UNIVERSITÄT DUISBURG ESSEN

NES Networked Embedded Systems

Programming in C/C++

Exercise Sheet 10: Threading

Task 01:

In this task you are going to write a simulated chat system with the possibility of having a few chat bots which can interact in a chat room. The chat will not work over the network, but will use multithreading with each chat bot running in its own thread to simulate a chat room. The bots will have different functionalities and chat modes which will be explained later.

When your program is started, it shall read input from the console continuously in a loop and process it to perform the appropriate action. There are 3 main categories of messages that can be expected:

- If the message starts with keyword "bot", then it is a command to spawn a new bot. The command shall be expected in the following format: "**`bot:<type>,<name>`**".
  - The type represents the type of chat bot to create (there can be multiple instances of same type). If the type of bot does not exist, the message can be ignored. See the table below for the different bot types and how they should be implemented
  - The name is the display name of the bot in the chat room. The name shall only contain alphanumeric characters (but you do not have to do a check for this).
- If the message is "exit" then the program should be properly terminated. There should be a 6 second delay before termination to give the bots time to finish processing.
- If the message does not conform to the patterns above, then it is normal text and should be published in the chat room i.e. forwarded to all the bots in the system. This means the above 2 message categories are not broadcast as text, but trigger only the appropriate actions.

After each message is processed, the main thread should sleep for 100 milliseconds before accepting the next input. The main loop represents the chat room and the bots must not run inside the main thread. The following design guidelines should be followed in creating the chat system:

- It is a broadcast-chat system. Everything you type in the console or which a bot sends should be received by everyone in the chat room. This implies the bots can interact with each other in respect to their defined functionality
  - Example: An echo bot is also meant to repeat the messages of other bots
- Spawning new bots should happen in the main thread exclusively.
- The exit-call is exclusive to the main-thread too. If a bot sends the "exit" message, it is just treated as a normal message. Special commands are only interpreted from the console.
- The bots never leave the chat unless the user triggers program termination.
- For better maintainability, the bots should be implemented as classes:

- There should be one abstract base class for bots dealing with the chat-related functionality of the bots
- The actual bots are derived from this abstract class (e.g. one class per type) and should ideally just handle the actual message-processing
- The base class can have static functions and variables, for example to keep track of all bots and distribute messages to them.

- The main-thread (which processes the user-input) is not required to be designed as a class, it is acceptable to just have a loop reading and processing user-input in the `main()` function
- Program termination is not required to be immediate after the 6-second delay. It is also acceptable if a few bots still finish their tasks and send and residual messages before exiting. But the application must finish eventually (in a clean way of course – no leftover threads can run upon exit)
- Messages sent by the user should not be printed out in the console window again

Bot Types

Type	Functionality	Example

echo	Every incoming message not containing a question mark (?) should be instantly repeated by this bot, but with a question mark appended to the message.	`bot:echo,Parrot`
		`Hello`
		`Parrot: Hello?`

counter	Every time "`startcounter`" is read, this bot should to count from 1 to 3 seconds. (with respect to the real time).
	The command must not have anything before or after it, e.g. it should not react to "Astartcounter" or "startcounter?"
	The first number should be sent instantly (So in total the timer actually runs 3 seconds).
	It should be possible to start multiple timers concurrently.
		`bot:counter,Tim`
		`startcounter`
		`Tim: 1`
		`Tim: 2`
		`Tim: 3`

delay	Listens for a message in the format "`delayed <txt>`" (without any prefix, e.g. it should not react to "A delayed message"), which makes it wait for 2 seconds and then send the given text again from the bot.
	It should be possible to issue multiple delayed messages concurrently.
	The message given by <txt> must not be altered.
	It is not required for the bot to react on its own messages. It is up to your implementation if it does ("delayed delayed msg").
		`bot:delay,Bob`
		`delayed A text`
		`Bob: A text`

prime	On the message "isprime a b …" it should check if the given numbers are primes. Checking should be concurrent in the sense that every number is checked in a separate thread and the result for each number is published as it becomes ready.
	The results are to be sent in the format
	"<num> is a prime" or "<num> is not a prime"
		`bot:prime,Tom`
		`isprime 7 37619 8`
		`Tom: 8 is not a prime`
		`Tom: 7 is a prime`
		`Tom: 37619 is a prime`

quiz	Holds a quiz based on questions provided in a file.
	The quiz is started on the command "startquiz <filename>".
	The bot then sends the first question. Once the answer was posted it sends out "Correct!" and goes on with the next question in the file. When there are no more questions, the bot should again wait for the startquiz command.
	Each quiz-bot can just run one quiz at a time.
	The quizfile-format is specified below.
	If the file was not found or unexpectedly ends, the bot again just waits for "startquiz" – no error message should be printed.
		`bot:quiz,Jim`
		`startquiz quiz.txt`
		`Jim: Question A`
		`Abcd`
		`Cool answer`
		`Jim: Correct!`
		`Jim: Question B`
		…

Quizfile-Format:
Each quizfile should contain an even amount of lines, every odd line contains the question while the following line contains the expected answer. If the file contains an odd amount of lines, the quiz should at least process all valid Questions. In this example the bot should ask Question A&B but ignore the “Invalid question…”:

```
Question A
Cool Answer
Question B
Another Answer
Invalid question because answer is missing
```

Submission: Submit source code following submission guidelines.
You executable program should be called **`chat`**
Points: code 80 pts, comments 20 pts