CEng 536 Advanced Unix
Fall 2025
HW1
Due: 16/11/2025

1 Description

In this homework you will write an IPC socket based tool assignment system for a gym. Assume there are k tools (i.e. a running mill) of a kind in a gym, and a typical person in the gym waits for a tool, use it for a period of time, than rest for a while, than use another turn in the same tool (for simplicity, we assume there is only one kind of tool in the gym). After repeating these in couple of turns, person leaves the gym.

The owner of the gym respects fairness a lot, so asks for an application that makes sure tools are equally shared by the customers when they are fully utilized. The fairness can be established by such set of rules:

- System keeps track of each customers time spent on the tools accumulatively (called *share*)
- Each customer is allowed to use a tool for q time units without intervention once s/he gets the tool.
- After q, the list of waiting customers is inspected if there is a customer with a smaller *share*. If so, customer leaves the tool, and the tool is assigned to the minimum *share* customer waiting in the queue.
- When a new customer arrives, s/he is assigned to the average *share* of all (on tool, waiting and resting) customers in the gym.
- When a customer rests for a long period, his/her *share* may be too small so that s/he can occupy a tool for long periods of time. In order to avoid it, there is a Q value which is the upper limit a customer can use a tool without interrupt. After this limit, customer will leave the tool, and inserted in the waiting list. The smallest *share* customer in the waiting list is assigned to the tool.
- A customer on a tool can keep using the tool after q, until any of the resting customers with a smaller *share* asks to use the tool. When it happens, the using customer immediately leaves the tool.
- A customer on a tool can keep using the tool after Q only if the waiting queue is empty.

Since there are k tools, the assignment should follow the following principles for tool assignments:

- Total tool usage is computed for each of the k tools. When There are more than one tool is available for a customer, the tools with the smaller total usage is chosen. If equal, the smallest numbered tool will be assigned.
- When a new customer arrives, or an existing customer comes back from rest, and all tools are occupied, the tool having the customer with the maximum current usage is considered as a candidate. If candidate customer has a smaller *share*, customer is inserted in the wait queue. Otherwise, if the candidate has used the tool for $t < q$, new customer is inserted into the wait queue. Otherwise, the candidate is asked out of the tool and inserted into the wait queue. The new customer gets the tool.

2 Application

This application will be implemented on a stream socket scenario. The application will be a multiprocess server. There will be a master process accepting connections, each of the k tool processes simulates the tools in the gym, especially handling the timings. In addition, for each connection, an agent process is created to serve the customer session and controls the per customer communication. The customer is assigned the default *share* (first ever customer will be assigned 0). The agent process is responsible for handling the following text commands from the client:

- REQUEST T to use a tool for T units. The agent will assign the customer to one of the tools. If not possible, the customer will be inserted in the wait queue. The client will be notified with the number of the assigned tool when the assignment is done. After T units of net tool use, the customer automatically enters the resting state, and the client will be notified.
- REST to leave a tool before the requested time, and take a rest. During the rest, customer will be in the gym but not on any waiting list. Even if there is an available tool, customer is not going to be assigned. The rest will continue until customer quits or sends another REQUEST command.
- REPORT to get a full report of the gym including number of customers in the waiting list, in the rest, average share, current share, share of customers on the tools etc.
- QUIT to leave the gym. Connection will be closed, customer will be deleted in all lists s/he exists. Agent will terminate. Connection close will have the same affect with QUIT.

Newly arrived customers will be on the resting state. Agents will inform clients in all events like when tool is taken away due to q and Q limits.

Since this is a multi-process implementation, customer information has to be kept on a shared memory data structure. In the minimum case, the number of customers, total share of customers, total usage of k tools, and data structure for the queue of waiting customers has to be kept in a shared memory data structure. For sake of simplicity, there is an upper bound of number of waiting customers which is 1000000. Since the access to the waiting customer list will be based on minimum *share*, you need to use a $\mathcal{O}(log(n))$ complexity data structure, a heap queue or red and black tree.

Tool processes mostly sleep to simulate the session of the user in the tool. After sleep, a tool process makes the current customer leave the tool and picks the next customer in an infinite loop. You don't need to use a process per tool if you like, you can run a main timer process keeping timing of all tools if you like.

