Iran University of Science & Technology

Operating Systems
Fall 2020

https://os-course.github.io/fall20

#3 Assignment - Fork and Threads

Download

Question 1

A web server is an application which can handle requests over the web. One of the characteristics that a web server should have is capable of responding to many clients within the low latency. E.g. consider Google as a web server which millions of clients can get their response simultaneously. A very simple web server program is attached to this homework. This web server only could response to clients one by one. Modify this program to address this issue.

In this question, you should modify the attached file in `q1` directory. Please Include the modified file in your submission with this exact name and directory.

- Step 1: Compile and Run the `server.c`
- Step 2: In your browser: http://localhost:8090/
- Step 3: Add `fork` to `server.c`; Then goto step 1.
- If you get the error: `In bind: Address already in use` try to change PORT number in the file and then recompile and run.

Question 2

In `q2` directory, there is another C code just like the question 1. Use `pthread` library in order to implement a multi-threaded server instead of multi-process server.

- Step 1: Compile and Run the `server.c`
- Step 2: In your browser: http://localhost:8090/
- Step 3: Include `pthread` library and add necessary changes to `server.c`; Then goto step 1.
- If you get the error: `In bind: Address already in use` try to change PORT number in the file and then recompile and run.

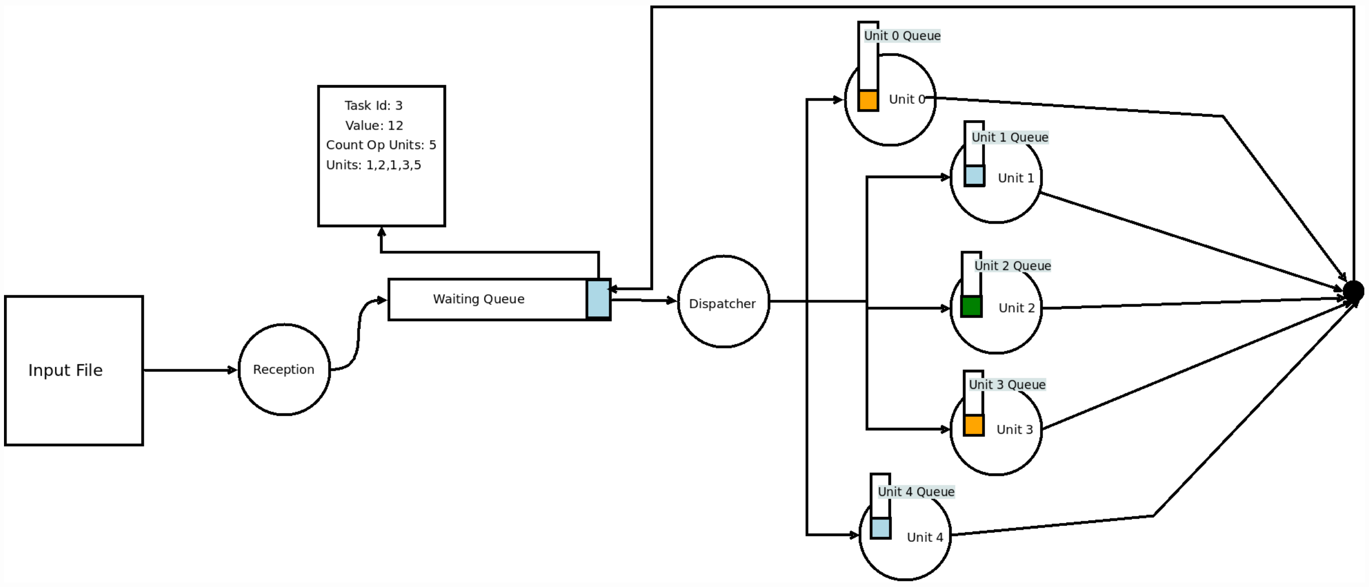
Question 3

With completion of the question 1 and 2 you should be able to differentiate the multi-process and multi-thread programming. Write a report and submit it in **PDF** format, using the questions 1 and 2, try to explain what is different between these two approach, which are the advantages and disadvantages of each solution. Also, try to understand how variables look like in memory.

Question 4

Description

Imagine a complex system like an office which has several processing units for doing tasks. In this office, there is a receptor whose job is queueing input tasks by order of reading them from the input. Then, a dispatcher should send these tasks to the corresponding unit of execution. The unit of execution does its own operation on the task, and if there are other associated units of executions, the task is returned to the waiting queue, otherwise, the task is done. Following figure illustrates the system structure.



Task Structure

The structure of task should have following information. You are allowed to add other attributes to it if you find it necessary. Note that the list of operations are **stack-like**. It means that the first unit to execute is the first element of its array.

```
struct task {
    int id;             // id of this task
    int value;          // value of this task
    int atime;          // the arrival time of the task
    int unit_count;     // number of units
    int *unit_id;       // list of units which task will be assigned to
}
```

Units and Operations

Each unit of execution has its own operation which will be done on the value of tasks. For example, the unit 0 adds 7 to the value of the task then modulates the new number with `M`. `M` is a constant value which should be defined in your program with value of 10000. The reason of using modular division is to avoid value overflow. Each of these units are implemented as threads. Also, these units have their own queue which these queues are filled by the dispatcher. The units should go to sleep for 0.5 second after doing the task.

Unit id	Operation	Desc.
0	+7, %M	Adds 7 to the value then modulate by M
1	*2, %M	Multiplies by 2 then modulate by M
2	^5, %M	Calculates the power 5 and then modulate by M
3	-19	Subtracts the value from 19
4	print	Prints out the value of the task

Input file and Receptor

The Receptor should read an input file and construct struct of each task, and determine the arrival of the task, then put that task in the `waiting queue`. A thread-safe structure is required for implementing waiting queue. The structure of input file is as follows:

```
task-id task-value units-count unit-id-1 unit-id-2 ...
```

Example of Input file

Two other examples are also attached.

```
0 123 4 0 0 4 2
1 78 3 1 2 4
2 -3 7 0 1 3 0 2 3 4
...
```

Print Unit

The output is generated by the `print` unit. The total time of being in the system until the `print` unit should be shown. The format of out put should look like the following. **POINT: to label arrival time and the duration of being in the queue you can use `clock_gettime` or `gettimeofday`. (what is difference? `man`)

```
<time stamp> tid: <task_id> value: <value>
13 tid: 0 value: 123
18 tid: 0 value: 137
...
```

Final Word!

On the first impression, this question might seem hard to you, but this is not true. Please study the assignment carefully and be sure it is not a hard one at all.

Despite ALLOWANCE of copying from each other, we are highly demanding and recommending that everyone complete this question on **his/her own**; because there are hundreds of ways to implement this assignment, and it is not **appropriate** that all answers look same.

Do not try to waste your time by searching for this assignment on the internet. We don't believe you could find anything similar!

Deadline

- **Saturday** 28th Nov. 23:00

Submission

Submit just a zip file ,containing your codes and a report, in LMS. The file should be named as [`9752xxxx.zip`]. For example `97521234.zip` .

Iran University of Science and Technology

School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran

webpages.iust.ac.ir/msharifi/