Informatics and Mathematical Modelling
Technical University of Denmark
Building 321
DK–2800 Lyngby
Denmark

HHL 25–08–2008

02152 CONCURRENT SYSTEMS

FALL 2008

Auxiliary Exercises

Petri Nets

Exercise Petri.1

Five boats shuttle between two jetties, A and C, via a jetty B. Jetty A and C each has a capacity of two boats, whereas jetty B has a capacity of three boats. Make a Petri Net model of the boat traffic.

Exercise Petri.2

Four actions A, B, C, and D are to be synchronized as follows:

> After execution of A, either B or C is executed. Concurrently with this, D is executed. All of this is repeated forever.

(a) Draw a Petri-net in which the actions A to D are represented by transitions and synchronized as described above.

(b) Which pairs of actions can be executed in parallel?

(c) Which interleavings (sequences of single transition firings) are possible for the first cycle of the execution?

Exercise Petri.3

Make a Petri Net for following process:

Root Galettes

1 leek	*2 large potatoes*	*salt, pepper*
2 carrots	*1 egg yolk*	*olive oil*
1 parsnip	*(1 tsp Maizena)*	

The leek is rinsed and finely chopped. The root vegetables are peeled and cut *en julienne*. Mix with egg yolk and optionally *Maizena* (cornflour). Season with salt and pepper and fry in hot oil like small pancakes, a couple of minutes on each side. Put on absorbing kitchen paper and serve.

Note: The paste is divided into 10 portions and there are two frying pans available.

Problem originally due to the Danish cook, Claus Meyer.

Exercise Petri.4

Draw the Petri Net $N = (P, T, F)$ where

$$\begin{aligned} P &= \{p_1, p_2, p_3\} \\ T &= \{t_1, t_2, t_3\} \\ F &= \{(p_1, t_1), (p_1, t_2), (p_3, t_2), (p_2, t_3), (t_1, p_2), (t_2, p_2), (t_2, p_3), (t_3, p_1)\} \end{aligned}$$

with the marking $M_0 = (2, 0, 1)$ (corresponding to (p_1, p_2, p_3)).

Write all *simultaneous firings* possible from M_0 using the notation $M_0 \xrightarrow{U} M'$.

Transition Systems

Exercise Trans.1

Find all interleavings of the two processes:

$$\begin{array}{ll} P_a: & a_1, a_2, a_3 \\ P_b: & b_1, b_2 \end{array}$$

Exercise Trans.2

Assume that two processes P_1 and P_2 consist of sequences of n_1 and n_2 actions respectively. Find an expression for the number of possible interleavings of P_1 and P_2.

Exercise Trans.3

Find all interleavings of the three processes:

$$\begin{array}{ll} P_a: & a_1 \\ P_b: & b_1, b_2 \\ P_c: & c_1 \end{array}$$

Exercise Trans.4

Assume that P_i is a process consisting of n_i actions. Find an expression for the number of interleavings of

$$P_1, P_2, \ldots, P_k$$

Exercise Trans.5

Draw *transition diagrams* for the two processes in the following program:

var X, Y : *Integer*;
$X := 1;\quad Y := 2;$

cobegin $X := Y + 1 \parallel Y := X - 1$ **coend**

Now, draw (the reachable part of) the *transition graph* for the full program. The nodes should be states of the form $(X, Y, t_1, t_2, \pi_1, \pi_2)$, where t_i are local variables and π_i are the control variables of the two processes.

From the graph, determine the possible final values of X and Y.

Shared Variables

Exercise Share.1 (Lock-step problem)

Write, using shared variables only, to pieces of program, $SYNC_A$ and $SYNC_B$, that synchronize to processes P_A and P_B such that they proceed in *lock-steps.* More precisely, if op_A and op_B are operations in P_A and P_B respectively, then the number of times these two operations have been executed must differ by at most one.

process P_A;	**process** P_B;
repeat	**repeat**
$SYNC_A$;	$SYNC_B$;
...	...
op_A;	op_B;
...	...
forever	**forever**

Exercise Share.2

In the following program, it is attempted to establish a critical region for two concurrent processes by using two shared boolean variables C_1 and C_2:

var C_1, C_2 : *boolean*;

$C_1 := \mathit{false}$; $C_2 := \mathit{false}$;

process P_1;		**process** P_2;	
repeat		**repeat**	
nc_1:	non-critical section$_1$;	nc_2:	non-critical section$_2$;
r_1:	**repeat**	r_2:	**repeat**
	a_1: $C_1 := \neg C_2$;		a_2: $C_2 := \neg C_1$;
	until $\neg C_2$;		**until** $\neg C_1$;
cs_1:	critical section$_1$;	cs_2:	critical section$_2$;
e_1:	$C_1 := \mathit{false}$	e_2:	$C_2 := \mathit{false}$
forever;		**forever**;	

(a) Draw the transition diagrams for P_1 and P_2.

