Systems Programming - INB(N)365

Assignment 1: Processes and Threads

**Group Working:** No group work is permitted. This is an individual assignment.

**Due Date:** Friday week 7 (3/9/2010)

**Weighting:** 20% of the assessment for INB(N)365

**Hardware:** STUDENT LABORATORY PCs

**Software:** VMware player, Ubuntu and GCC

**OBJECTIVES:** To explore how threads may be implemented in C.

To appreciate multi-threading when processing user input.

To understand how threads communicate via shared memory.

To understand the process synchronization.

**REFERENCE MATERIALS:**

1. Your lecture notes.
2. Textbook: Silberschatz, Abraham; Baer Galvin, Peter and Gagne, Greg, (2009) Operating System Concepts, 8th Edition. New York: John Wiley & Sons, (ISBN: 978-0-470-12872-5).
3. The Source Code of the tutorials about thread programming on the Blackboard.

**REQUIRED TO BE SUBMITTED:**

**In Hardcopy:**

In a single bound document, you are to provide the following:

1. A statement of completeness outlining the tasks you attempted in your assignment, deviations from the assignment specifications, and any known or potential logic errors that might exist.
2. A statement of the contribution of team members.
3. One page descriptions about the data structures and the high level design for Task 1.
4. Output listing as supporting evidence that your C program works for Task 1.
5. Your answer to Task 2 (for INN365 students only).

**On a USB stick:**

On a USB stick, you are required to supply the following files:

1. Assignment1.doc must contain:
   1. Statements of the contribution of team members and the completeness.
   2. One page descriptions about the data structures and the high level design for Task 1.
   3. Output listings as supporting evidence that your program code works for Task 1.
   4. A short user manual (half page) that will be used by your tutor to test your code for Task 1.
   5. Your answer to Task 2 (INN365 students only).

Please include this file in your USB stick even though you will have to submit the required documents in **hardcopy** as well.

1. All source code for your solution to Task 1 must be in the top-level folder “C”. It should contain all C files (\*.c), header files (if any) and the batch file used to compile and run your code. Include the ***make*** file if you have used it for compiling your C programs.
2. The binary executable file.

You must keep a copy of all your solutions at home including Assignment1.doc, all source code and the binary executable file(s). Please submit your assignments directly to Assignment Minder (<http://www.am.qut.edu.au/>).

Failure to submit these files will prevent your assignment from being marked.

#### **TASK 1 – C Threading** (20/20 for INB365 students and 14/20 for INN365 students)

*You need only attempt this task if you are INB365 students. For INN365 students, you need attempt both Task 1 and Task 2.*

*You can write the code using a single or multiple files.*

Phoenix is a small car park in the center of a beautiful city. It has a fixed number of places with two entrances and one exit. The entrances and exit are narrow so that only one car can pass at a time. Cars might arrive or depart at any moment. No car can enter when the car park full.

You are required to write a multi-threaded C program to simulate the Phoenix car park based on the producer/consumer pattern that you have learnt. You are to implement some threads of execution in your program. You may design two arrival threads (one for a waiting queue, and another for the car park), to emulate the car arrival task, and one departure thread to emulate the continuous departure task. Another thread, the *monitorThread*, will listen for user input, perform some administration task such as displaying the current cars in the car parks, and terminate the program. The *main* thread of this program will simply be used for creating an instance of the car park management system, within which the other threads will be created and started to run.

The *arrival thread* and *carparkEnter thread* continuously create cars and park them into the carPark. To make it simple, we assume that each car is identified by a unique car ID (a string similar to XY01234567, where XY can be any uppercase characters and 01234567 can be any integer sequence of eight (8) digits). To effectively use both entrances, arrival cars will be put in a waiting queue firstly by the *arrival thread*, and then it will be sent to an entrance by the *carparkEnter thread* if it is at the head of the queue and the car park is not full. If the car ID is an even number, it will be sent to entrance 2, otherwise it will be sent to entrance 1.

The carPark may be implemented using an Array or LinkedList structure. The carPark is to be bounded i.e. only a fixed number of parking bays may be available. If the car park is full, the *carparkEnter thread* "blocks" until there is a place available. Please note that you should print out a message like “No parking bays available. Arrival blocked” in this case. If the waiting queue is empty, the *carparkEnter thread* also blocks until there is at least a car in the queue. Please note that you should print out a message like “waiting queue is empty” in this case. If the thread tries to park a car with the same ID as a car currently in the car park, a proper message should be printed out.

