

## COMP2310-2002-03

### Assignment 2: Think about it

This assignment requires you to *think*, not hack away in C or Java. Indeed, although it is possible to try out your solutions to these problems on a computer, you should work (i.e. think) on the basis that it is not.

### Proposed due date: 11.59 p.m. EST, Friday 11 October 2002

This assignment is worth 15% of your final mark. It will be marked out of 20.

If you are having problems, seek help *early*! I will answer questions as necessary during lecture time, and, of course, on the newsgroups.

**Part Two of this specification is on the web page. It contains an up-to-date list of corrections, hints and tips for getting the assignment done. You should refer to it frequently! The web page also contains a link to the marking guide that the tutors and I will use to assess your work.**

## 1 A locking problem

Imagine that you are using a CPU that has a special instruction called FROBNITZ (specified here using a C-like syntax, but it executes atomically, i.e. as *one* machine instruction):

```
int FROBNITZ(int a, b, c)
{
    if (a == c) { c = b; return 0; }
    else { a = c; return 1; }
}
```

- Using the FROBNITZ instruction, develop a solution to the mutual exclusion problem for  $n$  processes. (In other words, specify a pre- and post-protocol that can be used around a critical region.) Don't worry about avoiding individual starvation. Explain your solution clearly and why it is correct.
- Is this instruction powerful enough to develop a fair solution to the mutual exclusion problem (i.e. that avoids individual starvation)? If so, give one and explain it. If not, explain why not.
- Find a real CPU (that has been commercially manufactured in large quantities) that has an instruction which behaves just like FROBNITZ. Give the name of the CPU, the name of the instruction, and give a reference that shows where you got the information.

## 2 The sightseeing problem

Suppose there are  $n$  tourist processes and one coach process. The tourists repeatedly wait to go on a sightseeing tour in the coach, which can hold  $C$  tourists,  $C < n$ . However, the coach only departs when it is full.

- (a) Develop code for the actions of the tourist and coach processes. Use semaphores for synchronization. Explain your solution.
- (b) Generalize your answer to (a) to employ  $m$  coach processes,  $m > 1$ . Since the coaches take the same route along a one-lane road, coaches can not pass each other; i.e., they must finish going around the route in the order in which they started. As in (a), a coach can depart only when it is full. Explain your solution.

### 3 A monitor problem

Read Hoare's paper on monitors (in the reading brick). In this paper, the 'signalling discipline' (i.e. the semantics of the `signal` call) is *Signal and Urgent Wait*: the process doing the signal is suspended, and the signalled process now continues execution. But there is another possible implementation – *Signal and Continue* – where the process that is signalled is unblocked, but does not yet resume execution: the process that called `signal` continues, and the signalled process is woken up at some later stage (perhaps when the signalling process itself calls `wait`, or exits the monitor).

Assume one producer process and  $n$  consumer processes share a bounded buffer having  $b$  slots. The producer deposits messages in the buffer; consumers fetch them. Every message deposited by the producer is to be received by all  $n$  consumers. Furthermore, each consumer is to receive the messages in the order they were deposited. However, consumers can receive messages at different times. For example, one consumer could receive up to  $b$  more messages than another if the second consumer is slow.

Develop a monitor that implements this kind of communication. Use the Signal and Continue discipline.

### 4 A message-passing problem

Consider a distributed system with  $n$  processes, labelled as  $P_0$  to  $P_{n-1}$ , each corresponding to a node in a connected, undirected graph. Each node (process) can communicate only with its neighbours. Each process has a local array `linked`, also indexed from 0 to  $n - 1$ . In process  $P_i$ , the value of `linked[j]` is 1 if  $P_i$  and  $P_j$  are neighbours, and 0 otherwise. (In process  $P_i$ , the value of `linked[i]` is 0.) The problem is for each process to pair itself with one of its neighbours. When the processes finish pairing up, each process should be paired or single, and no two neighbouring processes should be single.

Solve this problem using asynchronous message passing for communication. (You have access to `send` and `receive` operations that can send and receive integer values.) Every process is to execute the same algorithm. When the program terminates, every process should have stored in local variable `pair` the index of the neighbour it is paired with; the final value of `pair` should be  $i$  if process  $P_i$  is single. Your solution need not be optimal, in the sense of minimizing the number of single processes (which would make the problem very difficult).

### 5 Reflection

Reflection is one of the most important components of the 'thinking' part of your life. Include short answers to the following questions in your report. Please take this part seriously; I will.

- (a) (You should track the time you spend doing this assignment.) How long did it take you to (a) find the background information you need, from the reading brick and the web, and (b) solve each of the problems? (Do not include time spent in front of the TV.)

- (b) Which questions, if any, did you find easy? Why?
- (c) Which questions, if any, did you find difficult? Why?
- (d) What, if anything, have you learned by doing this assignment?

## 6 Your submission

### Do read this section carefully!

You will write a  $\text{\LaTeX}$  file `think.tex` and a `Makefile`. You will be penalized if your `Makefile` is broken and I have to fix it. It would therefore be a good idea to pay careful attention to this section. **Hot tip:** before submitting, copy all the files you are going to submit into an empty directory and test your `Makefile`.

Your `Makefile` must specify (at least) one target as follows. The command `make think.dvi` should produce a DVI file `think.dvi`.

You will be penalized (up to two marks, i.e. 10%) for any errors in spelling or grammar. Running the spelling checker is essential, but not sufficient! **There is an absolute page limit of 15 pages, not counting cover page, table of contents, or indexes. You will be penalized if you exceed this limit.**

Use pictures and diagrams where appropriate. You may produce them in  $\text{\LaTeX}$ , or use `xfig` to generate Encapsulated PostScript for inclusion with the `\includegraphics` command. (Any EPS files should have a `.eps` file suffix.) **Hot tip:** there aren't marks specifically allocated for 'descriptive aids', but they may help the reader to understand better if your text isn't completely clear.

Submit your assignment with the command:

```
mark comp2310 think think.tex Makefile *.eps
```

(make sure that you submit all of your PostScript figures!).

## 7 Acknowledgement

I got the ideas for these questions from Greg Andrews's book 'Foundations of Multithreaded, Parallel, and Distributed Programming'.

## 8 And last of all ...

Have fun!