Assignment 02: Threads and Scheduling

University of Puerto Rico at Rio Piedras
Department of Computer Science
CCOM4017: Operating Systems

Introduction

In this project the student will simulate a **Shortest Job First** scheduling algorithm for a distributed system that consists of one embedded device and a central computer cluster.

The student will assume that we have one embedded device that generates computational problems that are too heavy to be performed in their hardware, either because the device's battery will drain, they do not have enough memory, nor computational resources to perform them in a timely manner.

We will simulate a computer cluster that will have a head node that receives requests for computing time from the embedded device, will put the jobs in a queue of processes, and then two different compute nodes will "execute" them.

Objectives

- Practice threads implementations
- Practice inter process communication using sockets (TCP) and a shared buffer
- Identify critical regions
- Implementation of mutual exclusion and semaphores

REMARK: Your assignment MUST only use the primitive Locks and Semaphores objects. The queue must also be implemented by the student using a regular python list. **NO other synchronization, or synchronized object library may be used in this assignment.** The use of these libraries will immediately invalidate the assignment, and it will not be graded.

Prerequisites

- Python:

    - www.python.org
- Python threads:
    - https://docs.python.org/3/library/threading.html
    - https://pymotw.com/3/threading/ (good tutorial with examples)
- Python sockets (**TCP**):
    - https://docs.python.org/3/library/socket.html
- Threaded sockets (**You are NOT allowed to use this one for this assignment**)
    - https://docs.python.org/3/library/socketserver.html

Instructions

The embedded device:

The embedded device will consist of **one process (not threads)** that will:

1. Get the processes running in the computer where the simulation is running.
2. For each process obtained in step 1, send a different message to the cluster head node, with a random number that will simulate the time it takes for the process to complete.
    - sleep a **random short** (1 to 5 seconds) random period of time between sends.

For instance if the embedded device gets the processes: top, ls, and cut, then the embedded device will send to the head node the messages:

- "top:3"
- "ls:1"
- "cut:5"

where 3, 1, and 5 are random numbers generated to simulate the time it took to execute each process. Note that in this example the syntax is `Process Name:cpu time`.

Example of how to run the embedded device:

```
#python edevice.py <server address> <server port>
```

where <server address> is the IP address of the cluster, and <server port> is the port number of the cluster.

Remark: Your implementation of the edevice MUST be a process. You MUST use big port numbers less than 65,000

The cluster:

The cluster will consist of three threads:

1. One thread that will simulate the head node:
    - Receives the messages generated by the embedded device and
    - puts the “processes” in the **Shortest Job First** Scheduler queue,
2. and two threads to
    - extract the processes from the queue and “execute” them.

The first thread will act as a **producer to the scheduler** by:

1. Listening to the embedded device messages.
2. Extract the time and process name out of the message and store them in a shared queue.

The other two threads will act as **consumers** by:

1. Picking a message from the shared queue.
2. Keeping a table of the processes attended by each consumer and the total time consumed by each process.
3. Sleep the time extracted from the message picked from the queue.

For instance, let's assume the producer thread receives messages top:3, ls:1,cut:5, and puts them in the queue, and that the consumer 1 extracts top:3, and consumer 2 extracts ls:1,cut:5.

Consumer 1 must have a list [(“top”,3)], total = 3 and consumer 2 must have a list [(“ls”,1), (“cut”, 5)], total = 6.

Example of how to run the cluster:

```
#python cluster.py <server port>
```

Note: The consumers must not poll for messages in the shared queue (repeatedly ask for messages in the queue). The thread **MUST** block (Semaphores) until there is a message in the queue.

Example output of the cluster:

```
Consumer 1 consumed 3 seconds of CPU time
Consumer 2 consumed 6 seconds of CPU time
```

Other notes

Make every configuration variable a global variable. Example:

1. Range of sleep times
2. Range of process CPU time

Rubric

1. Programs that do not run (can't be tested) will automatically obtain 0 points.
2. Documentation of the code and the README (10 pts)
3. Use of an unsynchronized Data Structure for the messages (5 pts)
    a. **Remember that the use of a synchronized data structure will invalidate your project**.
4. Implementation of Semaphores for blocking and not polling or busy waiting (10 pts)
5. Implementation of Mutexes to protect the critical regions. (Only the critical regions) (10 pts)
6. Use of the sleep libraries (5 pts)
7. Implementation of TCP sockets for message communication(10 pts)
8. Working implementation of the cluster server (35 pts)
9. Working implementation of the embedded device. (15 pts)

Deliverables

- The source code of the programs (well documented)
- A README file with:
    - Good description of the program
    - How to use your program, how to execute the processes.

    - Any additional reference used to perform the project and the names of students who helped perform the project.
        - Verbal collaborations are allowed, however any form of code sharing is not allowed.

Bonus

Implement the embedded devices (edevice.py) such that it sends the jobs to the compute server and waits for a response of the compute server instead of just randomly sleep to send the next job.