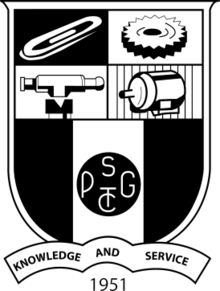
**CONCURRENCY**

**A report on package submitted by**

**Shreenidhi N 18PD33**

**Snehaa S 18PD36**

**Subject : Operating Systems 18XD44**

****

**Department Of Applied Mathematics and Computational Sciences**

**PSG College Of Technology,**

**Coimbatore - 04**

**CONCURRENCY**

**TABLE OF CONTENTS**

**ABSTRACT**

**1.1 INTRODUCTION**

**1.2 DESCRIPTION**

**1.3 SYSTEM CALLS USED**

**1.4 TOOLS AND TECHNOLOGIES**

**1.5 WORKFLOW**

**1.6 RESULTS AND DISCUSSION**

**1.7 CONCLUSION**

**1.8 BIBLIOGRAPHY**

**ABSTRACT**

A database can be shared to multiple users for accessing. The concurrent access to a database by multiple users is implemented using synchronization mechanisms. Precisely, we can call this as a reader-writer problem.

**1.1 INTRODUCTION**

The objective of this assignment is to provide a practical view of the synchronization mechanisms available in the POSIX threads library. In particular, this assignment focuses on the basic structures of mutexes and conditional variables as a means to control the access to critical sections of the program. In this respect, the use of mutexes permits enforcement of mutual exclusion when several threads try to access the critical region at the same time. Condition variables on the other hand permit to wait until a certain condition is met.

The synchronization will be implemented using POSIX thread syscalls.

In particular:

* pthread\_create will be used to create and execute new threads.
* pthread\_join will be used if any thread must wait for the finalization of another thread.
* Mutexes (pthread\_mutex\_init, pthread\_mutex\_lock, pthread\_mutex\_unlock, and pthread\_mutex\_destroy) will be used to protect the access to critical sections.
* Cond. variables (pthread\_cond\_init, pthread\_cond\_wait, pthread\_cond\_signal, and pthread\_cond\_destroy) will be used to wait until a certain condition is met.

The objective is to design and implement a concurrent access library operating on the top of a warehouse database, in order to guarantee the consistency of all operations even if those operations are performed in a concurrent manner.

**1.2 DESCRIPTION**

The starting point is a “pseudo” database named db\_warehouse. This database is a C library that allows you to manage products in a warehouse (create products, and modify the stock of a concrete product). On the top of this warehouse, acting as an interface between db\_warehouse and client threads, there is a library called sequential. This library receives requests from client threads, and forwards this request to the warehouse. The problem is that this sequential library is not thread-safe. There is a risk of corrupting the database if several threads perform operations at the same time. The student must implement the library called concurrent. This library must be a copy of sequential, but concurrent accesses must be controlled in order not to corrupt the database. The access restrictions that the library must fulfill are the following:

There are two types of operation: READ (get stock, get number of products) and WRITE (update stock, create new product, etc.).

* WRITE does not allow any other operation over the database.
* READ over the database can be performed concurrently up to MAX\_READERS (concurrent.h).

**1.3 SYSTEM CALLS USED**

Synchronization using POSIX threads system calls.

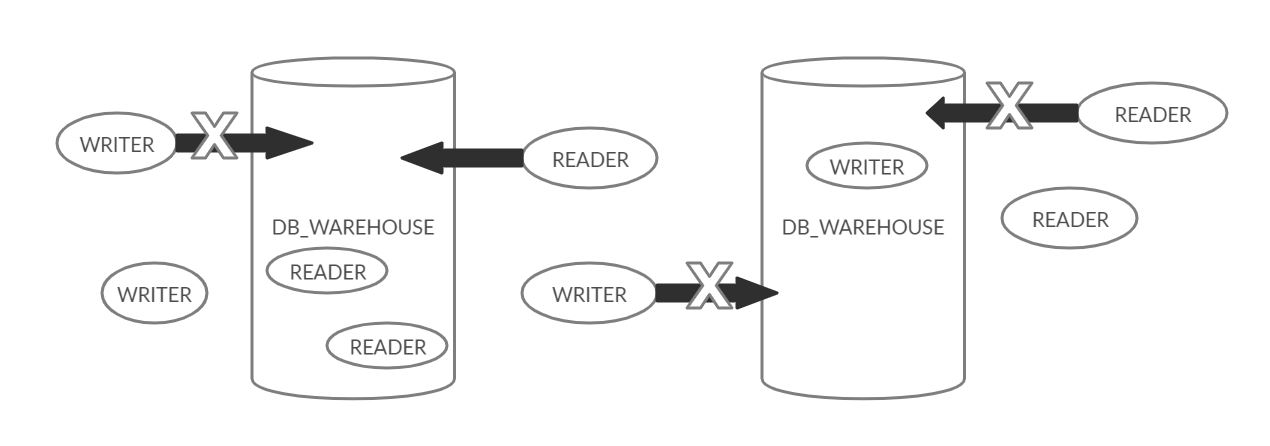
* pthread\_create will be used to create and execute new threads.
* pthread\_join will be used if any thread must wait for the finalization of another thread.
* Mutexes (pthread\_mutex\_init, pthread\_mutex\_lock, pthread\_mutex\_unlock, and pthread\_mutex\_destroy) will be used to protect the access to critical sections.
* Semaphore is a variable used to solve the critical section problems and to achieve process synchronization in the multi processing environment.

**1.4 TOOLS AND TECHNOLOGIES USED**

* Ubuntu LTS 18.04LTS Application
* CodeBlocks 17.12, the open-source, cross-platform IDE
* POSIX Threads Library
* GCC 7 compiler
* Creately for diagrammatic representation

**1.5 WORKFLOW**

* The readers-writers problem relates to an object such as a file that is shared between multiple processes.
* It is a synchronization problem.

****

There are two types of processes in this context.

* **Reader :** Any number of **readers** can read from the shared resource simultaneously.
* **Writer :** Only one **writer** can write to the shared resource. When a **writer** is writing data to the resource, no other process can access the resource. A **writer** cannot write to the resource if there are non zero number of readers accessing the resource at that time.

**1.6 RESULTS AND DISCUSSION**

We have used the multithreading concept and shown the synchronization.

Multiple threads were created and used for accessing the database warehouse which is synchronized and stock incrementation, decrementation which are also synchronized.

Therefore, concurrency is also achieved.

**1.7 CONCLUSION**

We have completed this project successfully and gained knowledge on how concurrency works. Using the POSIX library, we have learnt the various functionalities of it and implemented the same for the given problem statement.

Therefore, we have achieved synchronization without losing the consistency of data.

**1.8 BIBLIOGRAPHY**

**1.8.1 BOOKS**

1. El lenguaje de programación C: diseño e implementación de programas Félix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
2. The UNIX System S.R. Bourne Addison-Wesley, 1983. - Advanced UNIX Programming M.J. Rochkind Prentice-Hall, 1985.
3. Sistemas Operativos: Una visión aplicada Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
4. Programming Utilities and Libraries SUN Microsystems, 1990.
5. Unix man pages (man function)

**1.8.2 WEBSITES**

1. <https://www.geeksforgeeks.org/reader-writer-problem-using-monitors-pthreads/?ref=rp>