# CS311 Operating Systems
# Assignment 03: Concurrency and Synchronization *
# Due : 17th December, 2023.

FCSE
GIKI

Fall 2023

## 1 Objectives

1. Practice synchronization in parallel multi-threaded code.

## 2 Description

You are tasked with helping your TA schedule his office hours. The TA is teaching 2 classes this semester, **OS** and **PP (Parallel Programming)**, and is holding shared office hours for both classes in his office. Our TA is a popular TA and is overwhelmed by the large number of students showing up for his office hours to get help, so he decides to impose several restrictions.

1. The TA's office has only 3 seats, so no more than 3 students are allowed to be simultaneously present inside the TAs office. When the office is full and new students arrive they have to wait outside the office.

2. The TA gets confused when helping students from OS and PP at the same time.[1] He decides that while students from OS are in his office, no students from PP are allowed to enter, and the other way around.

3. The TA gets tired after answering too many questions. He decides that after helping 10 students he needs to take a break before he can help more students. So after the 10th student (counting since the last break) enters the TA's office no more students are admitted into the office, until after the TA's next break. Students that arrive while the TA is taking his break, or the count of 10 since the last break has reached, have to wait outside the office.

The goal is to let the TA finish with all students attending his office hour as quickly as possible while following the restrictions specified above. Correctness is the most important criterion, but reducing thread overhead is also important.

*Inspired from one Prof. Schroeder's assignments (`http://www.cs.toronto.edu/~bianca/`)
[1]Context switching is hard for humans too!

### 2.1 Working

We have provided a framework for the simulation, which creates a thread for the TA and a thread for each student who wants to attend the office hour. You will need to add synchronization to ensure the above restrictions.

The student threads are implemented in the functions `class_os_student()` and `class_pp_student()`, which simulate students from OS and PP, respectively. After being created a student from OS executes three functions: he enters the office (`class_os_enter()`), he asks questions (`ask_questions()`) and then leaves the office (`class_os_leave()`). A student from PP calls the corresponding functions `class_pp_enter(), ask_questions() and class_pp_leave()`. All synchronization between threads should be added in the `..._enter()` and `..._leave()` functions of the students and the function TAthread(), which implements the TA.

The simulation framework is implemented in the provided file `ohours.c`. The program expects as an argument the name of an input file which controls the simulation. The input file specifies the arrival of students and the amount of time students spend in the TA's office, and the class they belong to. More precisely, the file has one line for each student containing two numbers and a string. The first number is the time (in seconds) between the arrival of this student and the previous student. The second number is the number of seconds the students needs to spend with the TA. The last element is a string indicating the student class. You can assume that the input will always be provided correctly.

The provided code implements all the functionality for the students and the TA, but does not implement any synchronization. To help you in developing your code we have inserted a number of assert statements that help you check for correctness. DO NOT delete those assert statements. Also, do not make any changes to the functions `ask_questions` and `take_break`. You might want to add additional assert statements, for example for ensuring that the number of students since the last break is less than the limit, or for ensuring that there are not OS and PFUN students in the office at the same time.

You can also *temporarily* comment the assertions to see how a full run may look like.

Before you start your work, compile `ohours.c` and try to run it on this sample input file `sample_input.txt`. This will simulate 3 different students asking questions for 25, 10 and 15 seconds, respectively. The time between the first and the second student arriving is 10 seconds and the time between the second and the last student is 5 seconds.

The test input files can have many more students with varying arrival times as well as questioning times. Try constructing such files to test your program under different conditions.

## 3 Design Document

You will submit a design document titled `design.docx`. Your design document should describe and justify the decisions you made in designing the algorithm and the locking structure. It should also contain a short explanation of why you think the constraints of the assignment have been satisfied, mutual exclusion is guaranteed and concurrency is maximized.

## 4 Rubric

The goal is to ensure all 3 restrictions described above, but we want you to get there step by step.

## 4.1 Submission

You will submit:

1. your modified version of `ohours.c` titled u2021xxx_ohours.c where xxx are the last 3 digits of your registration number.
2. your design document as mentioned in section 3 as MSWord file titled u2021xxx_a3.docx.

## 4.2 Marks

1. As a first step, ensure that there are never more students in the TA office than there are seats. **(20 points)**
2. Then add synchronization to make sure that there are never students from OS and PP in the office at the same time. **(25 points)**
3. Then extend your program to make sure that the TA gets a break after he has helped 10 students, before he helps more students. You can make the TA take a break by making him call the function `take_break`, which will put the TA thread to sleep for 5 seconds. **(25 points)**
4. Then see if you can reduce the synchronization overhead, i.e., eliminate any useless synchronization, if there's any. **(10 marks)**
5. Finally, the design document explanations should be correct, clear, and in sufficient detail. **(20 marks)**

Your program should ensure progress, i.e. if there is no student in the TAs office and the TA is not currently taking a break an arriving student should not be forced to wait. Similarly, if an arriving student is compatible with the students currently in the office, and there is space in office, they should not be forced to wait, unless the TA is due for a break. We do not expect you to guarantee fairness of your solution, i.e. we don't require that waiting students are admitted in first-come-first-serve order or that OS and PP students receive equal turns.

The program should, how ever, finish by printing the message: Office hour simulation done.

## 4.3 Penalties

1. Code doesn't compile **(-100 marks)**
2. Code has warnings **(-20 marks)**. Compile your code with **-Wall** flag.
3. Code has memory leaks **(-20 marks)**, in case you used dynamic memory.
4. Submission not correct, i.e., missing PDF, messed up program output, etc. **(-20 marks)**
5. Late submission: **-20 marks + -10*num_days**

**Marks obtained: = max (0, marks + penalties).**

# 5 General Info

This is an individual assignment. You are not allowed to copy code from friends or internet.

You are allowed to use any of the POSIX synchronization primitives that are available for threads, i.e. mutexes, semaphores and condition variables. You can read this tutorial [2] for a description and examples on mutexes and condition variables. For POSIX semaphores you can take a look at this document [3].

There is of course always your fantastic course book!

## 5.1 output

Do not change the output statements of the program. You can add as much output as you desire for your own debugging but make sure that you remove them from the final submission.

In case of any confusion, refer to the provided sample implementation and run it on the accompanied input file.

Marks obtained: = max (0, marks + penalties).

[2] `https://computing.llnl.gov/tutorials/pthreads/`
[3] `http://www.csc.villanova.edu/~mdamian/threads/posixsem.html`