<u>Week 4 Classwork due on Friday Oct 5, 22:00 Hour</u>

<u>Group 5</u>

Wong Ann Yi (1004000)
Liu Bowen (1004028)
Tan Chin Leong Leonard (1004041)

<u>Exercise 1</u>

Identify and describe the possible time-of-check time-of-use (TOCTOU) errors in the following code snippet, and provide correction of the errors:

```
int addMoneyOnDate(String sourceAccount, String destinationAccount,

Money amount, Date date)

{Date today = getCurrentDate();

if (date.equals(today))

        if (currentUser.owns(sourceAccount))

                if (sourceAccount != destinationAccount) {

                        File *sourceFile = new File(sourceAccount);

                        File *destinationFile = new File(destinationAccount);

                        if (balance(sourceFile) >= amount) {

                                debitByAmount(sourceFile, amount);

                                creditByAmount(destinationFile, amount);

                                }

        }

}
```

Answer:

We will use the semaphore to lock the Add Money function of debit and credit so that this will prevent other processes to debit amount from the source account simultaneously. This will also preserve the atomicity of the source account and Add Money function. Assuming we initialize the semaphore Z as 1. When the source file is used by the function, the semaphore will be zero and prevent other processes from using the source file. Upon completion of the Add Money function, the semaphore Z will be increment by 1 allowing another process to access the source account.

```
int addMoneyOnDate(String sourceAccount, String destinationAccount,
Money amount, Date date)
Semaphore Z = 1;
{Date today = getCurrentDate();
if (date.equals(today))
    if (currentUser.owns(sourceAccount))
        if (sourceAccount != destinationAccount && Z != 0){
            wait (Z);
            File *sourceFile = new File(sourceAccount);
            File *destinationFile = new File(destinationAccount);
            if (balance(sourceFile) >= amount) {
                debitByAmount(sourceFile, amount);
                creditByAmount(destinationFile, amount);
                signal (Z)
            }
    }
}
```

Exercise 2

A computer has three commonly used resources designated A, B and C. Up to three processes designated X, Y and Z run on the computer and each makes periodic use of two of the three resources.

• Process X acquires A, then B, uses both and then releases both.

• Process Y acquires B, then C, uses both and then releases both.

• Process Z acquires C, then A, uses both and then releases both.

a) If two of these processes are running simultaneously on the machine, can a deadlock occur? If so, describe the deadlock scenario.
   Answer:
   The deadlock scenario would not happen. For example, if process X acquires A and B, resource C can only be acquired by Z first. Therefore, process X can continue to run with no deadlock. Upon completion and the release of A and B, Z (which has C) can acquire A to run. This left the process Y with resource B waiting for C to be released. When C and A are released, process Y will run with B and C while process X will acquire A and wait for the release of B.
   As a result, there is no deadlock. See below diagram for illustration.

Time	Process X	Process Y	Process Z
1	A	0	C
2	A+B	0	0
3	0	B	C+A

b) Describe a scenario in which deadlock occurs if all three processes are running simultaneously on the machine.
   Answer:

There will be deadlock if three processes are running concurrently. See below table for illustration. The three processes are holding on to the three resources and causing a deadlock.

Time	Process X	Process Y	Process Z
1	A	B	C
2	0	0	0
3	0	0	0

c) Modify the algorithm for acquiring resources so that deadlock cannot occur with three processes running.

Answer:

We can modify algorithm as follows:

• Process X acquires A, then B, uses both and then releases both.

• Process Y acquires B, then C, uses both and then releases both.

• Process Z acquires A, then C, uses both and then releases both.

See below table for illustration. Assuming each process only needs to process the resources once. Then there will not be any deadlock.

Time	Process X	Process Y	Process Z
1	A	B	0
2	A	B+C	0
3	A+B	0	0
4	0	B	A
5	0	0	A+C

Exercise 3

The following is a set of three interacting processes that can access two shared semaphores: U = 2; V = 1;

```
[Process 1]         [Process 2]         [Process 3]

L1: wait(U)         L2: wait(V)         L3: wait(V)

type("C")           type("A")           type("D")

signal(V)           type("B")           goto L3

goto L1             signal(V)

                    goto L2
```

Within each process the statements are executed sequentially, but statements from different processes can be interleaved in any order that's consistent with the constraints imposed by the semaphores. Assume that once execution begins, the processes will be allowed to run until all 3 processes are stuck in a wait() statement, at which point execution is halted.

a) Assuming execution is eventually halted, how many C's are printed when the set of processes runs? Why?

	U	V	Output
L1	1	2	"C"
L1	0	3	"C"
L1	-1		stuck

We will only consider process 1 since the question is on the number of "C". Therefore, there are 2 times "C" output. There are value 2 in semaphore U. Therefore, the process 1 utilizes the 2 values to print 2 "C" character.

b) Assuming execution is eventually halted, how many D's are printed when this set of processes runs? Why?

Answer:

Process 1:

	U	V	Output
L1	1	2	“C”
L1	0	3	“C”
L1	-1		stuck

For process 2:

	V (beginning)	V(after)	Output
L2	2	3	“AB”
L2	2	3	“AB”
many times	2	3	No network changed

Therefore, for process 3:

	V	Output
L3	2	“D”
L3	1	“D”
L3	0	“D”
L3	-1	stuck

Therefore, there are 3 times “D” output. This is because semaphore V can be increased to 3 due to two signal V by Process 1.

c) Is CCDDABD a possible output sequence when this set of processes runs? Why?

	U	V	Output
L1	1	2	“C”
L1	0	3	“C”
L3	0	2	“D”
L3	0	1	“D”
L2	0	1	“AB”
L3	0	1	“D”

Therefore, the CCDDABD is a possible output sequence and the sequence is: L1->L1->L3->L3->L2->L3