(b) Show that the program does **not** ensure mutual exclusion.

(c) Assume that a_1 and a_2 are executed as *atomic statements* instead, i.e. a_1: $\langle\, C_1 := \neg C_2 \,\rangle$ and a_2: $\langle\, C_2 := \neg C_1 \,\rangle$.

Determine whether the algorithm now ensures mutual exclusion.

Exercise Share.3

Consider the problem of establishing a critical region using a coordinator process addressed in Andrews Ex. 3.12. A proposal for the form of the processes is:

process $P[i : 1..n] =$
  repeat
    non critical section$_i$;
    $enter[i] := true$;
    $\langle$ **await** $in[i] \rangle$;
    critical section$_i$;
    $in[i] := false$
  forever

(a) Write a proposal for the coordinator process.

(b) Express mutual exclusion among n process as an invariant.

(c) State and prove some auxiliary invariants of your program that may be combined to show mutual exclusion.

Hint: What can be said about $in[i]$ in the critical section? What is is known about the state of the coordinator process when $in[i]$ is true?

(d) Is your algorithm fair?

Theory (Safety and Liveness)

Exercise Theory.1

Consider the concurrent program:

$$
\begin{array}{l}
\textbf{var}\ x, y : \mathit{integer} := 0; \\
\\
\textbf{co} \\
\quad \textbf{repeat}\ a_1\!: \langle\, y < 2 \rightarrow y := y+1;\ \ x := y \,\rangle\ \textbf{forever} \\
\| \\
\quad \textbf{repeat}\ a_2\!: \langle\, x = 0 \rightarrow y := 0 \,\rangle\ \textbf{forever} \\
\| \\
\quad \textbf{repeat}\ a_3\!: x := 0\ \textbf{forever} \\
\textbf{oc}
\end{array}
$$

Question 1.1:

(a) Prove inductively that $I \triangleq 0 \leq x \leq y \leq 2$ is an invariant of the program.

(b) Draw the (reachable part of) the transition graph for the program. Since control remains at the a-actions, only the (x, y) part of the state needs be shown.

(c) Determine whether $\neg(x = 1 \wedge y = 2)$ is an invariant of the program.

Question 1.2:

(a) Argue that $\Box\Diamond x = 1$ holds for the program under the assumption of weak fairness.

(b) Show that $\Box\Diamond x = 2$ does not hold, even under the assumption of strong fairness.

Question 1.3:

(a) Assume that the action a_2 cannot be considered atomic as a whole.

Draw the transition diagram representing a_2 then and show that I is no longer an invariant of the program.

(b) In the original program, assume that the action a_1 is replaced by the refinement:

$$b_1\!:\ \textbf{await}\ y < 2;\quad c_1\!:\ t := y;\quad d_1\!:\ y := t + 1;\quad e_1\!: \langle\, x := y \,\rangle$$

where t is a local integer variable.

State a predicate H that implies I, holds initially, and is inductive for the program (i.e. strong enough to be preserved by all atomic actions).

Semaphores

Exercise Sema.1

Three processes P_1, P_2, and P_3 execute three operations A, B, and C respectively.

The operations are to be synchronized using semaphores as follows:

var SA, SB, SC : *semaphore*;

$SA := 0;\;\; SB := 0;\;\; SC := 0;$

process P_A;	**process** P_B;	**process** P_C;
repeat	**repeat**	**repeat**
A;	B;	$\mathtt{P}(SC)$;
$\mathtt{V}(SC)$;	$\mathtt{V}(SC)$;	$\mathtt{P}(SC)$;
$\mathtt{P}(SA)$	$\mathtt{P}(SB)$	C;
forever	**forever**	$\mathtt{V}(SA)$;
		$\mathtt{V}(SB)$;
		forever

Draw a Petri net in which the operations A, B, and C are synchronized the same way as in the above program. In the net, the operations must occur as transitions.

