EAST WEST UNIVERSITY

Project Report:

Acrylic Painting Problem

Course Code: CSE325

Course Title: Operating Systems

Date of Submission: 30/05/25

Group: 03

Submitted by:

Names	ID
Md.Romjan Ali	2022-3-60-247
Md Moon Rahman Nayem	2022-3-60-210
Raqibul Hasan	2022-3-60-261
Shakhaoat Hossain	2022-3-60-259
Jobeir Ahamed dip	2022-3-60-307

Submitted to:

Course Instructor: Khairum Islam

Lecturer

Department of CSE

Introduction

This project simulates a collaborative acrylic painting process where multiple artists work together in sequence to complete a painting. It divides the workflow into three stages—Preparation, Painting, and Finishing—each handled by separate processes to model real-world task separation. Inside the painting stage, multiple threads represent individual artists working one after another, coordinated using synchronization tools like semaphores and mutexes. The use of shared memory allows all processes and threads to update and share a common progress report, ensuring proper communication and tracking throughout the project. This design demonstrates how multi-processing and multi-threading can be combined to manage complex collaborative tasks safely and efficiently.

Objective

The primary goal of this project is to simulate the real-world scenario of a collaborative painting session involving multiple artists, each responsible for a specific portion of the artwork. The entire painting process is divided into three distinct stages—Preparation, Painting, and Finishing—each handled by separate processes and threads. The design ensures that each stage starts only after the previous one completes, while within the painting stage, individual artist threads work in sequence, maintaining proper synchronization. To achieve this, the project utilizes:

- **Semaphores** for controlling the order and synchronization between artist threads,
- **Mutexes** to provide mutual exclusion when accessing shared resources,
- **Shared memory** to maintain a common progress report accessible across all processes, and
- **Forked processes** to clearly separate the stages of the painting workflow.

SystemDesign

The system is architected into three main stages, each represented as separate processes to mimic real-life workflow separation:

1. **Preparation Stage:** Executed in its own child process, this stage sets up the canvas and prepares the environment, simulating groundwork tasks.
2. **Painting Stage:** This stage is executed by another child process, which creates multiple artist threads. Each artist thread simulates painting a specific section of the canvas. Synchronization ensures artists work one after the other without overlap.
3. **Finishing Stage:** The final stage, executed as a separate child process, applies the finishing touches to the painting and completes the artwork.

Components

- **Main Process:** Oversees the entire workflow by spawning child processes for each stage using `fork()`. It also manages the lifecycle of shared memory.
- **Artist Threads:** Multiple threads inside the painting process simulate individual artists working on the canvas.
- **Semaphores:** Used to enforce sequential painting by allowing only one artist thread to work at a time, preventing conflicts or race conditions.
- **Shared Memory:** Provides a shared space where progress reports from all stages and artists are stored and updated continuously.
- **Mutex Lock:** Ensures safe concurrent access when multiple threads update the shared progress report to avoid data corruption.

Implementation Details

1. **Preparation Stage:**
    - Runs as an independent child process created by the main process.
    - Simulates canvas setup and preparatory activities with a delay introduced by `sleep(2)`.
    - Outputs progress logs to the console and appends status updates to the shared progress report in memory.
2. **Painting Stage:**
    - Runs as a separate child process and spawns four artist threads.
    - Each thread represents an artist painting a section of the canvas sequentially.
    - Semaphores are used to ensure that only one artist paints at a time, passing control from one artist to the next in order.
    - A mutex lock protects the shared progress report when threads write updates, preventing race conditions.
    - Each artist simulates painting for 2 seconds, followed by a drying period of 1 second before signaling the next artist.
3. **Finishing Stage:**
    - Executes as the final child process after the painting stage completes.
    - Simulates final touches with a 2-second delay.
    - Updates and logs completion status to the shared progress report.

Synchronization Techniques

- **Semaphores:**

  An array of semaphores is employed to control the painting order among artist threads. Each semaphore corresponds to an artist and is used to block or allow the thread to proceed. This mechanism ensures strict sequential execution, preventing multiple artists from painting simultaneously and causing data inconsistency.

