CS474 Operating System

Project #3 Group programing project
(Due date Nov 22 at 11:59 pm)

October 19, 2020

You may work in groups of two, three, or four. A group should submit one program and one final report, presentation and all members of the group will get the same grade for the report and different grades for presentation (each member of group have to present some part of project).

Format to write your report

Your report should have the following sections:

- Title: The title should be descriptive and fit in one line across the page.
- Authors: This should be right under the title, says who you are.
- Abstract: This is the paper in brief and should state the basic contents and conclusions of the paper. It should be complete enough to understand what will be covered in the project.
- Intro: This is a short overview of what you did, and what you learnt. This should contain more motivation than the abstract.
- Methodology: Description of algorithms you used and a description of the platform you used to the level of detail such that someone else could reproduce the same experiments elsewhere.
- Results: Include your program, input data, outputs of your program
- Conclusions: Summarize the conclusions here, and discuss things you have learnt during this experimentation.

Grading Criteria:

Total possible points: 200

Proposal topic: 20 (due date Nov 5 at 11:59 pm)
Code: 80
Report: 60

Presentation: 40

Note:

- This is subjective, but it would not be fair to grade a group that solves a simple problem the same that takes on a challenging one. The amount of challenge depends on the need for advanced structures learned on this course (multi-thread, synchronization) and the amount of logic needed to solve the problem.
- Some points will be given to a correct solution. On top of that, the quality of the solution will be assessed as well. Again, it would not be fair to grade the same a solution that is better than another, given both are correct. Quality aspects include code indentation, code well commented, a factored design, good variable naming, etc.

Submission

- Submission via Canvas Assignment. (Zip file including report, code, slides, input data, and read me file)
- The name of the file should be the name of the students in this format - LastName FirstName (names of all students in the group).
- Presentation will be on Tuesday and Thursday, Dec 1, and 3.

Some project topics that you can choose (if you cannot choose your own topic)

Problem 1. Synchronization: There is a deep canyon somewhere in Kruger National Park, South Africa, and a single rope that spans the canyon. Baboons can cross the canyon by swinging hand- over-hand on the rope, but if two baboons going in opposite directions meet in the middle, they will fight and drop to their deaths. Furthermore, the rope is only strong enough to hold three baboons. If there are more baboons on the rope at the same time, it will break. Assuming that we can teach the baboons to use semaphores, we would like to design a synchronization scheme with the following properties.

1. Once a baboon has begun to cross, it is guaranteed to get to the other side without running into a baboon going the other way.
2. There are never more than three baboons on the rope. The order of the baboons crossing the rope should be preserved; i.e., the order in which they enter the rope should be the order in which they exit the rope.
3. A continuing stream of baboons crossing in one direction should not bar baboons going the other way indefinitely (no starvation). Solve this requirement such that the FIFO order is preserved. That is, a baboon trying to cross to the left/right that arrives earlier than a baboon trying to cross in the opposite direction gets on the rope first.

Implementation Specifications

- Implement the behavior of the baboons in C using the Pthreads library.
- Make use of an input parameter that indicates the time required for a baboon to cross the canyon. Assume, all the baboons require the same time to cross the canyon.
- Overall, this problem deals with the access to a common resource, the rope. Access to the rope here means access for the threads to the critical section of your program, and how long a baboon takes to cross the canyon is indicated by the amount of sleep time (in seconds), which is the amount of thread sleep time in the critical section of your program.

Input

The input to the program should be provided in an input _le. The input _le name and the time (in seconds) required for a baboon to cross the canyon have to be command-line parameters. The format of the input file is given below.

- Line 1: L,R,R,R,R,R,L,L,R (a sequence of alphabets L (meaning left) and R (meaning right) separated by a comma, that indicate the side of the rope a baboon is trying to cross the rope from). Please note that, the delimiter here is a ',' (comma), and it does not appear after the last symbol in the first line of the file.

Problem 2. Sockets: The requirement is to develop a writer and a server program that communicates with the writers (identical) by making use of sockets. Three identical writer programs (except for the names, writer 1, writer 2, and writer 3, respectively) will run on one or more machines and the server on another machine.

