Operating Systems (10904-01 and 10904-02) **FS 2023**

University of Basel, Department of Mathematics and Computer Science

Lecturers:	Prof. Dr. Florina M. Ciorba	florina.ciorba@unibas.ch
	Dr. Ahmed Eleliemy	ahmed.eleliemy@unibas.ch
Assistants:	Thomas Jakobsche	thomas.jakobsche@unibas.ch
Tutors:	Reto Krummenacher	reto.krummenacher@unibas.ch
	Agni Ramadani	agni.ramadani@unibas.ch
	Selaudin Agolli	s.agolli@unibas.ch

Exercise 3: Synchronization (10 points)

Given: April 14, 2023
Deadline: May 02, 2023

Objectives

- Understand the producer-consumer problem (semaphores and mutex locks).
- Investigate the dining-philosopher problem and report on the output.
- Correct the dining-philosopher problem and report the correct output.
- Investigate and correct example code that contains problems similar to deadlocks.

Tasks

- Task 1: Bounded-Buffer and Producer-Consumer (3 points)
- Task 2: Dining-Philosopher (3 points)
- Task 3: Problem Investigation (2 points)
- Task 4: Synchronization Problems (1 point)
- Task 5: Deadlock vs. Starvation (1 point)

Instructions

- You can solve this exercises in teams of two.
- Submit the solution of each task with detailed comments that clarify your solution.
- Show your solution and upload it to https://adam.unibas.ch.
- Provide all deliverables as an archive file.
- In total, at least 65% of exercise points have to be obtained (with a min of 30% of each exercise).

Task 1: Bounded-Buffer and Producer-Consumer **(3 points)**
In this task you will work on the bounded-buffer problem using the producer-consumer model. Producers and consumers (running as separate threads) move items to and from a buffer with a fixed size. T1.c contains the code without the necessary synchronization.

Hint: In this bounded-buffer example producers should stop producing when the buffer is full, and consumers should only consume items that are actually in the buffer.

To compile the code: **gcc -o T1 T1.c -lpthread**
To execute the code: **./T1 <duration> <producer threads> <consumer threads>**

i) Execute T1.c with the parameters below, report the output and explain the problems.

- ./T1 10 5 0
- ./T1 10 0 5

ii) Correct the code by inserting the necessary synchronization, execute your corrected code with the parameters below, report the output and explain the correct process of the producer-consumer model. **Hint:** You can use counting semaphores and mutex locks.

- ./T1 10 5 0
- ./T1 10 0 5
- ./T1 10 2 2

You must use the given source file T1.c as your starting point. All you need is to implement the open TODOs in the code.

Task 2: Dining-Philosopher **(3 points)**
In this task you will work on the dining-philosophers problem using condition variables. Philosophers spend their lives alternating between thinking and eating, thinking and eating, etc. They occasionally try to pick up forks to eat from a bowl at the center of the table. They can only eat when their neighbors are not eating.

Hint: If you do not see the "DINNER IS OVER" message at the end of the program, then something is wrong and your code might encounter a deadlock. Deadlocks might not always occur, so try to run your code multiple times to be sure.

To compile the code: **make all**
To execute the code: **./diningphilosophers**

There are multiple files in this task. All you need is to implement the open TODOs in the code (main.c and dining.c).

Task 3: Problem Investigation (2 points)

Investigate the code example given below, in Listing 1. What is the name of the problem and how can you solve it?

Listing 1: problem example

```c
// thread one runs in this function
void *do_work_one(void *param)
{
    int done = 0;
    while (!done)
    {
        pthread_mutex_lock(&first_mutex);
        if (pthread_mutex_trylock(&second_mutex))
        {
            // do some work
            pthread_mutex_unlock(&second_mutex);
            pthread_mutex_unlock(&first_mutex);
            done = 1;
        }
        else
            pthread_mutex_unlock(&first_mutex);
    }
    pthread_exit(0);
}

// thread two runs in this function
void *do_work_two(void *param)
{
    int done = 0;
    while (!done)
    {
        pthread_mutex_lock(&second_mutex);
        if (pthread_mutex_trylock(&first_mutex))
        {
            // do some work
            pthread_mutex_unlock(&first_mutex);
            pthread_mutex_unlock(&second_mutex);
            done = 1;
        }
        else
            pthread_mutex_unlock(&second_mutex);
    }
    pthread_exit(0);
}
```

Task 4: Synchronization Problems (1 point)

Describe the classical synchronization problems and tools to solve them.

Task 5: Deadlock vs. Starvation (1 point)

Describe the difference between deadlocks and starvation.