- **Mutex:**

  The mutex lock guarantees exclusive access to the shared progress report when multiple threads attempt to update it concurrently. This prevents race conditions and ensures the integrity of the progress data.

- **SharedMemory:**

  Shared memory serves as a common communication space across all processes involved. It holds the progress report string, allowing processes and threads to write updates that persist beyond individual process lifetimes. This enables the main process to finally retrieve and print the complete progress report after all stages conclude.

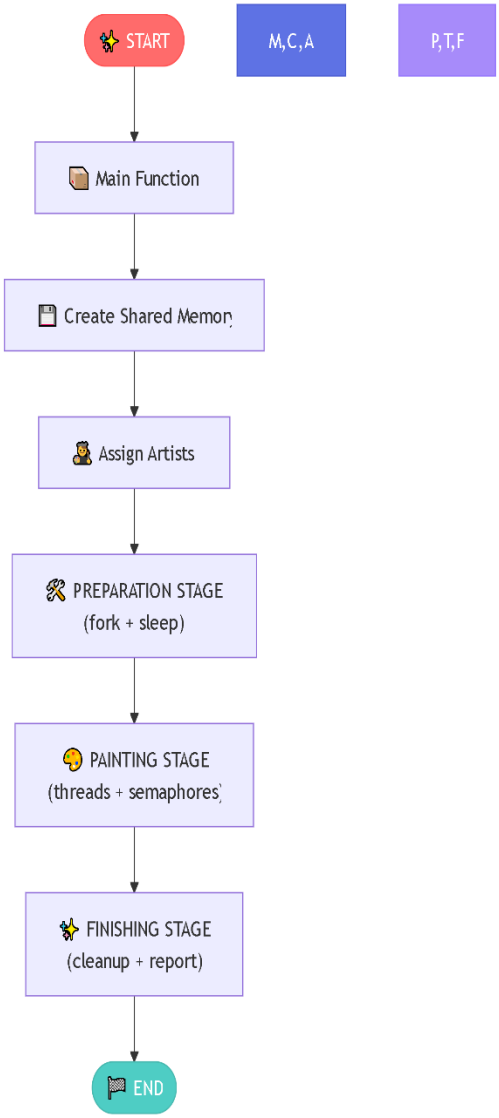
Output

```
ubuntu@ubuntu:~/Desktop$ ./project
Preparation stage started.
Preparation stage completed.
Painting stage started.
Artist 1 is starting their section.
Artist 1 finished painting. Paint drying...
Artist 2 is starting their section.
Artist 2 finished painting. Paint drying...
Artist 3 is starting their section.
Artist 3 finished painting. Paint drying...
Artist 4 is starting their section.
Artist 4 finished painting. Paint drying...
Painting stage completed.
Finishing stage started.
Finishing stage completed.
```

```
Progress Report:
Artist 1 started painting.
Artist 1 finished painting.
Artist 2 started painting.
Artist 2 finished painting.
Artist 3 started painting.
Artist 3 finished painting.
Artist 4 started painting.
Artist 4 finished painting.

Collaborative acrylic painting project completed successfully!
ubuntu@ubuntu:~/Desktop$
```

Flowchart


START
M,C,A
P,T,F
Main Function
Create Shared Memory
Assign Artists
PREPARATION STAGE
(fork + sleep)
PAINTING STAGE
(threads + semaphores)
FINISHING STAGE
(cleanup + report)
END


main function
Create Share Memory
Assign artist for Painting
Thread create
Use Semaphore
Use Mutex Lock
Thread Join
Output

Conclusion

The Collaborative Acrylic Painting System project successfully demonstrates the effective use of multi-processing and multi-threading to simulate a structured painting workflow involving multiple artists. By leveraging semaphores for sequential execution, mutex locks for thread-safe shared memory updates, and process synchronization with `fork()` and `wait()`, the system ensures orderly progression through preparation, painting, and finishing stages. The shared memory segment maintains a consolidated progress report, providing visibility into each artist's contribution. This project highlights key synchronization concepts in concurrent programming while offering a foundation for scalable, real-world collaborative task management. Future enhancements could include dynamic artist allocation and real-time progress visualization, further improving flexibility and user interaction.