 2 placed fruit 2 in crate slot 9
Picker 3 picked fruit 1
Picker 3 placed fruit 1 in crate slot 10
Picker 2 is done.
Picker 1 is done.
Picker 1 is done.
Picker 3 is done.
Loader: Moving crate to truck [ 10 9 8 7 6 5 4 3 2 1 ]
Loader: All crates delivered. Exiting.
Main: All work done. Simulation complete.
```

Conclusion

The Spring Workers simulation demonstrates the practical application of synchronization primitives such as mutexes and condition variables in solving a real-world inspired problem. The solution effectively ensures mutual exclusion and proper coordination between concurrent threads. Through this exercise, we were able to understand thread behaviour, synchronization challenges, and how to design systems that prevent data races and deadlocks. The simulation not only meets the problem requirements but also reflects a solid grasp of system-level programming principles.