# CSE224 HW2

**Yawen Zhao A53280596**

## Part 1

The result of this problem I get is shown below:

```
{'keyword': 231, 'literature': 'The wren\nEarns his living\nNoiselessly.\n-
Kobayahsi Issa'}
```

The code of this problem is shown below. (Here I use Python)

```
import xmlrpc.client

s = xmlrpc.client.ServerProxy("http://cse224.sysnet.ucsd.edu:7777/RPC2")
print(s.litserver.getLiterature("A53280596"))
```

## Part 2

4.2 For the current process one message copy has arrived at, do following guarded actions:

```
program judge_messgae
type message = record
      seqNum : int
      sourceId : string
      destId : string
      var : the type of variable of process
define p_cur : process, m : message, p_que : queue
initially p_que.offer(p_cur), m.seqNum = random num,
[]        m.souceId = i, m.destId = j, m.var = v_i
do p_que.poll().Id == m.destId -> p_cur.var = m.var; break;
[] p_que.poll().Id != m.destId -> for p_next in p_cur.outgoing_edges do
                                      p_que.offer(p_next)
                                  end for
```

```
od
```

4.3 Do the following guarded actions:

```
program buy_soda
define paid : int, numOfSoda : int
initially paid = 0
do paid < 50 * num_of_soda -> coin = getMoney();
                              do coin != quarter && coin != dime -> continue;
                              do coin == quarter || coin == dime -> paid += coin
[] paid >= 50 give the customer num_of_soda cans soda; paid = 0;
od
```

4.8

```
program book_admin
define bookA : boolean,
initially bookA, bookB = true
do bookA -> Alice get bookA;  bookA = false;
[] bookB -> Bob get bookB;  bookB = false;
[] bookA & bookB -> Carol get book A and B;  bookA = false; bookB = false;
[] !bookA -> skip
[] !bookB -> skip
od

program renter
define Alice, Bob, Carol: string
do Alice rents bookA -> do A -> A = false
                        od
[] Bob rents bookB -> do B -> B = false
                      od
[] Carol rents book -> do A & B -> A = false; B = false;
                        od
[] Alice return bookA -> A = true
[] Bob return bookB -> B = true
[] Carol return bookA and bookB -> A = true; B = true
od
```

This will need a strongly fairness.

If we need to transfer it to a weak fairness schedule, we can set a switch botton variable.

```
program book_admin
define bookA : boolean, switchA, switchB : boolean
initially bookA, bookB = true
do bookA & !switchA -> Alice get bookA;  bookA = false;
[] bookB & !switchB -> Bob get bookB;  bookB = false;
[] bookA & bookB -> Carol get book A and B;  bookA = false; bookB = false;
[] !bookA -> skip
[] !bookB -> skip
od

program renter
define Alice, Bob, Carol: string, switchA, switchB : boolean
do Alice rents bookA -> do bookA -> bookA = false
                        od
[] Bob rents bookB -> do bookB -> bookB = false
                      od
[] Carol rents book -> do bookA & bookB -> A = false; B = false;
                        od
[] Alice return bookA -> A = true; switchA = true;
[] Bob return bookB -> B = true; switchB = true;
[] Carol return bookA and bookB -> A = true; B = true; switchA = false; switch =
false;
od
```

5.3

Assume the global state $s = i$, and the well founded $WF$ set is $[n-1, n-2, \ldots, 0]$. The meauring function $f$ transfer s to WF is $w = n - 1 - s$ if $X[s] \neq t$ and $w = 0$ if $X[s] = t$. As $i$ keep increases as the program described, the value $w$ will be deduced to 0. which shows the program will be terminated in a bounded number of steps.

5.5

Draw an arrow ($\rightarrow$) from clock i to clock j if $c[i] > c[j]$, else draw an arrow ($\leftarrow$) from clock j to clock i if $c[i] < c[j]$. If $c[i] = c[j]$, the arrow will dissapear. Then we can get observe that every round of tick, we will have at least one clock deduced to have only $\rightarrow$. As $\rightarrow$ and $\leftarrow$ number are the same, the number of $\rightarrow$ and $\leftarrow$ will keep deducing as some of the arrows are disappear.

We can define a function D that maps the set of global states of a system to a set of nonnegative integers:

$D = d[0] + d[1] + \ldots + d[n-1]$

Let d[i] represent the number of $\rightarrow$ of the clock i has. Then $d[i] \geqslant 0$ and thus $D \geqslant 0$. As the arrows are keeping dissapear, $D$ will keep decrease to 0.

5.8

If at start there is no number in channel, we will send the max integer of B to channel 1and decide if we need to switch it with pop the min integer of A to B by compare it with A. If it's larger than the min integer of A, we will send the min value of A to channel 2 back to B and compute the current maximum value in B and send it to channel 1 to A again. Keep this process until the maximum value of B is smaller than the min value of A.

```
program sort_pq
define A : int[], B : int[], c1, c2 : channel; change : boolean;
initially c1 = null, c2 = null, change = false;
{program p}
define a : integer
initially a = min(A)
do !empty(c1) -> get an integer i from channel c1
                 do i > a -> send a to c2; A.add(i); t = min(A)
                 [] i <= a -> send i to c2
                 od
od

program q
define b : integer
initially b = max(B)
do !change -> send b to c1
[] !empty(c2) -> get an integer j from channel c2
                 do j != b -> B.add(j); b = max(B); send b to c1
                 od
od
```

5.14

In a completely connected network, every two node is connected to each other. If we assume that if there exist two node i and j that $|x[i] - x[j]| = 2$. As we know that for every two neighbor node, they will change only 1 for one time. If we need to reach $|x[i] - x[j]| = 2$, we first need to reach $|x[i] - x[j]| = 1$. However if, for example, $x[i] - x[j] = 1$ (we assume $x[i] < x[j]$), then for $x[i]$ it will stay the same as there exist j in its neighbor that $x[i] < x[j]$ and it will never reach $x[i] - x[j] = 2$