Systems Programming

B.Sc. (Hons) in Software Development

Assignment 2 (Home)

Implement a low-level program

Instructions to Students

Read the following instructions carefully before you start the assignment. If you do not understand any of them, ask your lecturer for further explanation.

- This Home Assignment has a total weight of **50%**.
- Copying is strictly prohibited and will be penalised through a referral and other disciplinary procedures as per MCAST Policies, Procedures, and Regulations.
- Upload all your files (code, build scripts and documentation) to MAST VLE as instructed by your lecturer.
- Each task has the marks associated with it clearly indicated, with a complete marking scheme at the end of the assignment.

An individual interview may be held upon assignment submission. You are expected to be able to explain your choices and code in all tasks to achieve the marks associated with each task.

Scenario description: Basic fitness monitor

You are tasked with implementing software for a simple fitness monitor with no display. The device will offer the following features:

- RATE – Reads the heart rate in beats per minute.
- TEMP – Reads the skin temperature in degrees Celsius.
- STEPS – Returns the number of steps done today.
- STATS – Returns statistics about usage and battery level.
- RESET – Resets the device, clearing all its memory.

The architecture of this system features a server running on the fitness band and a client running on a computer to collect data over Wi-Fi and display it to the user. The server on the band keeps a record of all the commands received.

The protocol below documents the communication between client and server. A sample set of request/response pairs can be found in the Appendix.

Request	Response	Documentation
RATE	<number>	Between 0 and 250
TEMP	<number>	Between 30.0 and 44.5.
STEPS	<number>	0 or larger.
STATS	String	A String of text that shows the percentage of battery remaining and some other statistics of times each command (RATE, TEMP, and STEPS) was received by the server since last boot (or reset).
RESET	OK	Resets server memory (clears the record of each command that is sent by the client).
<anything else>	UNKNOWN	Any other request will receive UNKOWN as a response.

Your assignment consists of 3 parts:

A. You are asked to implement a:
   - Server program that implements the protocol above, using random or hard-coded values for the three sensors (RATE, TEMP, and STEPS).
   - Client program that communicates with the server and displays information received to the user.

B. You need to add File I/O features to both client and server.

C. You are to improve the server to allow for concurrent client connections.

Home Part A: Basic Client/Server (31 marks)

KU1.1 Arrange the operation of a multi-source file compilation:

a) Use header files appropriately to optimise compilation (1 mark).
b) Use multiple source files to ensure code modularity (1 mark).
c) No warnings when client and server are compiled with `-Wall` (3 marks).

AA4.1 Produce relevant code to implement a sockets network client (to run on computer):

a) Connect to the server and show a message if connection fails (1 mark).
b) Once connected, it should ask the user to enter a request (e.g., TEMP or STEPS), send it to the server and receive the response (3 marks).
c) Show descriptive strings for values returned (1 mark), e.g.:
    - if TEMP is over 37.8°C print "Fever"
    - between 37.0°C and 37.7°C print “Normal”
    - 36.9°C or less, print “Low”.
d) Properly check return codes from the Sockets API functions, and act accordingly (2 marks).

AA4.2 Produce relevant code to implement a sockets network server (to run on band):

a) Listen for connections using port 54321 (1 mark).
b) Continue listening after each client disconnects (1 mark).
c) Implement the required logic according to protocol specifications (3 marks).
d) Properly check return codes from the Sockets API functions, and act accordingly (2 marks).

KU1.3 Arrange a ‘Make’ or ‘Build’ utility.
Create one makefile with the following targets:

a) **build_all**: builds all executables (default target) (1 mark).
b) **clean_all**: deletes all the compiler generated files (1 mark).
c) **rebuild_all:** performs a clean followed by a build (1 mark).
d) **debug**: builds the debug version of the client and server in directory `/bin/dbg` (1 mark).
e) **release**: builds the release version of the client and server in directory `/bin/rel` (1 mark).

AA2.2 Produce relevant code to implement a data structure.
You are required to store the commands that the server receives in a data structure according to the following specifications:

a) Create a data structure that can fulfil the requirements (1 mark).
b) Implement a data structure to store the commands received by the server (4 marks).
c) Proper and efficient memory management (2 mark).

Home Part B: Advanced Features (19 marks)

A new version of the fitness tracker will feature a small internal storage where all the commands received by the server will be saved to a file for troubleshooting, analytics, and future reference. A new optional feature to log all server responses will also be added to the client.

AA3.2 Produce relevant code to implement File I/O.
Upgrade your server to store the records on internal storage using File I/O operations:

a) Select a file format (e.g., CVS, JSON, XML) to save the records. Justify your selection and include a sample of the file content in the assignment documentation (1 mark).
b) Persist the server records to the file whenever one of the three main commands (TEMP, RATE, STEPS) is received (2 marks).
c) Empty or delete the file whenever a RESET command is received (1 mark).
d) Load the saved records from the file when the server starts (2 marks).
e) Properly check return codes from the file-related functions, handling any errors (1 mark).

AA1.4 Construct a multi-source toolchain for a given device.
The client needs to offer the option of keeping responses in a log file:

a) Using pre-compiler directives, create two versions of the client:
   - **Full**: saves all responses to a log file (2 marks).
   - **Normal**: runs only in memory, no log file (1 mark).
b) Include a correct timestamp for each log file entry (1 mark).
c) Update the makefile to build the two versions (1 mark).
d) Pass the log filename as a command line argument. If this is missing, a default filename should be used. When using the default filename, the user should be informed (2 marks).