Exercise Sema.2

Recall the problem and solution to Exercise Petri.2.

Now, the four operations/actions are to be executed by four sequential processes P_A, P_B, P_C, and P_D respectively. Write a program using semaphores to synchronize the four processes such that the operations A to D are synchronized as in the Petri-net. (The choice of which operation to execute, B or C, need not be fair, and can be left to the semaphore mechanism.)

Exercise Sema.3

The meeting problem (barrier problem, lock-step problem) for two processes has been solved in Section 3.6 in [Basic] using general semaphores.

(a) Show that this solutions does **not** work with *binary semaphores*.

(b) Solve the meeting problem for two processes using binary semaphores only. Use the semaphore invariant to show that binary semaphores are sufficient.

Exercise Sema.4

Solve the meeting problem for three processes using semaphores.

Exercise Sema.5

Write a piece of code $SYNC_i$ that solves the meeting/barrier problem for an arbitrary number of processes N using semaphores only.

Monitors

In all the below exercises you should assume Signal-and-Continue (SC) semantics of condition queues unless otherwise stated.

Exercise Mon.1

Write a monitor with two procedures $SYNC_A$ and $SYNC_B$ to be used by two processes P_A and P_B respectively. The monitor should synchronize the two processes, i.e. make them meet/wait for each other.

Exercise Mon.2

Now, the above problem is generalized to making N processes meet before any of them can proceed (also known as the *barrier problem*). Write a monitor with a single prococedure $SYNC$ to be used by all the processes for the synchronization.

Exercise Mon.3

The general meeting problem from the preceeding exercise is now to be modified such that the N processes not only meet but also "share the loot". I.e. each process comes with a number (given as a parameter to $SYNC$) and get the mean value of all numbers back (as a return value). Write a monitor that solves this problem. Beware that due to the SC semantics, processes may call the monitor again before all have got their share of the loot.

Hint: Use an extra "pre-queue" where processes may be delayed while the sharing takes place.

Exercise Mon.4

Write a monitor with two operations *sleep* and *wakeup* that implements the synchronization mechanism of Andrews Ex. 4.6.

CSP

From Concurrent Programming Exam, June 1994

PROBLEM 3 (approx. 15 %)

Three CSP processes P_1, P_2, and P_3 perform three operations A, B, and C respectively. The operations are to be synchronized which is accomplished by communication among the processes:

process P_1 =	**process** P_2 =	**process** P_3 =
repeat	**repeat**	**repeat**
$P_2\,!\,()$;	**if** $P_1\,?\,() \rightarrow$ **skip**	$P_2\,!\,()$;
A	$[\,]$ $P_3\,?\,() \rightarrow$ **skip**	C
forever	**fi**;	**forever**
	B	
	forever	

Question 3.1:

Draw a Petri-net in which the three operations A, B, and C are synchronized the same way as in the CSP program. In the net, the operations should be represented by transitions.

The operations are now to be executed by three sequential processes P_A, P_B, and P_C:

process P_A =	**process** P_B =	**process** P_C =
repeat	**repeat**	**repeat**
A	B	C
forever	**forever**	**forever**

Question 3.2:

Show how semaphores can be used to synchronize the three processes such that A, B, and C are synchronized in the same way as in the CSP program.

Deadlocks

From Concurrent Programming Exam, December 1998

PROBLEM 4 (approx. 10 %)

In a system there is one instance of a resource type A, two instances of a type B, and three instances of a type C. The resources are used by four processes P_1, P_2, P_3, and P_4. The processes have declared their maximal resource demands as shown below. Furthermore, it is shown which resources have been allocated and which are requested at a certain moment.

	Max		
	A	B	C
P_1	1	2	0
P_2	0	1	1
P_3	1	0	3
P_4	0	2	2

	Allocation			*Request*		
	A	B	C	A	B	C
P_1	1	0	0	0	0	0
P_2	0	1	0	0	0	0
P_3	0	0	1	0	0	1
P_4	0	1	0	0	0	1

Question 4.1:

Draw a resource allocation graph corresponding to this situation. In the graph, the expected, not yet requested, resource needs should be indicated by dashed arrows.

Question 4.2:

(a) Show that the situation is safe.

(b) Determine whether P_4 can be granted the requested C-instance according to the banker's algorithm.

