CS 588: Computer System Lab
(January-May 2023)
Assignment – 4

Submission Deadline: 11:55 pm on Thursday, 6th April 2023 (hard deadline)

This assignment is a programming assignment where you need to implement the given scenario in C/C++ programming language. The scenario description is given in this document.

Instructions:

- **Each group needs to implement all three questions and make one single submission on shared Gmail ID. Only one member from a group needs to make the submission.**
- **The information about the groups is contained in Table 1 given below.**
- **The application should be implemented in C/C++ programming language only. No other programming language other than C will be accepted.**
- **Submit the set of source code files of the application as a zipped file on cs558.2023@gmail.com by the deadline of 11:55 pm on Thursday, 6th April 2023 (hard deadline). The ZIP file's name should be the same as your group number, for example, "Group_4.zip", or "Group_4.rar", or "Group_4.tar.gz".**
- **The assignment will be evaluated offline/through viva-voce during your lab session on Friday, 7th April 2023) where you will need to explain your source codes and execute them before the evaluator.**
- **Write your own source codes and do not copy from any source. Plagiarism detection tool will be used and any detection of unfair means will be penalised by awarding NEGATIVE marks (equal to the maximum marks for the assignment).**

Question1

In this question, you will use ordinary pipes to implement an inter-process communication scheme for message passing between processes. Assume that there are two directories, d1 and d2, and there are different files in each one of them. Also, each file contains a short-length string of characters. You have to use a parent process that forks two children processes and have each child process check one of the directories.

- Assume that child 1 (child 2) is responsible to check the directory d1 (directory d2, respectively). This child process has to create a list of names of the files in the directory and their contents.
- After creating the list, each child process will send its list to the other child process using a pipe.
- If one of the child processes encounters an error while reading the files in their directory, it should immediately convey the message to other child with its current updates using message passing scheme.

- Upon receiving the list, child 2 (child 1) will create the files listed by child 1 (child 2) in directory d2 (directory d1, respectively) and fill the files with their initial contents.

After these steps, the directories d1 and d2 should be identical.

Question2

Write a C/C++ program that simulates a simple online exam system using message passing for inter-process communication. The program should prompt the user for the number of students taking the exam and the number of questions on the exam. The program should then create a child process for each student and use message queues to communicate the exam questions to the child process.

- Each child process should randomly answer the questions and send the answers back to the parent process using another message queue.
- The parent process should collect all the answers and grade each student's exam.
- The program should display the grade for each student and the overall grade distribution for the exam.
- After grading all the exams, the parent process should wait for all the child processes to finish using the message queue synchronization mechanism. If any child process fails to respond or terminates unexpectedly, the program should display a warning message and take appropriate action to handle the situation

Question3

In this question, you write a multi-threaded Event-reservation system for Nehru Centre. Suppose that you have e events. The auditorium has a capacity c. Queries made to the reservation system are of three types:

1) Inquire the number of available seats in a events,
2) Book k tickets in a events, and
3) Cancel a booked ticket.

In order that the reservation system is not overloaded, there is a limit – call it **MAX** – on the maximum number of active queries at any instant. Moreover, in order to insure consistency of the database, different threads reading/modifying the reservation for the same event must go through a mechanism of mutual exclusion. You are asked to use the *pthread* API calls in order to implement a simulation of this reservation system.

The main (master) thread creates a list of *e* events, and initializes the number of available seats in each events to *c*. The master thread then creates *s* threads that run concurrently in a loop, and make automatically generated random queries periodically. Each query is of one of the three types mentioned above. The type of the query, the event number (for queries of type 1 or 2), the number *k* of seats to book (type 2), and the ticket to cancel (type 3) are generated randomly. During each query (that is, between the beginning and the (successful) completion of a query), a thread sleeps for a random short interval (this may, for example, simulate bank transaction time for booking and cancellation queries).

Moreover, between making two consecutive queries, a thread sleeps for a random short interval. Let us now see how the limit MAX on the number of active queries, and the mutual exclusion are to be handled.

At any point of time, at most MAX queries can be active. Any new query ((MAX + 1)st or (MAX + 2)nd or so on) must wait until one or more of the active queries finish. Use appropriate condition variable(s) to enforce this restriction. Note that this wait is to be interpreted as *blocking*, that is, a thread waiting for the server load to reduce must block until signaled by another thread during the completion of an active query.

For write query(type 2 or 3) :

- Two or more threads are not allowed to write the data concurrently for the same event. Moreover, a write cannot run concurrently when a read query is active on the same event.
- Writing the data for two different events may proceed concurrently. Concurrent reads for the same events are also allowed.

One possibility is to create a mutex for each of the e events. But this may be impractical particularly if e is large. In fact, there can be at most MAX active queries at any time. So a better approach is to maintain a shared table with MAX entries. Each entry in the shared table is a triple consisting of a event number, the query type, and the number of the thread which has made this query. A blank entry may be represented by the event number –1. A read query can proceed provided that the corresponding event is not in the shared table in write mode. Likewise, a write query can proceed provided that the corresponding event is not in the shared table in read/write mode. Access to the shared table should be guarded by an appropriate mutex. Whenever a thread succeeds in making a query, it populates a blank entry in the shared table with the appropriate triple. When the query completes, that particular entry is deleted from the shared table by the thread. Notice that when a query fails in order to insure database consistency, the thread is not blocked. It instead proceeds to make the next query (after a short sleep) in the loop.

After all the threads run for a predetermined amount of time T, the server needs to shut down. At this time, all worker threads exit one by one. After this, the master thread prints the current reservation status for all the events, and exits. Use an appropriate barrier in order to synchronize the master thread with the termination of all the worker threads.
Typical values of the parameters that your program should handle are:

e = the number of events = 100
c = the capacity of auditorium = 500
k = the number of tickets booked in a query of type 2 = A random integer in the range 5–10
s = the number of worker threads = 20
MAX = the maximum number of concurrent active queries = 5
T = the total running time = 1 – 10 minute(s)

Each worker thread may maintain a private (non-shared) list of bookings made by it. A cancellation request is chosen randomly from this private list. The threads should print appropriate diagnostic messages (like query inputs and outputs along with thread numbers, waiting and signaling on a query, time out, and so on).