KU3.1 Outline specific commands which can be used implement a concurrent application.

You have an option for this task:

Option 3.1A:
Write a technical report (approx. 800 words) covering the following:

- What changes would you make to your server to achieve concurrency? You are not expected to provide the full code but a detailed explanation of what is needed and why would benefit from a few snippets (4 marks).
- What techniques would you use to protect any relevant variables from being modified by different threads at the same time (1 mark).

Option 3.1B:
Update your server code to achieve concurrency, i.e., the ability to handle two clients at the same time. Make sure that you protect any relevant variables from being modified by different threads at the same time (4 marks). Update the makefile to build this modified version of the server (1 mark).

Marking Scheme

Task	Inadequate	Low Quality	Best Quality	Score
Home Part A				
KU 1.1a	No header file	A header file (.h) was used but some of its contents would have fit better in the source files (.c)	Correct use of at least one header file	
		0.5 mark	1 mark	
KU 1.1b	Missing or single file used	N/A	Correct use of multiple files to ensure maintainability and code reuse	
			1 mark	
KU 1.1c	Warnings in both client & server	No warnings in working client (1 mark) No warnings in working server (1 mark)	Both binaries (all tasks implemented) compile with no warnings	
		1 - 2 marks	3 marks	
				KU1.1 Total marks:
AA 4.1a	Missing	N/A	Feature implemented correctly	
			1 mark	
AA 4.1b	Missing	Feature partly implemented	All features implemented correctly	
		1-2 marks	3 marks	
AA 4.1c	Missing	N/A	Feature implemented correctly	
			1 mark	
AA 4.1d	Return values not checked correctly	1 or 2 return values from Sockets API calls not checked correctly	All return values from Sockets API calls checked correctly	
		1 mark	2 marks	
				AA4.1 Total marks:
AA 4.2a	Missing	N/A	Feature implemented correctly	
			1 mark	
AA 4.2b	Missing	Partial answer	All features implemented correctly	
		1-2 marks	3 marks	
AA 4.2c	Missing	N/A	Feature implemented correctly	
			1 mark	
AA 4.2d	Return values not checked correctly	1 or 2 return values from Sockets API calls not checked correctly	All return values from Sockets API calls checked correctly	
		1 mark	2 marks	
				AA4.2 Total marks:
KU 1.3	Missing makefile	Incomplete makefile	A fully implemented makefile	
		1 mark for each correct target	5 marks	
				KU1.3 Total marks:
AA 2.2a	Missing	Incomplete data structure	Complete data structure	
			1 mark	
AA 2.2b	Missing	Record keeping: 2 marks Statistics collection: 1 mark Resetting of data structure: 1 mark	Complete and correct data structure implementation	
		1-3 marks	4 marks	
AA 2.2c	N/A	Valid but incomplete attempt	Proper memory usage and freeing, memory-efficient design	
		1 mark	2 marks	
				AA2.2 Total marks:

Home Part B				
AA 3.2a	No justification	N/A	Complete justification and sample	
			1 mark	
AA 3.2b	Persistence not implemented	Opening & closing file correctly: 1 mark Adding entries correctly following the design correctly: 1 mark	Correct implementation	
		1 mark	2 marks	
AA 3.2c	Reset not implemented	N/A	Resetting of data in file works correctly	
			1 mark	
AA 3.2d	Loading not implemented	Loading implemented but buggy or crashes	Loading of data from file works correctly	
		1 mark	2 marks	
AA 3.2e	N/A	N/A	All returns checked correctly	
			1 mark	
				AA3.2 Total marks:
AA 1.4a	Two versions not implemented	2 marks for full version 1 mark for normal version	Complete and correct implementation	
		1-2 marks	3 marks	
AA 1.4b	No timestamp	N/A	Timestamp successfully written	
			1 mark	
AA 1.4c	Missing	N/A	Correct building/cleaning in makefile	
			1 mark	
AA 1.4d	Cmd. line args. not used	1 mark each for: - reading the file name argument correctly - defaulting to client.log when command line argument is missing (not crashing)	Complete implementation	
		1 mark	2 marks	
				AA1.4 Total marks:
Opt. KU 3.1A	No technical report	Report vague or lacks detail about the work done to achieve concurrency and synchronisation.	Complete and clear report	
		1-2 marks	5 marks	
Opt. KU 3.1B	No implementation	An attempt was done but code does not compile, crashes or does not achieve concurrency and synchronisation.	Implemented all the requirements correctly	
		1-3 marks	5 marks	
				KU3.1 marks for selected option:

Appendix: Protocol example usage

Request (sent by client)	Response (sent by server)	Description of response
RESET	OK	All values cleared
TEMP	38.1	User has a Fever
STEPS	1000	User only walked 1000 steps so far today
RATE	65	User heart rate is low at 65 beats per minute
HELLO	UNKNOWN	Unknown command
STEPS	1234	User only walked 1234 steps so far today
STATS	BATT:65,RATE:2,TEMP:1,STEPS:1	Battery level is 65% and number of calls recorded for each command since last boot/reset are 2, 1 and 1 respectively.
RESET	OK	All values cleared
STEPS	0	Steps have been reset to zero by previous command
STATS	BATT:63,RATE:0,TEMP:0,STEPS:1	Battery level is now 63% and number of calls recorded for each command since last reset.