Exercises on Hoare Monitors

Exercise 1: Producers - Consumers

We consider the consumer/producers problem, but now we want to organise the synchronisation with Hoare monitors.

The specification of the monitor is:
Monitor Prod_Conso {
void Insert(. . .);
void Extract(. . .);
end Prod_Conso ;

For each of the three versions (same as the semaphores one):
- Give the specification of the monitor
- Precise the conditions of blocking and unblocking of one producer and one consumer.
- Deduce the state variables and conditions variables handled in the monitor.
- Give the code of the monitor

Exercise 2: Readers - Writers

The model of Readers-Writers is representing a situation in the management of shared files or accesses to databases.
Two families of processes access the information.
- Readers only want to read the data. They can access in parallel the data.
- Writers want to modify the data. They must access the data in mutual exclusion.

The behaviour of the Reader is therefore:
- Begin
- …
- Ask for reading
- Read
- Signal the end of reading
- …
- End

The behaviour of the writer is:
- Begin
- …
- Ask to write
- Write the modification
- Signal the end of writing
- …
- End

We propose to use a Hoare monitor to handle the access to the shared resource.
The specification of the monitor is:
Monitor Reader_Writer {
void Begin_Read();
void End_Read();
void Begin_Write();

```
void End_Write();
}
```

We want to implement several versions:
Version 1: Priority of the readers over the writers
When no reader reads, readers and writers have the same priority. When a reader is reading, all other readers asking to read can read, whatever the number of waiting writers. When a writer writes, no-one else can access the resource (no reader, no other writer). When the writer have finished to write, it tries to activate a reader in priority, and writers afterwards.

Version 2: Priority of the writers over the readers.
We want to insure priority to writers against readers so that the data is always up to date for a reader. When a writer wants to access the resource, it can't interrupt a currently working reader or writer. A new writer has no priority against other already waiting writers. But it has priority against all waiting readers.

Version 3: Fairer access to resources.
To which erroneous situations can lead the two versions 1 and 2?
In this version, a finishing writer must let all waiting readers the access in priority, and not to the writers (just as in first version). and when these readers are finishing, potential waiting readers must respect the priority rule of the writers over the readers (just as in version 2).

Version 4: FIFO Accesses
We suppose now that the requests are processed in their incoming order. To order globally the readers and the writers, all processes will be blocked on the same condition. Indeed, using two queues would not allow to know if the reader 4 arrived before the writer 3. We can note that when awaking it is not possible to distinguish between a reader and a writer. Hence it might be possible to awake then to put to sleep again a process (reader or writer) if if was not possible to unblock it at this moment.

Questions:
For each of these versions:
- give the specification of the monitor
- precise the conditions of blocking and unblocking of readers and writers
- deduce the state and condition variables of the monitor
- give the code of the monitor

Exercise 3: Orders Processing

Your preferred shopping mall decided for Christmas time to give his main clients a given number of special counters (NB_COUNTERS) to handle their orders.
The principles of the functioning of these counters are the following:
- one client awaits a free counter, let his order, then waits for his order to be processed.
- the processing of one order needs several successive steps (NB_STEPS) and dedicated employees are in charge of each of these steps. The step "i" of an order can only starts when the employee in charge of step "i-1" has finished his work.
- when the order has been processed, the client quits the counter, fully satisfied.

For instance, in practice, the mall is setting up 3 counters and the processing of an order needs 4 steps : (1) list the ordered items (2) fetch the ordered items (3) wrap them as Christmas presents (4) make the client pay

We consider two kind of processes: Client and Employee
- A process Client let an order to a free counter and waits his order to be processed. At most NB_COUNTERS orders can be processed in parallel.
- A process Employee, dedicated to step "i" takes in charge the orders in the order of their arrival, process his work on the order and bring the order to the original counter. Then he can take in charge the next order. When the last step has been processed, then the Client can restart his execution.

We suppose defined the following types:
- StepNumber: an integer between 0 and NB_STEPS
- CounterNumber: an integer between 0 and NB_COUNTERS
- Order: the order established by the client

And we suppose to have the following subprogram:
• void applyStep (StepNumber aStep, Order *anOrder) ;
that can be used by an employée to process his work (i.e. the step aStep) and modify therefore the order anOrder.

We want to use a Hoare Monitor to synchronise the processes.

```
Monitor HandleRequests {
void order(void);      // used by the client

// used by an employee
void startStep(StepNumber aStep, Order *anOrder, CounterNumber *aCounter);
void finishStep(Order anOrder, CounterNumber aCounter);
}
```

The behaviour of the Client process is:

```
Process Client {
        // go to the shopping mall
        HandleRequests.order();
        // go home, satisfied
}
```

```
Process Employee (StepNumber appliedStep) {
while (1) {
HandleRequests.startStep(appliedStep, &anOrderToProcess, &originalCounter) ;
applyStep(appliedStep, &anOrderToProcess) ;
HandleRequests.finishStep(anOrderToProcess, originalCounter) ;
}
}
```

Questions:
- Precise the conditions of blocking and unblocking of a Client and Employee processes.
- Deduce the state and conditions variables handled by the monitor.
- Write the code of the monitor.