Assignment Specifications

- Each writer program connects and sends a message (i.e. message\writer1" will be sent to the server from writer 1).
- The server should then accept the socket connection from a writer, and create a thread (child) after the socket is accepted.
- Once the thread is created, the server is also responsible for the communication (Step 4) with the writer, and then the parent process (in the server) should loop back to listen for other incoming connections - this makes the parent ready for other connections.
- A message from the writer should be stored by the appropriate thread processes in shared memory. This memory is shared by the server and all its children, and the child threads (in the server) can store only one message. The message length will be no more than 15 characters (14 chars plus 1 char string terminator in C), such as \writer1", \writer2" and \writer3".
- A writer process can be invoked (after compiling using the make file provided) with the writer name. For instance, suppose the writer program name is \writerp", the writer should be started by typing \writerp" that will make the writer read the message from the file \input file.txt" (the input file should be present in the same directory as the writer program).
- The shared memory must be protected (critical section) so that the stored message will never get corrupted during the concurrent connections from multiple writers.
- Once a message from a writer is successfully stored in the shared memory by one of the server's (child) threads, after a 2 second sleep, the actual contents of the shared memory should be sent back to the original writer by the same server (child) thread.
- Please set the server up to run on any C4 Lab PC. The server should loop for a total of n connection accepts; where n will be three in our trials.

The writer prints a message just before sending the message to the server. The server should then print the contents of the shared memory (after reading the incoming message into shared

memory) and send back the contents of the shared memory to the writer (acknowledgement). As soon as the writer receives the message

Problem 3. A Simulator for Banker's Algorithm

The simulator herein described uses the common process deadlock avoidance algorithm known as Banker's Algorithm to categorize process execution sequences as "safe" and "unsafe." Based on user given input, the simulator runs through all possible, process permutations and applies the steps given in Banker's Algorithm to determine whether each sequence is safe or not.

Parameters required by Banker's Algorithm are Available, Max, Allocated, and Need. Available is the quantity of resources that is initially available to all the processes. Max is the top number of resources that each process may hold at any time. "Allocated" is the number of resources initially held by each process. Need is the difference between Max and Allocated for each process and is used to determine the most number of resources a given process could potentially request.

Problem 4. Measuring the Performance of Page Replacement Algorithms

In this project you are to evaluate how applications respond to a variety of page replacement algorithms. For this, you are to write a memory simulator and evaluate memory performance using traces from real or simulated applications.

Simulator Requirements

Your job is to build a simulator that reads a memory trace and simulates the action of a virtual memory system with a single level page table. Your simulator should keep track of what pages are loaded into memory. As it processes each memory event from the trace, it should check to see if the corresponding page is loaded. If not, it should choose a page to remove from memory. If the page to be replaced is "dirty" (that is, previous accesses to it included a Write access), it must be saved to disk. Finally, the new page is to be loaded into memory from disk, and the page table is updated. Assume that all pages and page frames are 4 KB (4096 bytes).

Of course, this is just a simulation of the page table, so you do not actually need to read and write data from disk. Just keep track of what pages are loaded. When a simulated disk read or write must occur, simply increment a counter to keep track of disk reads and writes, respectively.

Implement the following page replacement algorithms:

1. FIFO: first in first out.

2. LRU: least recently used.
3. LFU: least frequently used.
4. MFU: most frequently used.
5. OPT: optimal page replacement.

Project 5. Dining Philosophers Problem

The dining philosopher's problem shows the problems inherent in resource allocation when there are fewer resources than processes that require the resources. In the problem, there is an x number of philosophers and an x-1 number of chopsticks. Each philosopher needs two chopsticks to eat from the bowl of rice in the middle of the table. The situation requires deft handling to avoid starvation. We will explore using an arbitration solution to the problem in C, but may have to change the solution based on success in implementing the program.

Project 6. Parallel Merge Sort Using Shared Memory

Run each recursive call on a child process.

Project 7. Writer-reader problem (Version 1)

Project 8. Writer-reader problem (Version 2)

Project 9. A runway of airplanes at an airport. The runway can have only one plane on it at one time. There are two types of planes: planes taking off and planes landing. For each plane that takes off, one must land. As well, planes must take off in the order in which they enter the takeoff queue. Planes must land in the order in which they enter the landing queue. Use semaphores to design a synchronization scheme with some properties.