FURTHER SELECTED EXAM PROBLEMS

From Concurrent Programming Exam, December 1998

PROBLEM 1 (approx. 25 %)

The below implementation of a critical region for two processes, P_1 and P_1, utilizes a machine instruction that indivisibly sets an integer variable to the value of another variable plus one. In the program this instruction is denoted by statements of the form $<X := Y+1>$. The variables C_1 and C_2 are not changed other places than shown.

var C_1, C_2 : *integer*;

$C_1 := 0;\quad C_2 := 0;$

process P_1;	**process** P_2;
repeat	**repeat**
nc_1: non-critical section$_1$;	nc_2: non-critical section$_2$;
a_1: $<C_1 := C_2+1>$;	a_2: $<C_2 := C_1+1>$;
w_1: **while** $C_1 \neq 1$ **do** ;	w_2: **while** $C_2 \neq 1$ **do** ;
cs_1: critical section$_1$;	cs_2: critical section$_2$;
d_1: $C_1 := 0$;	d_2: $C_2 := 0$;
e_1: **if** $C_2 > 1$ **then** f_1: $C_2 := 1$	e_2: **if** $C_1 > 1$ **then** f_2: $C_1 := 1$
forever;	**forever**;

The following predicates are invariants of the program:

$$
\begin{array}{lclr}
G_i & \triangleq & C_i \geq 0 & i=1,2 \\
H_i & \triangleq & C_i > 0 \Leftrightarrow \mathit{in}\ w_i..d_i & i=1,2
\end{array}
$$

Question 1.1:

(a) State with a brief argument which value C_1 has when P_1 is *in* cs_1.

(b) *Omitted*

(c) Define a predicate I of the form

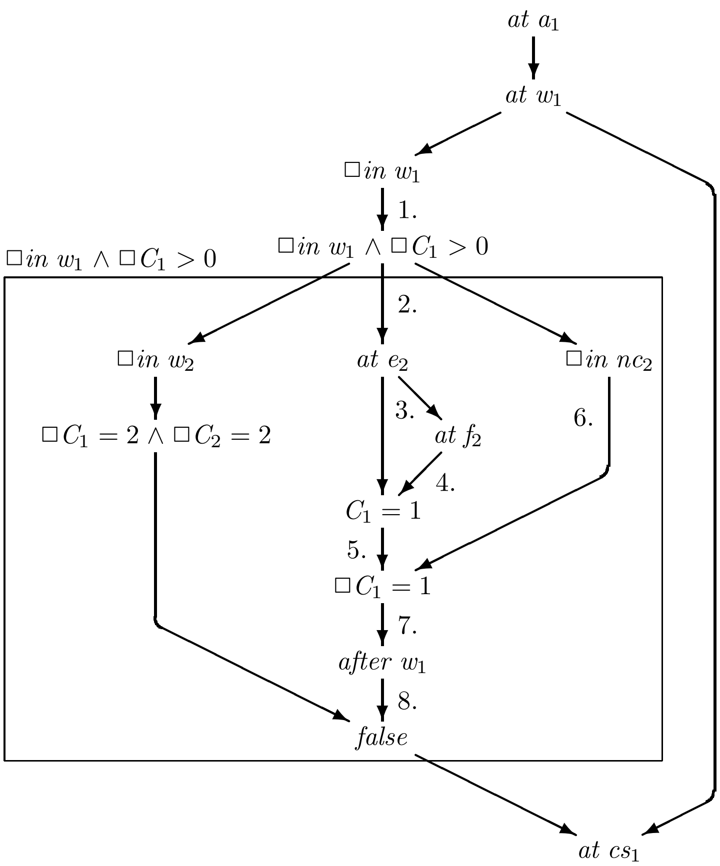
$$I \triangleq C_1 = C_2 \Rightarrow \ldots$$

expressing what holds about C_1 and C_2, when they have the same value.

Argue (informally) that I is an invariant of the program.

(d) Show, using I, that mutual exclusion is ensured in the given program.

You are now given the following proof lattice for the program:



Further, you are informed that also the following predicates are invariants of the program:

$$K_i \quad \triangleq \quad C_i > 1 \Rightarrow C_i = 2 \wedge in\ w_i \wedge in\ w_j..e_j \qquad\qquad i = 1, 2,\ j \neq i$$

Question 1.2:

Argue for the validity of the steps 1. to 8. indicated in the proof lattice.