The *departure thread* randomly selects a car and tries to remove it from the carPark. It also needs to work out the time of the car staying in the car park. If there are no cars in the carPark, the *departure thread* "blocks" until the carPark contains at least one car. A message such as “Car park empty. Departure blocked” should be printed out in this situation. If the thread tries to remove a car that does not exist in the car park, a proper message should also be printed out.

The *carparkEnter thread* and the *departure thread* share a common data structure CarPark. The CarPark will be managed by the addCar and removeCar functions, which are then called by the *carparkEnter thread* and the *departure thread* respectively.

The *monitorThread* listens to the keyboard for input from the user and controls the termination of this program. The program will be monitored by a console (text) terminal (a single window). It can only be terminated by pressing the ‘q’ or ‘Q’ key. If the user presses the ‘q’ or ‘Q’ key, the program quits (the monitor thread stops the arrival and departure threads then stops itself **gracefully**). If ‘c’ or ‘C’ is entered, the program will print out a list of cars currently in the car park (hint: this functionality may be implemented by checking every 25 milliseconds for the user pressing the keyboard).

Any code you write should be succinctly commented. Although a simple skeleton file is provided for your convenience, you are not required to use it.

As the synchronization issue will have been taught by the due date of this assignment, **synchronization** should be implemented in this assignment wherever necessary. This means you need to use the **synchronization primitives** provided by the **POSIX Threads API**. You need to get the program running, manage the synchronization properly, avoid hanging and generate the correct result.

The program will be console based. No bonuses will be given for fancy or complex user interfaces. Remember to include “printf” in C statements to show what is happening wherever required.

#### **Sample Output:**

Welcome to the simulation of a Car Park Management System.

Press 'C/c' (followed by return) to show car park status

Press 'Q/q' (followed by return) to quit program

<--- Press return to continue --->

arriving car -> FS 32253741 to the waiting queue at [2009-08-10 18:17:43.616]

There are some cars in the enter waiting queue.

arriving car -> FS 32253741 from entrance 1 at [2009-08-10 18:17:43.617]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

departing car -> FS 32253741 at [2009-08-10 18:17:43.617] (stay 0hs-0ms-0ss-0mmms)

Car park has vacancies now.

arriving car -> NV 70542495 to the waiting queue at [2009-08-10 18:17:44.617]

There are some cars in the enter waiting queue.

!-!-!-!-! The car park is empty. !-!-!-!-!

arriving car -> NV 70542495 from entrance 1 at [2009-08-10 18:17:44.618]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

departing car -> NV 70542495 at [2009-08-10 18:17:44.618] (stay 0hs-0ms-0ss-0mmms)

Car park has vacancies now.

arriving car -> XB 30235537 to the waiting queue at [2009-08-10 18:17:45.619]

There are some cars in the enter waiting queue.

arriving car -> XB 30235537 from entrance 1 at [2009-08-10 18:17:45.620]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

arriving car -> YM 75185195 to the waiting queue at [2009-08-10 18:17:46.621]

There are some cars in the enter waiting queue.

arriving car -> YM 75185195 from entrance 1 at [2009-08-10 18:17:46.622]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

arriving car -> GA 70365807 to the waiting queue at [2009-08-10 18:17:47.624]

There are some cars in the enter waiting queue.

arriving car -> GA 70365807 from entrance 1 at [2009-08-10 18:17:47.625]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

departing car -> GA 70365807 at [2009-08-10 18:17:48.624] (stay 0hs-0ms-1ss--1mmms)

Car park has vacancies now.

departing car -> XB 30235537 at [2009-08-10 18:17:50.785] (stay 0hs-0ms-5ss-165mmms)

Car park has vacancies now.

arriving car -> RE 32358208 to the waiting queue at [2009-08-10 18:17:50.795]

arriving car -> RE 32358208 from entrance 2 at [2009-08-10 18:17:50.795]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

There are some cars in the enter waiting queue.

departing car -> RE 32358208 at [2009-08-10 18:17:52.786] (stay 0hs-0ms-2ss--9mmms)

Car park has vacancies now.

arriving car -> MC 25172043 to the waiting queue at [2009-08-10 18:17:52.795]

arriving car -> MC 25172043 from entrance 1 at [2009-08-10 18:17:52.796]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

There are some cars in the enter waiting queue.

departing car -> MC 25172043 at [2009-08-10 18:17:53.788] (stay 0hs-0ms-1ss--8mmms)

