Iran University of
Science and Technology

School of Computer Engineering

Dr. Reza Entezari Maleki

Fall 2021

Operating Systems

Practical Assignment IV

Hadi Sheikhi – Mohammadmostafa Rostamkhani

Deadline: 20 Nov. 2021, 23:59:59

Question 1

Description

In this exercise, we are going to learn more about the concepts of Thread & Synchronization.

In this question, you need to implement the **π** sequence using Threads. In fact, you have to write a program that calculates up to a certain number of terms from the sequence **π**, which is obtained from the following formula:

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \frac{\pi}{4}.$$

Note that your program must be able to work with a number of different threads.

For example, if you calculate up to 10 millionths of this sequence in your program using 10 threads, each thread should calculate one million terms of this sequence.

- ✓ For example, the first thread calculates the terms (0, 10, 20, ...), the second thread calculates the terms (1, 11, 21, ...) & similarly the tenth thread calculates the terms (9, 19, 29, ...).

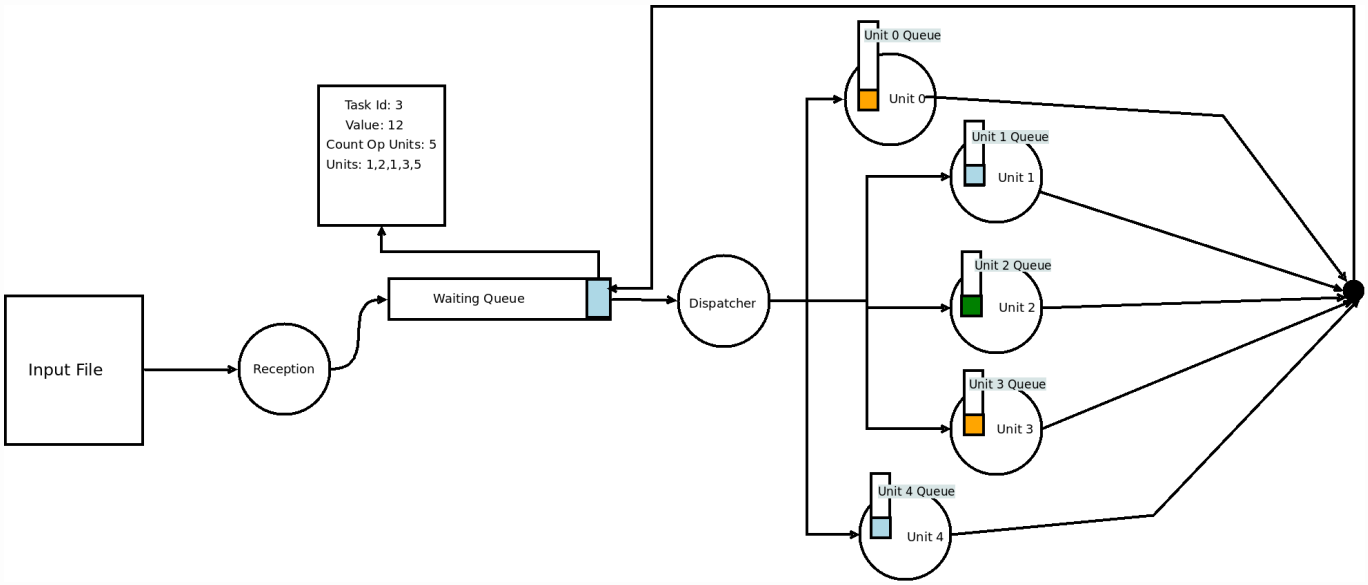
Additional Notes

- There is no problem to use `math.h` library. (You can use the `pow` function to calculate the power in this sequence.)
- Be sure to name your program file as `pi.c`.
- You must call your program using the `-pthread` and `-lm` parameters when you want to link it.

Question 2

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
...
```