6SENG002W Concurrent Programming
Printing System Coursework 2020/21

1. Hints for the FSP Printing System

Basically, you need to allow all the various users to share the printer, i.e. either students printing a document or a technician refilling the paper. This must be done securely by ensuring that each user accesses the printer **mutually exclusively (ME)**.

So to develop the system you need to use all of the following elements.

1. The 1 Printer that prints the document & keeps track of how much paper it has, so that the students can get it & use it to print a document.

SEE:

(a) the indexed actions & processes examples in the notes. Lecture 3, PART I, slides: 3 - 15.

(b) tutorial Exercise 3.3 [Answer: its modelling a variable that can hold either a 0 or 1] & Exercise 3.4.

2. ME Sharing of a resource by the 2 students & 1 technician, using "acquire" & "release" actions.

SEE:

(a) Lecture 4, PART II: How to Achieve Mutually Exclusive Access [slides: 20 – 45] using "acquire" & "release" actions - ME sharing of a resource, Process Labelling, Process Prefix Label Sets & the Examples

3. Allow the 2 students & 1 technician to **communicate** with the Printer, i.e. the documents id, amount of paper to refill.

SEE:

(a) Lecture 4: PART III - Inter-Process Communication using "Synchronised Message Passing", [slides 46 - 49]

4. Ensuring that the Printer does not break ME by performing **independent asynchronous actions** when interacting with a user (student/technician) after that user has "locked" it.

SEE:

(a) Lecture 4: Process Alphabet Extensions [slides 41 - 45]

5. For examples of FSP Structure Diagrams see Lecture 4: PART IV, in particular Structure Diagram Example 2, [slide 56].

6. For example of Process's Trace Trees, see Lecture 2, PART IV, [slides 41 – 46].

2. Java Part CW HINTS

2.1 Implementing the Printer "Monitor" Class

1. The **Printer** IS NOT A Java THREAD,

2. The **Printer** must be implemented as a **Java monitor,**

   SEE: Lectures 8 & 9 - these cover Java monitors.

3. ****NOT BY ANY OTHER MEANS****, I.E. **certainly NOT**:

   (a) by using the synchronized statement:

```
synchronized ( obj ){
                        \\ critical section
                    }
```

   (b) Java's built in Semaphore class:
   `semaphore.acquire()` & `semaphore.release()`, etc.

   (c) Some other "thing" you found on the web.

4. The check list for a **creating "safe & secure" Java monitor class** is given in lectures 8 & 9, but to summarise:

   - ALL data members (ie. variables declared in the class) MUST be encapsulated, i.e. only accessible via the **`public synchronized`** methods. So add the appropriate access modifiers when declaring them.

   - The class' constructor should have the necessary parameters to be able to completely initialise the state on the monitor.

   - ALL monitor class methods in its interface (i.e. its "`public`" ones) must be "`synchronized`".

   - In other words, ALL public methods MUST be declared as "`synchronized`", ie. "`synchronized`" must be added to their method signatures.

   - You need to check how each monitor operation should be performed, so *basically:*
     Does it have any preconditions that must be true before the operation can be performed?

     - if **yes** then it needs a "`while`-loop guard" (See Lecture 9)
     - if **no** then it does not.

   - If a monitor method **"changes the state of the monitor"**, i.e. modifies one of the monitor's variables, then it SHOULD call "`notifyAll()`" before it exits the method.

This is because some other thread may be "waiting" for the monitor's state to change to be able to complete its monitor operation.

- If a monitor method DOES NOT "change the state of the monitor" then it does NOT need to call "`notifyAll()`".

2.2 Implementing the Printer's Users: Students & Technicians.

1. These should ALL be implemented as a Java THREAD.

2. By SUBCLASSING the `Thread` class.

3. Their constructors should include the necessary parameters to be able to completely initialise the state on the thread.

    - references to any objects it needs to access.
    - its id, thread group, etc.

4. The constructor should call an appropriate Thread class constructor.

5. It SHOULD NOT CREATE its own instances of any shared objects.
   Otherwise, they will not be visible outside it & hence not shared. (See 3.)

2.3 Implementing the Printing System

This is the main program that **creates all of the components of the system & links everything together.** Basically, it is the “code version” of the diagram of the system given in the CW Specification.

(See CW Specification & the Producer/Consumer example in Lecture 8.)