CMPT 300 **Project 1** **Due: Dec. 1st, 2003, 4:30pm**

Place in assignment drop-in boxes

Requirements

- It is a team project. Your team could have up to 3 students.
- All answers must be computer generated (including text and diagrams), single-side printed, and the text must be in NO smaller than 11-point font with AT LEAST 25-mm margins (left, right, top, bottom).
- The hand-in version must be ordered correctly and stapled in the top left corner.
- The hand-in version must include a header page indicating: student names, student numbers, user ids, and the operating system (UNIX or Linux) your programs work on..
- Submit all source codes and makefiles through online submission server. Make sure your programs and makefiles work correctly on the Linux or UNIX machines in CSIL lab.

The purpose of this assignment is for you to observe the effects of uncontrolled concurrent access to a shared resource (in this case the variable *seats_available*), and then to provide synchronization to enforce controlled execution to access the variable.

Part A:

Find appropriate documentation, which discusses *pthreads*, and hence learn about this package.

Hand-in

Please list 3-5 sources that you found most informative and helpful to learn about programming using *pthreads*. The sources can be web pages, books, and any information sources publicly accessible. For each source, please give a short summary and comment on its goodness and badness. The summary and comment should not exceed 10 lines for each source.

Marks: 5

Part B:

Use facilities from the *pthreads* package in the following programs. You are NOT allowed to use any other synchronization mechanism not in *pthreads*.

You are to design and write 3 programs which simulate the following travel agent booking problem:

Air SN has just instituted flights, on a Saab XJK turbo-prop seating 36 passengers, from Boston to Halifax and return. Consider Air SN's flight 888 which departs Halifax non-stop for Boston at 8:08 am on February 18, 2002. Passengers may book (and cancel) seats on this flight by contacting any recognized travel agent.

Each program should:

a) create 6 concurrent threads, with each thread representing a travel agent.
b) create a (shared) integer variable *seats_available* - which will represent the number of seats (still) available on the flight (and thus hence initialize it to 36).
c) each thread should read *seats_available*, then print both the current value of *seats_available* and the thread's ID. Each thread will think a while (1-2 seconds in random), then either increment (indicating that a passenger is cancelling his/her seat) or decrement (indicating that a seat is being booked) *seats_available* by 1. For this simulation, it is assumed that the ratio of seat bookings to cancellations is 3:1. i.e., 75% of the time a thread (i.e. travel agent acting on a passenger's instructions) will book a seat and 25% of the time will cancel a seat.
d) each thread repeats c) until *seats_available* reaches 0 (i.e. the flight is sold out - Air SN is determined to make a mint on this routing!!), at which point it terminates.
e) the program terminates when all threads come to an end.

Program 1:

is to illustrate uncontrolled concurrent access to the shared resource, *seats_available*.

Compile & run your program (several times) & see if more than 1 thread prints the same value of *seats_available*, or if other 'strange' happenings occur. If not, make a thread relinquish the CPU just before decrementing / incrementing *seats_available*, and introduce a random delay when a thread regains the CPU. Adjust the delays until some value is printed more than once.

Your assignment report should include: commented source code with makefile (submitted online), at least 2 sample runs annotated with explanations as to why you got those 'strange' results which you did. Also include a discussion of why the 2 runs are different in output.

Marks: correctly working program	20
explanations & discussions	10

Program 2:

is to illustrate controlled concurrent access to the shared resource, *seats_available*.

Use an appropriate (& available) synchronization mechanism to force your processes to execute in a controlled concurrent manner so that *seats_available* always maintains integrity of data as seats are booked and cancelled.

Again provide properly documented source code with makefile (submitted online), and 2 sample runs. Also provide a brief discussion of how your synchronization mechanism achieves its goal.

Marks: correctly working program	20
explanations & discussions	5

NOTE:

1) No mention has been made about the erroneous possibility of *seats_available* taking on a value 36. It is left to you to design a solution to this possibility - you must discuss / illustrate this design in the documentation comprising your assignment report.

Marks: 5

2) You need not make this simulation more complex than that described, i.e. you need not worry about booking multiple seats in one go; you need not consider booking requests matched to assigned seating nor passenger names; etc

Program 3:

is to enhance the functionality of Program 2.

It takes 1 second to book a seat, i.e., after the *seats_available* is incremented or decremented, the next request will be taken after 1 second. The customers are in two classes: frequent flyers and regular customers. Frequent flyers have higher priority in booking. 30% of the customers are frequent flyers. When two or more customers with the same priority are waiting for the booking, a first-come-first-serve policy is supposed. Once an agent decides to book or cancel a seat, it waits until the request is satisfied.

Write a program to simulate this scenario.

Again provide properly documented source code with makefile (submitted online), and 2 sample runs. Also provide a brief discussion of how your synchronization mechanism achieves its goal.

Marks: correctly working program 15
explanations & discussions 5