Car park has vacancies now.

arriving car -> VS 43188693 to the waiting queue at [2009-08-10 18:17:53.798]

arriving car -> VS 43188693 from entrance 1 at [2009-08-10 18:17:53.798]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

There are some cars in the enter waiting queue.

departing car -> YM 75185195 at [2009-08-10 18:17:54.790] (stay 0hs-0ms-8ss-168mmms)

Car park has vacancies now.

arriving car -> KL 99582437 to the waiting queue at [2009-08-10 18:17:54.811]

arriving car -> KL 99582437 from entrance 1 at [2009-08-10 18:17:54.811]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

There are some cars in the enter waiting queue.

departing car -> VS 43188693 at [2009-08-10 18:17:55.792] (stay 0hs-0ms-2ss--6mmms)

Car park has vacancies now.

arriving car -> GC 64024134 to the waiting queue at [2009-08-10 18:17:55.812]

arriving car -> GC 64024134 from entrance 2 at [2009-08-10 18:17:55.812]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

There are some cars in the enter waiting queue.

arriving car -> WY 56021690 to the waiting queue at [2009-08-10 18:17:56.813]

arriving car -> WY 56021690 from entrance 2 at [2009-08-10 18:17:56.814]

There are some cars in the car park.

!+!+!+!+! The enter waiting queue is empty. !+!+!+!+!

There are some cars in the enter waiting queue.

[2009-08-10 18:17:58.290] <--- Key [c] has been pressed --->

[2009-08-10 18:17:58.290]

\*\*\* Phoenix Car Park \*\*\*

----------------------------------------

car 01) KL 99582437 arrive at [2009-08-10 18:17:54.811]

car 02) GC 64024134 arrive at [2009-08-10 18:17:55.812]

car 03) WY 56021690 arrive at [2009-08-10 18:17:56.814]

3 car(s) in total, 7 place(s) available

----------------------------------------

departing car -> GC 64024134 at [2009-08-10 18:17:58.794] (stay 0hs-0ms-3ss--18mmms)

Car park has vacancies now.

arriving car -> RM 32853523 to the waiting queue at [2009-08-10 18:17:58.817]

arriving car -> RM 32853523 from entrance 1 at [2009-08-10 18:17:58.817]

There are some cars in the car park.

[2009-08-10 18:18:20.713] <--- Key [q] has been pressed --->

[2009-08-10 18:18:20.713] <--- Quitting Program --->

**TASK 2** (6/20 for INN365 students)

*This task is only for INN365 students. You do not need to attempt this task if you are INB365 students.*

The first known correct software solution to the critical-section problem for *n* processes with a lower bound on waiting of *n* − 1 turns was presented by Eisenberg and McGuire. The processes share the following variables:

enum pstate {idle, want\_in, in\_cs};

pstate flag[n];

int turn;

where, “want\_in” denotes the process is ready, “in\_cs” means the process is in its critical section, and “idle” means idle state.

All the elements of flag are initially idle; the initial value of turn is immaterial (between 0 and *n*-1). The structure of process *Pi* is shown in Table 1.

1. Give a brief explanation of what the algorithm does to ensure mutual exclusion.
2. Prove that the algorithm satisfies all three requirements for the critical-section problem.

|  |
| --- |
| do {  while (TRUE) {  flag[i] = want\_in;  j = turn;  while (j! = i) {  if (flag[j] != idle) {  j = turn;  else  j = (j+1) % n;  }  flag[i] = in\_cs;  j = 0;  while ((j<n) && (j==i || flag[j] != in\_cs))  j++;  if ((j>=n) && (turn == i || flag[turn] == idle))  break;  }  // Critical section    j = (turn + 1) % n;    while (flag[j] == idle)  j = (j + 1) % n;    turn = j;  flag[i] = idle;  // Remainder section  } while (TRUE) |
| **Table 1.** The structure of process *Pi* |

**An Overall** **Marking Guide**

Task 1 will be marked according to the design and the correct implementation of the specified functions and a good programming style, including:

* Correct and running code
* Clarity of code (suitable indentation etc.)
* Appropriate design and comments
* Good documentation

Please keep these issues in mind during the design and implementation of this assignment.

Task 2 will be marked on the correctness of relevant concepts and descriptions. No running code is required.

A detailed marking sheet with feedback will be returned to students once the assignments are marked.

**END OF ASSIGNMENT 1**