University of Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

University of Stuttgart, Summer 2023

Exercise 6: Concurrency and Control Abstraction

(Deadline for uploading solutions: July 20, 2023, 11:59pm, Stuttgart time)

The materials provided for this homework are:

- a pdf file with the text of the homework (this document);
- a zip file with **the templates file and folder structure you must use for the submission**.

```
exercise6.zip
 ├─ task_1
 │   ├─ example.txt
 │   ├─ program_1.txt
 │   ├─ program_2.txt
 │   ├─ program_3.txt
 │   ├─ program_4.txt
 │   └─ program_5.txt
 ├─ task_2
 │   └─ main.c
 ├─ task_3
 │   ├─ add.c
 │   ├─ sum.c
 │   ├─ max.c
 │   └─ matmult.c
 └─ task_4
     ├─ program_1.js
     ├─ program_2.js
     ├─ program_3.js
     └─ program_4.js
```

Figure 1: Folder structure to use for submission.

The submission must be compressed in a zip file (No ~~rar~~, ~~7z~~, ~~gz~~...) using exactly the folder structure and names in Figure 1.

1 Task 1 (10% of total points of the exercise)

In this task, you are presented with five files containing excerpts of concurrent Java programs. It is assumed that the declarations in the initial lines sequentially initialize the variables at the beginning of the program. Subsequently, the methods ”thread1” and ”thread2” are executed in separate threads, and the program awaits their completion, typically through the use of the ”join()” method.
Each file includes four questions after the code. The first question inquires whether a data race exists, to which you should respond with either ”YES” or ”NO”. Following that, you are requested to indicate the line numbers where the data race occurs, as displayed in your text or code editor. Additionally, you should identify the variable names involved in the race. Lastly, provide all the potential values for the given variable at the end of execution. Use the file example.txt as a reference for the format of the answers.

2 Task 2 (30% of total points of the exercise)

In this task, you are provided with the code for a producer-consumer system, and the objective is to introduce synchronization using mutex to ensure proper functionality of the code.
The producer-consumer system consists of three producers and three consumers. The producers, denoted as P1, P2, and P3, generate characters R, G, and B, and add them to `producer_array`, which is a variable shared between all the threads. P1 produces R and G, P2 produces G and B, and P3 produces B and R. The producers are required to maintain a sequence of the form RGBRGB...RGB when appending the produced characters to `producer_array`.
The consumers, denoted as C1, C2, and C3, remove the last character from `producer_array` and add it to `consumer_array`, which is a variable shared between all consumer threads. The consumers must ensure that the resulting sequence in `consumer_array` follows the pattern RGBRGB...RGB.
The code for this system is provided in the file main.c within the task_2 folder. You can compile the code using the command: `gcc -pthread main.c -o main`. Upon running the compiled file, you will observe that the produced sequence in both the producer-array and consumer-array does not adhere to the desired format (RGBRGB...).
Your task is to introduce the necessary synchronization using mutexes to ensure that the code generates the desired output. It is important to note that you are not permitted to modify the system's logic or make changes to the existing lines of code. **You are only allowed to add synchronization lines, i.e., declarations and usages of mutexes.**
Hint: Use `pthread_mutex_t` from the `pthread` library. Your code will be tested using **C17 and GCC 9.4**.

3 Task 3 (30% of total points of the exercise)

This task involves parallelizing existing sequential C programs using OpenMP. You are provided with four small C programs, and your objective is to insert the appropriate OpenMP pragmas to parallelize them. The regions of the program that should execute in parallel are indicated through comments in the original source code of the programs. To submit your solution, please modify the files within the task_3 folder.
To compile and run both the original and parallelized programs, execute the following command (on Linux): `gcc -fopenmp my_file.c -o my_prog`, where `my_file.c` represents the name of the file to be compiled, and `my_prog` represents the desired name for the compiled file.
The parallelized programs will be compiled using **C17 and GCC version 9.4**. Modifying the program semantics beyond parallelization or adding/removing code outside of OpenMP pragmas will result in a failed task.

4 Task 4 (30% of total points of the exercise)

This task focuses on using promises and async/await to handle long-running functions. The task_4 folder contains multiple files, each containing the implementation of a function that currently uses callbacks and/or promises. Your objective is to transform these functions as follows:

- For program_1: Transform the code inside ExecuteTasks to use promises instead of callbacks. Refer to the example `computeSum` given in the lecture for help. A test case is given at the end of the file.
- For program_2: Transform the code inside `executeTasks` to use async/await with promises instead of callbacks. Again, refer to the example given in the lecture for help. Two test cases are given at the end of the file.

- For program_3: The functions `makeHTTPRequest`, `parseResponse`, and `processData` already return promises. Your task is to transform the code of the `fetchData` function from callback style to promise style, leveraging the fact that the three called functions already return promises. Two test cases are given at the end of the file with the expected output.
- For program_4: You are given a function that already uses promises. Your goal is to call the functions `getUserInfo, getUserComments, getUserPosts` in conjunction with `await` and adjust the code accordingly. In other words, suggest another implementation of the function `getUserDataOld` using `await` mechanism. Two test cases are given at the end of the file.

Please refer to the comments and instructions provided within each file in the task_4 folder for more details and guidance.