IN.5022: Concurrent and Distributed Computing - Serie 2

Antoine Tissot

October 4, 2023

1 Exercise 1

```
program coffee machine

define totalAmountDeposited : double;

initially totalAmountDeposited = 0;

do
    coin = 10 cents → totalAmountDeposited = totalAmountDeposited + 0.10;
    coin = 1 franc → totalAmountDeposited = totalAmountDeposited + 1.00;
    coin != 10 cents && coin != 1 franc → Reject and eject coin;

    totalAmountDeposited >= 2.50 → Dispense coffee and exit loop;
od
```

2 Exercise 2

```
programm library

define A; define B;

% Alice
do true →
    Request A;
    do
        ¬A → skip;
    od
    use A;
    release A;
od

% Bob
do true →
    Request B;
    do
        ¬B → skip;
    od
    use B;
    release B;
```

```
od

% Carol
do true →
    Request A && B;
    do
        ¬A || ¬B → skip;
    od
    use A && B;
    release A && B;
od

% Librarian
do true →
    carolRequest A && B →
        do
           ¬A && ¬B → wait until A and B are released
           ¬A && B → wait until A is released
            A && ¬B → wait until B is released
            A && B → grant A and B
        od
    aliceRequest A →
        do
           ¬A → wait until A is released
            A → grant A
        od
    bobRequest b →
        do
           ¬B → wait until B is released
            B → grant B
        od
od
```

Fairness : if Carol keeps requesting books A and B while Alice and Bob are also attempting to borrow the books. Without some fairness mechanism, Carol might always get the books, or Alice might always get her book, thereby indefinitely delaying Bob. This program can therefore be considered as *unfair*.

3 Exercise 3

```
programm ring

define N, M;
define token;

% Process Function
process p(id, N, M)

    define counter = 0;    // Initialize local counter for each process
    define nextProcessID = (id + 1) % N;  // Calculates the successor process id
         in the ring

    do true →
        recieve token from p((id - 1 + N) % N);  // Receives a token from
            predecessor
        counter = counter + 1;
```

```
        print counter;
        if counter < M →
            send token to p(nextProcessID, N, M);  // Send token to successor
        else
            write counter to a file;
            print "process end";  // Terminates the process after receiving M
                tokens
        end if
    od
end process

% Starter Process
process s(N, M)
    // Initialize the ring
    for i = 1 to N
    do
        create p(i, N, M);  // Launch each process in the ring only once
    od

    // Start the token circulation
    send token to p(1, N, M);  // Start the circulation at p1
end process
```