

Lecturers:	Prof. Dr. Florina M. Ciorba Dr. Osman Simsek	florina.ciorba@unibas.ch osman.simsek@unibas.ch
Assistants:	Thomas Jakobsche	thomas.jakobsche@unibas.ch
Tutors:	Reto Krummenacher Agni Ramadani Tom Rodewald	reto.krummenacher@unibas.ch agni.ramadani@unibas.ch tom.rodewald@unibas.ch

Exercise 2: Process Management

(10 points)

Given: March 22, 2024

Deadline: April 16, 2024, 10:00

Objectives

- Understand the fork-join model for processes and threads.
- Learn multi-process programming on shared-memory systems.
- Understand inter-process communication on shared-memory systems.
- Learn multi-threaded programming using Pthreads.
- Familiarize with concurrency issues in a multi-threaded code.
- Understand how to implement CPU scheduling policies.

Tasks

- Task 1: Multi-process programming (1.5 points)
- Task 2: Multi-threaded programming (1.5 points)
- Task 3: Hybrid multi-process and multi-threaded programming (1 point)
- Task 4: Concurrency (2 points)
- Task 5: CPU Scheduling (3 points)
- Task 6: Parallelism vs. Concurrency (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> with all deliverables in a ZIP folder with the naming scheme: *[GroupID]_Ex[SheetNo]_LastName1_LastName2*.
- In total, at least 65% of exercise points have to be obtained (with a min of 30% of each exercise).

Task 1: Multi-process programming (1.5 points)

In this task, you are given an integer input array. You must write a C program where the process executing the program will fork another process. Now, you have the parent process and the newly forked child process.

You must make one process calculate the mean value of the given array, while the other process calculates the median value of the array.

The child process must communicate the value it calculates to the parent process, while the parent process must display both, the mean and the median to the user. You also have to print the involved PIDs (Process IDs).

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

Task 2: Multi-threaded programming (1.5 points)

This task is similar to Task 1 except that you must do it using threads instead of processes. One thread must calculate the mean, while the other one calculates the median. You also have to print the involved TIDs (Thread IDs).

You must use the given source file T2.c as your starting point. You need to implement the open TODOs in the code.

Task 3: Hybrid multi-process and multi-threaded programming (1 point)

This task combines Task 1 and 2. Fork multiple processes, each process with a different number of threads. Print the PIDs and TIDs.

You must use the given source file T3.c as your starting point. You need to implement the open TODOs in the code.

Task 4: Concurrency

(2 points)

In this task, you must create a team of threads that accumulate the values within an input array to a common global variable. The main thread should display the value of the global variable once all threads finish. (**Hint:** You can solve this with mutex locks.)

You must use the given source file T4.c as your starting point. You need to implement the open TODOs in the code.

Task 5: CPU Scheduling

(3 points)

Given T5.c, one can find the following functions: `init_processes` and `display`. Both functions accept a struct array as an argument. The struct is called `process`, and it holds the following information: `process_id`, `arrival_time`, `execution_time`, `start_time`.

The `init_processes` function initializes the struct array.

The `arrival_time` and `execution_time` are randomly initialized, while `start_time` is initialized to -1. The `display` function prints the values of the items within the struct array.

You must implement two functions: `schedule_FCFS` and `display_average_waiting_time`.

- the `schedule_FCFS` function calculates the `start_time` of each process. The function assumes that first-come-first-serve policy is applied to schedule all the processes.
- the `display_average_waiting_time` function calculates the average process waiting time.
- assuming that all processes arrived at time 0,
 - implement `schedule_SJF` that applies the shortest-job-first policy
 - implement `schedule_LSF` that applies the longest-job-first policy

You must use the given source file T5.c as your starting point. You need to implement the open TODOs in the code.

Task 6: Parallelism vs. Concurrency

(1 points)

Explain the difference between parallelism and concurrency for this exercise.