COMPSCI 3SH3 Winter, 2021
Dr. Bojan Nokovic

Assignment/Homework 2, Mar. 6th 2021
Assignment due date: Mar. 21st 23:59:59.
Note: Please work on this assignment individually. Students copying each other's answer will get a zero and will perform poor on midterm and final.

1. **Threads** (10 Marks)

   a) **Question: 4.10** (2 marks): Which of the following components of program state are shared across threads in a multithreaded process?
      i) Register values
      ii) Heap memory
      iii) Global variables
      iv) Stack memory

   b) **Question: 4.13** (1 mark):
   Is it possible to have concurrency but not parallelism? Explain.

   c) **Question: 4.14** (3 marks): Using Amdahl's Law, calculate the speedup gain for the following applications:
      i) 40 percent parallel with (a) eight processing cores and (b) sixteen processing cores
      ii) 67 percent parallel with (a) two processing cores and (b) four processing cores
      iii) 90 percent parallel with (a) four processing cores and (b) eight processing cores

   d) **Question: 4.16** (4 marks):
   A system with two dual-core processors has four processors available

for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between start-up and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).

i) How many threads will you create to perform the input and output? Explain.

ii) How many threads will you create for the CPU-intensive portion of the application? Explain.

2. **Synchronization** (10 Marks)

a) **Question: 6.22** (6 marks)
Consider the code example for allocating and releasing processes shown in Listing 1.

a) Identify the race condition(s).

b) Assume you have a `mutex lock` named `mutex` with the operations `acquire()` and `release()`. Indicate where the locking needs to be placed to prevent the race condition(s).

c) Could we replace the integer variable

```
int number_of_processes = 0
```

with the atomic integer

```
atomic_t number_of_processes = 0
```

to prevent the race condition(s)?

Listing 1: Allocating and releasing processes

```
#define MAX_PROCESSES 255
int number_of_processes = 0;

/* the implementation of fork() calls this function */
int allocate_process() {
  int new_pid;

  if (number_of_processes == MAX_PROCESSES)
    return -1;
  else { /* allocate necessary process resources */
    ++number_of_processes;
    return new_pid;
  }
}

/* the implementation of exit() calls this function */
void release_process() {
  /* release process resources */
  --number_of_processes;
}
```

b) **Question: 6.31** (4 marks)

Design an algorithm (using pseudocode in Listing 2) for a monitor that implements an alarm clock that enables a calling program to delay itself for a specified number of time units (*ticks*). You may assume the existence of a real hardware clock that invokes a function `tick()` in your monitor at regular intervals.

Listing 2: Monitor

```
monitor alarm {
    condition c;

    void delay(int ticks) {
        // TODO: implement delay
    }

    void tick() {
       // TODO: implement tick
    }
}
```

3. **Deadlock** (10 Marks)

a) **Question: 8.3** (6 marks): Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	ABCD	ABCD	ABCD
T_0	0 0 1 2	0 0 1 2	1 5 2 0
T_1	1 0 0 0	1 7 5 0	
T_2	1 3 5 4	2 3 5 6	
T_3	0 6 3 2	0 6 5 2	
T_4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

a) What is the content of the matrix *Need*?

b) Is the system in a safe state?

c) If a request from thread T_1 arrives for (0,4,2,0), can the request be granted immediately?

b) **Question: 8.18** (4 marks):
Which of the four resource-allocation graphs shown in Figure 1 illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution.

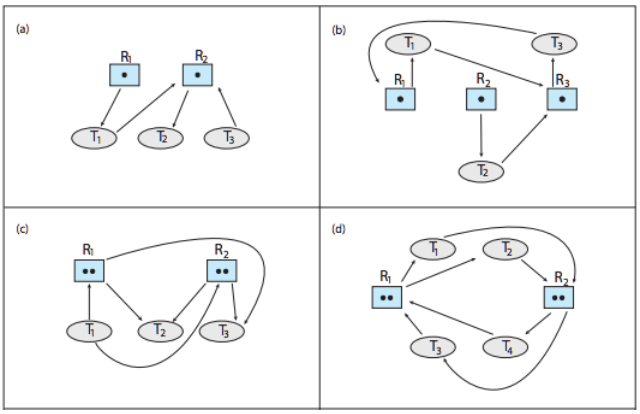

Figure 1: Resource-alocation graph