CEng 536 - Advanced Unix

Fall 2017-2018
HW 2
Due: 13/11/2017, 23:55

1 Description

In this homework, you are going to implement a notification based broadcast messaging system with IPC and sockets. Your messaging service will work on a TCP or Unix domain socket and accept connection requests. Each connection will be served by a seperate process that we will call *agents*. Clients connecting to service will send text based commands and get text based results and notifications as long as they keep the connection open.

Messaging service will work as a shared bulletin board. Each message that is sent by a client will be added to a shared message list. All clients can get the list of messages. Each message has a 256 bytes maximum length. Total number of messages (maxnmess) and sum of the lengths of all messages (maxtotmesslen) have a upper limit that will be given as a command line parameter. When either of these limits are reached, system will cycle back and start overwriting oldest messages. For example, if system has a maxnmess = 100 and maxtotmesslen = 10000. If there are 100 messages of average length 90, message 101 will be written on first message and continue from there. If there are 50 messages of average length 200, next message will overflow total message length and message 51 will overwrite the first message.

The clients can retrieve the last n messages with a command described below. In addition, clients can register an arbitrary word that they are interested in. Whenever any client sends a message containing this word, message will be sent to the client by the agent it is connected to. Words are alphanumeric case insensitive strings that are delimited by a sequence of space, tab, and punctuation characters. There is a limit of 100000 for sum of number of followed words for all clients at any instance of time. The follow request for the message 10000+1 is rejected. Also you can assume an upper limit of 10000 concurrent connections in the system.

By the nature of the service, agents in different processes has to keep messages and the followed words registry on a shared memory. All operations on the shared data structures should be safe for synchronization, deadlock free, responsive and efficient for CPU (no busy waiting/polling allowed).

Each agent reads text based service requests from client and serves them. Each command is given on a single line ended by '\n'. The following commands should be implemented:

`SEND` *`single line message`*

Add the message (without the space after `SEND` and '\n' at the end) to the message board. Take necessary actions to notify the agents that has one of the words in the message in their interest list.

`LAST` *`n`*

Get the last *n* messages in the board. If 0 is given, all messages are retrieved. All messages are written on a single line in clients socket.

`FOLLOW` *`word`*

Add the *`word`* to the words of interest for the client. All following `ADD` requests by any agent that contains this word will be written the clients connection. *`word`* should be on a word boundary in the sent message (clients following "berry" will not get "strawberry"). The matching messages that are sent before this request are ignored. Notification from followed messages have first occurence of the word of interest is enclosed in "`[...]`". When more than one word is interested in and message contains two or more of it, message is repeated for each matching word. When client asks to follow same word again an error ("`You are already following it`") is generated.

`UNFOLLOW` *`word`*
: Remove the *`word`* from the words of interest for the client. No error reporting is needed if client is not following the *`word`*.

`FOLLOWING`
: List the words that the client follows.

`BYE`
: Same as shutting down the connection (terminating the client etc). Closes the socket and cleans up all information about the client (`FOLLOW` information basically).

All successfull commands other than `LAST` is replied by a "`<ok>`" line.

2 Sample Connection

A sample client session is given below as a hint to output formats.

```
SEND hello everyone i am the one
<ok>
LAST
hello i am the first client
hello everyone i am the one
FOLLOW unix
<ok>
LAST 1
hello everyone i am the one
first client uses [unix] if you are interested in
LAST 3
hello i am the first client
hello everyone i am the one
first client uses unix if you are interested in
FOLLOW linux
<ok>
SEND Is Linux different than UNIX?
<ok>
Is Linux different than [UNIX]?
Is [Linux] different than UNIX?
UNFOLLOW unix
<ok>
LAST 2
Is Linux different than UNIX?
Hello I am someone else just sent a message about unix.
FOLLOW Linux
You are already following it
BYE
```

Only the order of messages in the board is significant. The order of replies and notifications can be arbitrary.
Command line of your server program will be like:
`./messboard` *`maxnmess maxtotmesslen address`*

If the *`address`* is not a valid integer (you can check result of `sscanf("%d", argv[3])`), it is regarded as a Unix domain socket path. Otherwise, it is taken as an IP port and your server will listen on '0.0.0.0' with the given port. Since it is a connection based application, you need to use stream sockets.

3 Implementation and Evaluation

As long as no deadlock, no busy wait/poll constraints are met, you can use any synchronization primitive and shared memory mechanism in standart library, system V IPC and/or POSIX IPC. No other library is allowed. For each agent, you can use multithreading to listen on client socket and do some other task at the same time. However each connection should be served by a seperate process.
Shared memory is fixed at attachment/mapping time. So you need to use a fixed static storage area for shared data structures. In order to make this memory efficient, instead of allocating $256 \cdot maxnmess$, you can make something more clever. You can allocate a $maxtotmesslen$ bytes area in shared memory and a $maxnmess$ sized integer array to mark start of messages in the message buffer.
Another point of efficiency is the search for a word for the agents following it. Trivial implementation searches all words in the sentence in the the followed word list sequentially (upper limit is 100000). Instead you can implement a static hash to do all search, insert and delete operations efficiently.
Total number of points you can get in this homework is 110. When you skip implementation of memory efficient message, you loose 15 points. When you skip implementation of word hash, again you loose 15 points. As a result, the lazy implementation will get a 80 maximum.
Hint: For implementation of follow notifications, you can get either signals (to related agent) or better than that, you can use condition variables of pthread library. Check for PTHREAD_PROCESS_SHARED attribute in pthread mutexes and condition variables. When the storage of a mutex or condition variable is on a shared memory area and it is initialized with this attribute, you can use pthread calls. For each agent in the system, create a shared condition variable and for all words that agent follows, create an entry from word to this condition variable.
You need to submit a tar.gz (no zip, rar or others) file containing a `Makefile` and sources in directory named your METU userid. `Makefile` will create binary `messboard` for your homework. Following this specifications help me evaluating your homework so please obey them.
You can use `metuclass.metu.edu.tr` to submit your homework.