Question 1.3:

It is now assumed that the statements e_1 and e_2 (including f_1 and f_2) are removed from the given program:

(a) Argue briefly that mutual exclusion is still ensured.

(b) Show that the implementation of the critical region is no longer resolute.

[Resolution: If both processes wants to enter the region at the same time, one of them will succeed.]

From Concurrent Systems Exam, December 2001

PROBLEM 1 (approx. 20 %)

Consider the concurrent program:

$$
\begin{array}{l}
\textbf{var } x, y : \textit{integer} := 0; \\
\\
\textbf{co} \\
\quad \textbf{repeat } a_1\colon \langle\, x < 2 \wedge y = 0 \rightarrow x := x + 1 \,\rangle \textbf{ forever} \\
\| \\
\quad \textbf{repeat } a_2\colon \langle\, y := x;\;\; x := 0 \,\rangle \textbf{ forever} \\
\| \\
\quad \textbf{repeat } a_3\colon \langle\, x = 1 \rightarrow y := 2 \,\rangle \textbf{ forever} \\
\textbf{oc}
\end{array}
$$

Question 1.1:

(a) Prove inductively that $I \triangleq (y = 0 \vee x \neq y)$ is an invariant of the program.

(b) Draw the (reachable part of) the transition graph for the program. Since control remains at the a-actions, only the (x, y) part of the state needs to be shown.

(c) Determine from the transition graph, whether $(x = 0 \vee y = 0)$ is an invariant of the program.

Question 1.2:

(a) Argue that $\Box\Diamond y = 0$ holds for the program under the assumption of weak fairness.

(b) Determine whether $\Box\Diamond y = 2$ holds for the program under the assumption of strong fairness.

Question 1.3:

Assume that the program is modified by replacing the action a_1 by the refinement:

$$b_1\colon \langle\, \textbf{await } x < 2 \,\rangle; \quad c_1\colon \langle\, \textbf{await } y = 0 \,\rangle; \quad d_1\colon \langle\, x := x + 1 \,\rangle$$

(a) Show that I is not an invariant of the modified program.

(b) We would like to prove that $x \leq 2$ is an invariant of the modified program.

State a predicate H that (i) implies $x \leq 2$, (ii) holds initially, and (iii) is inductive for the modified program.

[For H to be inductive, it must be strong enough to be preserved by all atomic actions, but you need not demonstrate this.]

From Concurrent Systems Exam, December 2002

PROBLEM 1 (approx. 20 %)

The below implementation of a critical region for two processes, P_1 and P_2, utilizes a machine instruction that indivisibly tests a boolean variable and, if false, sets an integer variable to a constant value. In the program this instruction is denoted by statements of the form $\langle$ **if** $\neg B$ **then** $X := k \rangle$. The variables C_1, C_2 and *Turn* are not changed other places than shown.

var C_1, C_2 : *boolean* := *false*;
    Turn : *integer* := 1;

process P_1;	**process** P_2;
repeat	**repeat**
nc_1: non-critical section$_1$;	nc_2: non-critical section$_2$;
a_1: $C_1 := true$;	a_2: $C_2 := true$;
b_1: $\langle$ **if** $\neg C_2$ **then** $Turn := 1 \rangle$;	b_2: $\langle$ **if** $\neg C_1$ **then** $Turn := 2 \rangle$;
w_1: **await** $Turn = 1$;	w_2: **await** $Turn = 2$;
cs_1: critical section$_1$;	cs_2: critical section$_2$;
e_1: $C_1 := false$;	e_2: $C_2 := false$;
f_1: $Turn := 2$	f_2: $Turn := 1$
forever;	**forever**;

Question 1.1:

(a) Draw the part of the transition diagram for P_1 that represents b_1.

(b) Show that mutual exclusion would not be ensured by the program if the statements b_1 and b_2 were executed in the usual, non-atomic way.

(c) Define predicates I_i of the form

$$I_i \triangleq in\ cs_i \Rightarrow \dots \qquad\qquad i = 1, 2$$

that express what can be said about *Turn* when P_i is in its critical section.

Argue that I_1 and I_2 are invariants of the program.

(d) Show, using I_1 and I_2, that mutual exclusion is ensured in the above program.

You are informed that $H_1 \triangleq at\ w_1 \wedge in\ nc_2 \Rightarrow Turn = 1$ is an invariant of the program.

Question 1.2:

(a) Express as a formula of temporal logic, the property of fairness (eventual entry) for the critical region.

