Birla Institute of Technology & Science, Pilani Hyderabad Campus

CS C372/IS C362: Operating Systems

Assignment 2(Synchronization, Deadlocks and Starvation)

Assigned: 06.10.2012 Time of Submission: 29.10.2012 by 12:00 noon

Q1: Problem on shared memory / global variable based on semaphore.

Use Case: Train reservation system

There are two train stations S1 and S2. Train T1 runs from S1 to S2, while train T2 runs from S2 to S1. Both Trains have 3000 regular berths available initially, and 500 Tatkal quota berths.

Reservations are allowed 120 days in advance and assume a random request for 4000 regular seats, 750 tatkal seats for both the trains. Tatkal reservation is only allowed in the last 48 hours. Half the reservation requests would be for 2 seats (inclusive of Tatkal). In all 20% of the successful regular reservations are cancelled in the week before train departure. Maximum of 20 people may be on in Waitlist at any time (including at chart preparation).

End of every day, berth availability status is appended for each train in a separate file. Reservation requests and cancellations are allowed up to 60 minutes before start of train. Chart is to be prepared 30 minutes before departure.

Assume: Trains leave exactly at midnight in both directions. Create program(s) that runs within 120 seconds – simulating linear time of 120 days.

Implementation hints: Use global data structures (one per train) to record the bookings. Minimize the size of critical section. Each successful booking should print the Berth Number allotted. For reservation requests, create realistic load compressed by a factor of approximately 86,000 (1 day is 86,400 seconds), in a separate thread(s) running at higher priority. You may choose to have different threads for regular, tatkal and cancellations.

Each booking request should output the following:

Book <Req #> <Train #> <Num. of Berth >: [Success/Fail] : <Berth(s)# allotted on success>

Cancellations will include an existing allotted berth and out is of the format:

Cancel <Req #> <Train#> <Berth Number(s)>: [Success/Fail]

Code should save the final reservation chart in a proper format. Do not worry about the passenger names, age etc. Primary information to track is Booking Request Number and its status (including allotted Berths).

Submission: Include output, daily berth availability status and final reservation charts along with code and compiled binary.

Q2: Problem based on deadlock and starvation.

Create a program using POSIX threads having M threads running at different priorities. There are files (say N files with M >N) which are shared among these M threads. If one thread is using the file (assuming that it writing onto the file), no other thread will be allowed to use it. The code for your

thread has to maintain a Table that tells which file it has acquired and for which file it's requests are pending. The table may look like the below table:

File	Presently With	Needed By (Requested At in seconds)
1	TNum1	TNum3(30),Tnum5(70)
2	..	..
..	..	..

It may so happen that a deadlock occurs: say thread1 has acquired file1 and file2 and requesting for file3 which is already acquired by thread2 which is waiting for file1. Now, you have to create one more thread (Monitor) which will check for deadlock after every 5 secs and if deadlock is detected it will use deadlock recovery mechanism to break the deadlock. For that your code has to always maintain a shadow copy of the original file before granting access to a thread. As and when a thread completes, you can remove the shadow copy or you can use it for recovery. The shadow copy can be used to undo the updates in-case of deadlock recovery (process rollback). Typically the processing of the file involves appending a line including the thread number to the file and a timestamp. If the file is help by a thread for 6 seconds, 6 such lines should appear.

Note that the usage time for a file by a thread is randomly between 2-6 seconds. On an average each running thread will request a file after running between (3*M)/N to (7*M)/N seconds.

Moreover the Monitor should detect starvation such that no thread has an outstanding request for more than 300 seconds. To avoid starvation, the monitor will increase the priority of the starved thread to the maximum priority in the "Needed By" queue. After the thread is out of starvation original priority should be restored. Log change of priorities.

For efficiency you may simulate the code to run at 100X – 1000X speed.

Submission: Include the output of sample runs where both deadlock and starvation is observed. You may add more logs in your application to ensure that "Rollbacks" and "Starvation Avoidance" is adequately observed. Also include the binary and you code.

Submission Instructions: Maintain same grouping as assignment 1. All your programs must run over either Fedora core or Ubuntu machines available in IPC Terminal 1 of Computer Science block. Submit source files and executables. Also submit a readme.txt with your group details. Put all your deliverables into a tar file (like, f2012023.tar) and send it to OS2012@bits-hyderabad.ac.in as a mail attachment. Also, please keep a backup of your work in the machine that you have used from the lab to implement the assignment. Copied codes will be awarded zero marks. Every assignment will have a demo/viva session which will be intimated later through a separate notice.

CSIS N/B, 06.10.2012

Instructor In-Charge
CS C372/IS C362

(Pl. remove on 15.10.2012)