NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Solution to Midterm EXAMINATION FOR
Semester 2, 2009/2010
CS 3211 - PARALLEL and CONCURRENT PROGRAMMING

March 2010 Time Allowed: 1 Hour 15 minutes

A. Define a process *Duplicate* which takes a value 0..9 as input, and outputs two copies of the value. Use process equations to define the process.
[**2 marks**]

Answer:

```
Duplicate =  in(x:0..9) -> ( out1(x) || out2(x) ) -> Duplicate.
```

or

```
Duplicate =  in(x:0..9) -> out1(x) -> out2(x) -> Duplicate.
```

B. Consider a lock process defined via a process equation

```
Lock = acquire -> release -> Lock.
```

Processes using the lock will synchronize with it using the *acquire* and *release* actions. Extend the above definition to define a lock process which can be acquired by the same process more than once, just like the recursive locking mechanism in Java which was taught in class.
[**2 marks**]

Answer:

```
Jlock(0) = acquire -> Jlock(1).

Jlock(x) = (acquire -> Jlock(x+1)) | (release -> Jlock(x-1)).    where x > 0
```

There is no bound on the number of times the lock may be acquired but the number of non-released acquisitions are kept track of.

C. Consider an asynchronous composition of two processes described in the Promela modeling language.

```
byte x;

proctype myproc(bool i) {
   do
   :: x = x + x;
   od
}

init{  atomic{ run myproc(0); run myproc(1)  } }
```

Use the semantics of Promela to find out the values taken by `x` during the program execution. Initially x is equal to 1.
[**2 marks**]

Answer: Every execution of the loop duplicates the value of `x`. So, `x` will assume the values of the form 2^n for $n \geq 0$.

D. Suppose the infinite loop in Question C is compiled into the following assembly code. The processes are running asynchronously, and each time a process is scheduled, only its next *instruction* is executed atomically. Initially x = 1, and we use a hypothetical machine with infinite memory. Note that x is a shared global variable and reg^i_A, reg^i_B are local registers in processes A and B respectively.

Process A		Process B	
$Loop_A$:	$reg^1_A = x$	$Loop_B$:	$reg^1_B = x$
	$reg^2_A = x$		$reg^2_B = x$
	$reg^3_A = reg^1_A + reg^2_A$		$reg^3_B = reg^1_B + reg^2_B$
	$x = reg^3_A$		$x = reg^3_B$
	go to $Loop_A$		go to $Loop_B$

Which of the following values can x reach in the execution of the above program — 2,3,5? Explain your answer in each case. If the value is possible, show a detailed interleaving producing the value. If the value is not possible, argue why it is not possible.
[3 marks]

Getting the value 2

```
regA1 = x (gets 1)
regA2 = x (gets 1)
regA3 = regA1 + regA2 (gets 2)
x = regA3  (gets 2)
                // the other thread has not even been scheduled at this point
```

Getting the value 3

```
                                     regB1 = x   (gets 1)
regA1 = x (gets 1)
regA2 = x (gets 1)
regA3 = regA1 + regA2 (gets 2)
x = regA3  (gets 2)
                                     regB2 = x   (gets 2)
                                     regB3 = regB1 + regB2   (gets 3)
                                     x = regB3   (gets 3)
```

Getting the value 5

```
                                     regB1 = x   (gets 1)
regA1 = x (gets 1)
regA2 = x (gets 1)
regA3 = regA1 + regA2 (gets 2)
x = regA3  (gets 2)

regA1 = x (gets 2)
                                     regB2 = x   (gets 2)
                                     regB3 = regB1 + regB2   (gets 3)
                                     x = regB3   (gets 3)
regA2 = x (gets 3)
regA3 = regA1 + regA2 (gets 5)
x = regA3  (gets 5)
```

E. Consider the following program. `flag1` and `flag2` are boolean variables which are initialized to false. Whenever a thread is scheduled, one line of code in the thread can be assumed to be executed atomically.

```
Thread 1                        Thread 2
-----------------------------------------------
while (!flag1)                  while(!flag1){
  flag2 = toggle(flag2);            if (!flag2)
                                        flag1 = true;}
```

`toggle` returns true if the input argument is false, and returns false if the input argument is true.

