WIPRO NGA Program- C++||LSP batch

Capstone Project Presentation - 12th August,2024

Project Title –Multi-Process To Do List Application

(using shared memory)

Presented by – Pooja Karda(24NAG1279_U29)

Project overview

This project implements a multi-process to-do list application using shared memory and semaphores for synchronization. The application allows multiple clients to access and modify a shared to-do list. It provides a simple to-do list application where multiple clients can add, complete, and view tasks.

Introduction

The project is a To-Do List application that utilizes shared memory and semaphores for synchronization. The application allows users to add, complete, and view to-do items in a shared memory segment. The primary objective is to create a to-do list application that can be accessed by multiple processes using IPC. The project is implemented using the C++ programming language with IPC techniques like shared memory and semaphores

Purpose and motivation

- Create a collaborative to-do list for managing tasks.

-The application is designed to facilitate task management in a collaborative environment where multiple processes can access and modify a shared to-do list.

- Demonstrate IPC mechanisms for inter-process communication.

- The project demonstrates the practical use of IPC mechanisms, highlighting their importance in enabling communication between processes.

Key components

- **Shared Memory**: Stores the to-do items and their states.

- Shared memory is used to store an array of To-Do Item structures, representing the tasks and their completion status.

- **Semaphores**: Ensure synchronization between processes.

- To synchronize access to the shared memory, preventing race conditions and ensuring data consistency.

- **User Interface**: A command-line interface is provided to interact with the to-do list application.

Project Scope

- The project focuses on implementing essential functionalities of a to-do list: adding tasks, completing tasks, and viewing tasks.
- Proper synchronization mechanisms (semaphores) are employed to handle concurrent access to shared memory.
- The application provides a command-line interface for users to interact with the to-do list.

Various Applications and Tools Used

- **Compiler**: g++ for compiling the C++ code

- The GNU Compiler Collection (g++) is used to compile the C++ source code.

- **Libraries:** <sys/shm.h>, <semaphore.h> for IPC

- The project includes system libraries for shared memory and semaphores to facilitate IPC.

- **Development Tools**: Text editors like VS Code or vim

- Development tools such as VS Code or vim are used for writing and editing the source code.

- **Linux**: The application is designed to run on a Linux operating system.

Modules

- **Shared Memory Management**: Creating and attaching shared memory.

- The code creates a shared memory segment using shmget and attaches to it using shmat.

- **To-Do Item Functions**: Adding and completing tasks

- The add To-Do Item function adds a new task, and the complete To-Do Item function marks a task as completed.

- **User Interface**: Command-line based interaction loop

- The main function includes a command-line interface that allows users to input commands to add, complete, and view tasks.

System Requirements

- **Operating System**: Linux/Unix-based OS

-The application requires a Linux or Unix-based operating system for IPC mechanisms.

- **Compiler:** g++ (GNU Compiler Collection).

-The code needs the g++ compiler to be compiled and executed.

- Shared memory and semaphore libraries (e.g.sys/shm.h, semaphore.h)
- Minimum 1 GB RAM
- Minimum 500 MB free disk space

List of Functions

- **add To-Do Item**: Adds a new item to the list.

- This function acquires the semaphore, adds a new task to the shared memory, and releases the semaphore.

- **Complete To-Do Item**: Marks an item as completed.

- This function acquires the semaphore, marks the specified task as completed, and releases the semaphore.

- **Print To-Do List:** Displays the current list of items.

- This function acquires the semaphore, prints the list of tasks and their status, and releases the semaphore.

Future Amendments

- **GUI Integration**: Develop a graphical user interface.

- Future versions of the application could include a graphical user interface for better user experience.

Persistent Storage: Save to-do list to disk.

- Implement functionality to save and load the to-do list from persistent storage, such as a file or database

Enhanced Features: Add due dates, priorities, etc.

- Introduce additional features like due dates, task priorities, and categorization..

Addressing Common Challenges

- **Synchronization Issues**: Proper use of semaphores.

 - Ensuring proper use of semaphores to avoid deadlocks and race conditions.

- **Memory Management**: Handling shared memory effectively.

- Efficiently managing shared memory allocation and deallocation.

- **Concurrency**: Ensuring data consistency across processes.

- Maintaining data consistency and integrity across concurrent processes acc essing shared memory.

Conclusion

- This project demonstrates a simple yet effective implementation of a client-server architecture using shared memory and semaphores for inter-process communication. The server manages a shared to-do list, while multiple clients can access and modify the list simultaneously. The use of semaphores ensures data consistency and prevents Corruption.
- The client-server code showcases the key aspects:-

-Effective use of shared memory for inter-process communication

- Synchronization using semaphores to prevent data corruption

- Simple and intuitive client-server interaction

Thank you