Sheet6

CONCURRENCY: DEADLOCK AND STARVATION

1) Give examples of reusable and consumable resources.
2) What are the three conditions that must be present for deadlock to be possible?
3) What are the four conditions that create deadlock?
4) How can the hold-and-wait condition be prevented?
5) List two ways in which the no-preemption condition can be prevented.
6) How can the circular wait condition be prevented?
7) What is the difference among deadlock avoidance, detection, and prevention?
8) Given the following state for the Banker's Algorithm.
   6 processes P0 through P5
   4 resource types: A (15 instances); B (6 instances); C (9 instances); D (10 instances)
   Snapshot at time T0:

Available

A	B	C	D
6	3	5	4

	Current Allocation				Maximum Demand			
Process	A	B	C	D	A	B	C	D
P0	2	0	2	1	9	5	5	5
P1	0	1	1	1	2	2	3	3
P2	4	1	0	2	7	5	4	4
P3	1	0	0	1	3	3	3	2
P4	1	1	0	0	5	2	2	1
P5	1	0	1	1	4	4	4	4

   a) Verify that the Available array has been calculated correctly.
   b) Calculate the Need matrix.
   c) Show that the current state is safe, that is, show a safe sequence of processes. In addition, to the sequence show how the Available (working array) changes as each process terminates.
   d) Given the request (3, 2, 3, 3) from Process 5. Should this request be granted? Why or why not?

9) In the code below, three processes are competing for six resources labeled A to F.

```
void P0()
{
   while(true) {
      get(A);
      get(B);
      get(C);
      // critical section
      // use A, B, C
      release(A);
      release (B);
      release (C);
   }
}
```

```
void P1()
{
   while(true) {
      get(D);
      get(E);
      get(B);
      // critical section
      // use D, E, B
      release(D);
      release (E);
      release (B);
   }
}
```

```
void P2()
{
   while(true) {
      get(C);
      get(F);
      get(D);
      // critical section
      // use C, F, D
      release(C);
      release (F);
      release (D);
   }
}
```

   a) Using a resource allocation graph, show the possibility of a deadlock in this implementation.
   b) Modify the order of some of the get requests to prevent the possibility of any deadlock. You cannot move requests across procedures, only change the order inside each procedure. Use a resource allocation graph to justify your answer.

10) Suppose the following two processes, foo and bar are executed concurrently and share the semaphore variables S and R (each initialized to 1) and the integer variable x (initialized to 0).

```
void foo( ) {
   do {
      semWait(S);
      semWait(R);
      x++;
      semSignal(S);
      semSignal(R);
   } while (1);
}
```

```
void bar( ) {
   do {
      semWait(R);
      semWait(S);
      x--;
      semSignal(S;
      semSignal(R);
   } while (1);
}
```

   a) Can the concurrent execution of these two processes result in one or both being blocked forever? If yes, give an execution sequence in which one or both are blocked forever.
   b) Can the concurrent execution of these two processes result in the indefinite postponement of one of them? If yes, give an execution sequence in which one is indefinitely postponed. Note: Indefinite postponement is equivalent to starvation.

11) Consider the following ways of handling deadlock: (1) banker's algorithm, (2) detect deadlock and kill thread, releasing all resources, (3) reserve all resources in advance, (4) restart thread and release all resources if thread needs to wait, (5) resource ordering, and (6) detect deadlock and roll back thread's actions.
   a) One criterion to use in evaluating different approaches to deadlock is which approach permits the greatest concurrency. In other words, which approach allows the most threads to make progress without waiting when there is no deadlock? Give a rank order from 1 to 6 for each of the ways of handling deadlock just listed, where 1 allows the greatest degree of concurrency. Comment on your ordering.
   b) Another criterion is efficiency; in other words, which requires the least processor overhead. Rank order the approaches from 1 to 6, with 1 being the most efficient, assuming that deadlock is a very rare event. Comment on your ordering. Does your ordering change if deadlocks occur frequently?

12) Comment on the following solution to the dining philosophers problem.
   a) A hungry philosopher first picks up his left fork; if his right fork is also available, he picks up his right fork and starts eating; otherwise he puts down his left fork again and repeats the cycle.
   b) A hungry philosopher picks up both forks at the same time (in a critical section), only if both forks are available.

13) Suppose that there are two types of philosophers. One type always picks up his left fork first (a "lefty"), and the other type always picks up his right fork first (a "righty"). The behavior of a lefty and a righty is defined as follows.

```
void lefty(int i)
{
   while(true) {
      think();
      wait( fork[i] );
      wait( fork[(i+1) % 5] );
      eat();
      signal( fork[(i+1) % 5] );
      signal( fork[i] );
   }
}
```

```
void righty(int i)
{
   while(true) {
      think();
      wait( fork[(i+1) % 5] );
      wait( fork[i] );
      eat();
      signal( fork[i] );
      signal( fork[(i+1) % 5] );
   }
}
```

Prove the following:
   a) Any seating arrangement of lefties and righties with at least one of each avoids deadlock.
   b) Any seating arrangement of lefties and righties with at least one of each prevents starvation.

How to submit the homework assignments?

- Solve the sheet individually without looking up the solution on the Internet. The sheet is to practice; it is a learning tool not an exam.
- Assignments are to be **handwritten**.
- Papers are to be scanned (I like camscanner app). Put all images in a pdf file (camscanner does that for you)
- Use MS Teams to submit
    - Your filename should be your user id