- Give an example interleaving where the program terminates.
- Give an example interleaving where the program does not terminate.
- What are the possible values of `flag1` and `flag2` when the program terminates. **3 marks**

Answer:

-
```
                                   while(!flag1)  // satisfied
                                       if (!flag2) // satisfied
                                           flag1 = true;
                                   while(!flag1)     // exit loop
                                   [CONTEXT SWITCH]
 while (!flag1) // exit loop
```

- An execution where the second thread is not scheduled at all.
- `flag1` must be true for the loops to terminate. We already showed an execution where `flag2 == false`. We can also have `flag2 == true` as follows.

```
                               while(!flag1)  // satisfied
                                   if (!flag2) // satisfied
                                   [CONTEXT SWITCH]
 while (!flag1) //satisfied
    flag2 = toggle(flag2)
[CONTEXT SWITCH]
                                       flag1 = true;
                               while(!flag1) // exit loop
                                       [CONTEXT SWITCH]
while (!flag1) // exit loop
```

F. Consider the following schematic code.

```
synchronized method_in_which_P1_waits(){
     while (!flag1)  wait();
     ...
}

synchronized method_in_which_P2_waits(){
     while (!flag2)  wait();
     ...
}

synchronized method_which_tries_to_move_ahead(){
int oracle_says_yes;

     oracle_says_yes = .... // read in integer value
     if (oracle_says_yes > 0) flag1 = true;
     else   flag2 = true;
     notifyAll();
}
```

Suppose process P1 is waiting on flag1, and process P2 is waiting on flag2. What will happen if we replace notifyAll with notify? **[2 marks]**

Answer: The wrong process may be awakened by the scheduler, and it will go back to wait again. The process which can get past the wait statement will however remain non schedulable and suffer from starvation (in terms of being eligible for execution).

G. Consider a concurrent system with a producer thread and a consumer thread manipulating a bounded buffer with at most five items. Consider the following schematic code for producer and consumer, using the `wait` and `notify` primitives.

```
int count = 0;

Producer thread executes:                         Consumer thread executes:
   while(true){                                           while(true){
       // produce the item here                             if (count == 0) wait();
       if (count == 5) wait();                              // retrieve the item from buffer
       // insert the item into buffer                       count = count - 1;
       count = count + 1;                                   if (count == 4) notify();
       if (count == 1) notify();                            // consume the item here
   }                                                     }
```

If we implemented this schematic code in Java, what could you expect about the correctness and efficiency of such a program? Give detailed comments. **3 marks**

Answer: Efficiency is not an issue since busy waiting is avoided using wait and notify.

The unprotected update of the count variable can lead to lost notifications

```
Producer                                 Consumer
-------------------------------------------------
                                         if (count ==0)
                                         [CONTEXT SWITCH]
if (count ==5) // false
// insert item
count++ // count is now 1
if (count == 1) notify();
[CONTEXT SWITCH]
                                              ... wait()  // notification was already sent !!
```

H. In class, we studied how processes can be connected together by relabeling of action names. Consider a process P which has two input actions which input data and acknowledgment, and two output actions which output data and acknowledgment.

```
P = din(x) -> dout(x) -> ain -> aout -> P
```

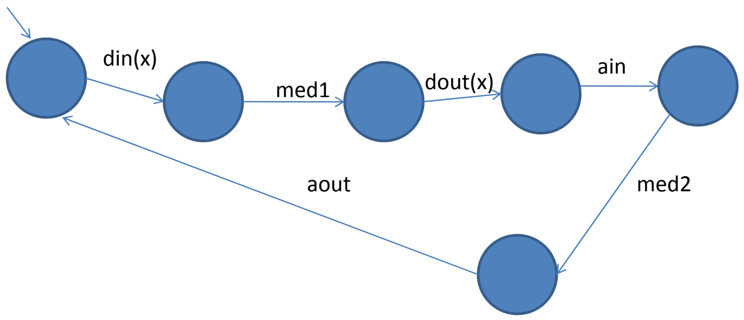
How can you connect two such processes, such that (i) the dataout of the first process is fed to the datain of the second process, and (ii) the ackout of the second process is fed to the ackin of the first process? Note that the datain, ackout of the first process as well as the dataout,ackin of the second process should be externally visible. No other actions should be externally visible. Write the process equation for the resultant process.

Compare the resultant process with P in terms of allowed execution traces. Explain your answer **3 marks**

Answer:

```
(P/{med1/dataout, med2/ackin} ||  P/{med1/datain, med2/ackout})\{med1,med2}
```

The resultant process is equivalent to P, as can be confirmed by drawing the state models of P and the resultant process. Following is the state model without the action hiding. The action hiding will make it equivalent to P.


din(x)
med1
dout(x)
ain
med2
aout