(b) Explain why the algorithm is deadlock-free.

(c) Argue that the algorithm ensures fairness for the critical region.

Hint: Show that one of the processes, say P_1, cannot remain forever at w_1. Fair process execution (weak fairness) is assumed.

From Concurrent Programming Exam, June 1991

PROBLEM 2 (approx. 25 %)

In a system a number of operations $C_1, C_2, \ldots, C_n$, $(n \geq 1)$ must be executed concurrently. However, before C_i can be executed, a private initilization operation B_i as well as a common start operation A must be executed. Thus, the operations are to be executed the following way:

(∗) Initially, A as well as $B_1, B_2, \ldots, B_n$ are executed concurrently. As soon as A and B_i have finished, C_i can be executed $(i : 1..n)$. When all the operations $C_1, C_2, \ldots, C_n$ have finished, the execution starts all over again.

Question 2.1:

For a system with $n = 2$, draw a Petri-net in which the five operations A, B_1, B_2, C_1, and C_2 are synchronized as described by (∗). In the net, the operations should be represented by transitions.

The operations are to be executed by a sequential process P plus n sequential processes $Q_1, Q_2, \ldots, Q_n$. The form of these processes are:

```
process P =                 process Q[i : 1..n] =
  repeat                      repeat
    A                           B_i;
  forever;                      C_i
                              forever;
```

Question 2.2:

Show how to synchronize these processes using semaphores such that the operations A, B_i, and C_i $(i : 1..n)$ become synchronized as described by (∗).

The processes P and $Q_1, Q_2, \ldots, Q_n$ are now to be sychronized using a monitor *Synch* instead. The monitor has three parameterless operations *Done*, *Start*, *End* to be called by the processes as shown below:

```
monitor Synch;
  procedure Done();
  procedure Start();
  procedure End();
end;

process P =                 process Q[i : 1..n] =
  repeat                        repeat
    A                             B_i;
    Synch.Done()                  Synch.Start();
  forever;                        C_i;
                                  Synch.End()
                                forever;
```

Question 2.3:

Write the monitor *Synch* such that the operations A, B_i, and C_i $(i : 1..n)$ become synchronized as described by (∗).

From Concurrent Systems Exam, December 2002

PROBLEM 3 (approx. 15 %)

Below, a monitor implementation of a synchronization mechanism *Gate* is shown. The gate may be opened or closed by an operation *Set*. Processes call *Pass*() to pass the gate and have to wait if the gate is closed. A special operation $Go(k)$ lets up to k of the currently waiting processes pass through the gate.

```
monitor Gate

  var open : boolean := false;
      Queue : condition;

  procedure Pass() {
    if ¬open then wait(Queue);
  }

  procedure Set(b : boolean) {
    open := b;
    if open then signal_all(Queue);
  }

  procedure Go(k : integer) {
    for j in 1..k do signal(Queue);
  }

end
```

Question 3.1:

(a) Define a predicate I expressing that calls of *Pass*() do not wait unnecessarily.

(b) Argue that I is an invariant of the monitor.

(c) Describe the effect of $Go(k)$ if there are less than k calls of *Pass*() currently waiting.

Question 3.2:

The functioning of the given monitor *Gate* is now to be implemented by a module with the following specification:

```
module Gate
  op Pass();
  op Set(boolean);
  op Go(integer);
end
```

Write a server process for the module *Gate* that services the operations by rendezvous in such a way that it functions like the given monitor *Gate* as seen from the calling processes.

From Concurrent Systems Exam, December 2003 (2-hours)

PROBLEM 3 (approx. 20 %)

Let N be a positive integer. The server-based module *Batch* given below implements a synchronization mechanism that "collects" a batch of N items provided by calls of *put*() which may then be "removed" by a call of *unload*().

```
module Batch
  op put();
  op unload();
body

  process Control;
    var count : integer := 0;
    repeat
      while count < N do
            in put() → count := count + 1 ni;
      in unload() → count := 0 ni;
    forever;

end Batch;
```

Question 3.1:

Assume $N = 3$. Suppose that, concurrently, *unload*() is called by two processes and *put*() is called by five processes. Assuming no further calls, describe the overall effect of these seven calls.

Question 3.2:

Now, the module *Batch* is to be replaced with a monitor which provides the same operations and behaves in the same way. Write such a monitor.