Also processes have to synchronize with eachother in the following cases:

1. Tool process informs the relevant agent process when tool is taken away from the customer
2. Tool processes go in idle state when no customer is requesting. In this case, when a new request arrives, they need to be waken up.
3. A tool process is informed by the agent of the assigned customer when there is a `REST` or `QUIT` request.
4. The tool process with the customer with the maximum share, is notified when a smaller share customer is arrived in the waiting queue. (if assigned customer has spent more than q in the current run). Only tool with the maximum current share is informed. In case of a tie, tool process with minimum share is selected.

You can use multithreading within each process but cannot combine master and agent processes, they need to be seperate processes. Multi-threading is especially required as two threads per agent as: one for blocking on socket input, other for waiting notifications from tools.

3 Shared Memory and Synchronization

Since master and the agents will be created via `fork()`, a shared memory created in the master will be inherited by the agents. Anonymous `mmap()` will be a good choice, without any persistency or a backing file.

Its a good idea to put all related parts in logical groups and create one or more shared memory areas as fixed size region, respecting the number of customer limit, and other known paramaters.

Note that, traditional heap variable allocators like `malloc()` and `new` will not work for shared memory. You need to write simple allocators for the task. For example you can have 1000000 customer structures in an array, and a free block array as part of the shared memory. You can use integer indices of customer structures, instead of pointers.

Another tricky part is the use of synchronization. Semaphores work for multiple processes but there is no substitute for condition variables (signals work but they are harder to control). A solution is to use `pthread` with mutex and condition variable attribute `PTHREAD_PROCESS_SHARED` is set. If mutex and/or condition variable is stored in the shared memory, and this flag is set, you can use `pthread` based calls. See `man pthread_mutexattr_setpshared`, `pthread_condattr_setpshared`. Pthread library also provides timeout based blocking, making timing and waiting possible in a single call.

4 Execution

Your code will compile into `hw1`. The following command line arguments are going to be supported:

```
./hw1 conn q Q k
```

`conn` is the address to be listened by the master process. If it starts with a `@`, it is assumed to be a unix path, thus a unix domain socket. Otherwise it should be in `ip:port` format, IP and port of the IPv4 socket address.

The basic unit is a milliseconds, ($1/1000th$ of a second). *`q`* and *`Q`* are integers for minimum and maximum tool usage limit in a single run for a customer as millisconds. *`k`* is the number of the tool processes started at the beginning.

5 Input/Output

Clients typically send `REQUEST` and `REST` requests repeatedly. The agents will send events related to the customer. One of the following output lines will be generated (shown in printf syntax):

```
Customer %d with share %d is assigned to the tool %d.
Customer %d with share %d is removed from the tool %d.
Customer %d with share %d leaves the tool %d.
```

You can take process id of the agent as the customer id.

The output of `REPORT` command is as follows:

```
k: %d, customers: %d waiting, %d resting, %d in total
average share: %.2f
waiting list:
customer   duration  share
---------------------------
%-12d %10d %12d
```

```
Tools:
id   totaluse currentuser share duration
--------------
%-5d %12d %-12d %10d %12d
```

Customers are reported by id of their agent processes. The time customer inserted in waiting list is preserved in milliseconds and report produces a duration column by subtracting insertion time from current time. The report is ordered by "share" in ascending order.

The tool ordering is done via tool index, from 0 to $k-1$. Each tool reports the total use, current user id, share and remaining duration left for the user. If tool is free, just report FREE.

6 Submission and Questions

Use odtuclass for your submission. Provide a `tar.gz` file containing a `makefile` that compiles the binary `hw1` in the current directory (some fancy tools create subdirectories for compiled binary, do not do that). You can use C or C++ with standard libraries. No other library then pthread (-lpthread -lrt) is allowed.

During the evaluation I will execute:

```
tar xzvf yoursubmission
pushd hw1
make hw1
# all my tests here
popd
```

in a loop. Any submission breaking this makes my life harder.

You can ask any homework related questions on the odtuclass forums.

I am going to provide implementation of tree like data structures that are compatible with shared memory, written by "Gemini", since they are out of the scope of this course. Use of generative AI, other than simple completion and syntax correction is prohibited as well as any other means of code sharing.