MCAST
Malta College of Arts, Science & Technology
INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

Low Level Programming II

B.Sc. (Hons) Software Development

Assignment 2 (Home)

Implement a low-level program

Instructions to Students

Read the following instructions carefully before you start the assignment. If needed, ask your lecturer for further explanation.

- This assignment has a total weight of **48%**.
- Copying is strictly prohibited and will be penalised through a referral and other disciplinary procedures as per MCAST Policies, Procedures, and Regulations.
- Make sure your **name, ID card number and group** are on the cover sheet.
- Submit a printed copy of this assignment brief and upload all your files as instructed by your lecturer.
- Each task has marks associated with it as detailed in the marking scheme.
- An individual interview may be held upon assignment submission. You are expected to be able to explain your choices and code in specific tasks to achieve the marks associated with each task.

Scenario

A **port scanner** is an application designed specifically to probe a computer for open ports. When performing a port scan, the application sends client requests to a range of server ports with the goal of finding active ports.

Port scanning is not a malicious process and is mostly used to determine services available on a remote machine. System administrators commonly use it to identify the network services running. Security researchers (and hackers) use it as a first step to identify and exploit vulnerabilities.

A **honeypot** is a computer security tool used to detect attempts at unauthorised access of networked systems. A simple honeypot is a server listening on multiple ports and logging all connections to a file.

Task A: Basic implementation (24 marks)

Your task is to develop a honeypot (server) and a port scanner (client):

KU1 (1.1): You are to arrange your source code appropriately:

a) Use Header files for structures and functions appropriately (1 mark).
b) Use multiple source files to ensure that the code is maintainable (2 marks).
c) No warnings when compiled with `-Wall` compiler switch (2 marks).

AA5 (4.2): Your client and server program should communicate using network sockets.

a) The honeypot should listen on the loopback IP address (127.0.0.1) using the port number passed as a command line argument. It should continue listening after each connection is closed (3 marks).
b) The port scanner should probe a range of ports (passed as command line arguments) on the loopback IP address, keeping track of those ports that accepted the connection (3 marks).
c) Both applications should properly check all the return codes from the Sockets API functions, gracefully handling any errors (1 mark).

KU3 (1.3): Create a makefile with the following specifications (5 marks).

a) **all**: the default target that builds all honeypot and scanner executables.
b) **debug**: a target that builds the debug version of the honeypot and scanner.
c) **release**: a target that builds the release version of the honeypot and scanner.
d) **clean**: a target that cleans up all the compiler generated files.
e) All the targets are in a single makefile. Debug and Release binaries need to be created in separate directories.

AA2 (2.2): You are required to store the ports that accepted connections in a linked list according to the following specifications:

a) Create a data structure for a linked list that holds the ports (1 mark).
b) Implement a linked list to store the port structures (4 marks).
c) When the port scan is finished traverse the linked list and print a summary of the results (2 marks).

Task B: Persistence (14 marks)

While storing the scan result in memory is enough for demonstration purposes, it is required to store it in a log file for auditing purposes.

AA3 (3.2): Upgrade your port scanner to store the data using File I/O:

a) Propose a standard log file format (e.g. JSON, XML, CSV, etc.) that can be used to save the list. Provide a sample in your documentation explaining why you think this format is adequate (2 marks).
b) Persist the open port list to the log file when the port scan finishes (4 marks).
c) Properly check all the return codes from the File I/O API functions, gracefully handling any errors (1 mark).

AA1 (1.4): They honeypot might run on embedded devices that do not have the ability to store files.

a) Using pre-compiler directives, create 2 versions of the honeypot, Full and Lite.
   The Full version saves details about every hit it gets to a log file.
   The Lite version keeps a counter of the total number of connections and on every hit, prints out the counter together with the other details.
   Details must include at least the date, time and port number.
   Remember to update the makefile to include these two versions (5 marks).
b) Pass the log file name as a command line argument to the version that saves to a file. If this is missing, a default filename (hits.log) should be used (2 marks) e.g.:
   `Warning: missing <file> command line argument, defaulting to hits.log`

Task C: Concurrency (10 marks)

SE3. With the honeypot implementation running on a single thread, one needs to have multiple processes running to monitor multiple ports. This is not the best scenario in terms of performance. You need to upgrade the code of your honeypot to have one process monitoring multiple ports.

3.3a: Upgrade your honeypot (both versions) to make it capable of handling multiple ports in one process. You can implement using the **pthread** library (no research needed, 5 marks) or the **select** function (research needed, 6 marks).

3.3b: Write a technical report (not longer than 1 page) explaining how your log file and any other relevant variables are protected from being modified by different threads at the same time. Make sure that you include code snippets in the report. You can also use diagrams if you think they help your explanation (4 marks).

***

Marking Scheme

	Inadequate	Low Quality	Best Quality	Score
		Task A		
1.1a	Missing submission or single file used	A header file (.h) was used but some of the its contents would have fit better in the source files (.c).	Correct use of at least one header file	
	0 Marks	0.5 Mark	1 Mark	
1.1b	Missing submission or single file used	Multiple source files (.c) used but maintainability and code re-use can be improved.	Correct use of source file to ensure maintainability and code reuse.	
	0 Marks	1 Mark	2 Marks	
1.1c	Both honeypot and scanner have at least one warning	One of the binaries compile with no warnings.	Both binaries compile with no warnings.	
	0 Marks	1 Mark	2 Marks	
KU1 (1.1) Total marks				
4.2a	Missing submission	Partial answer	All features implemented correctly.	
	0 Marks	1 Mark each for: - proper use of command line arguments - listening on the correct address - continued listening after client disconnect	3 Mark	
4.2b	Missing submission	Partial answer	All features implemented correctly.	
	0 Marks	1 Mark each for: - proper use of command line arguments - connecting to the correct address - keeping track of active ports	3 Mark	
4.2c	Less half of returns from Sockets API calls checked correctly	At least half of returns from Sockets API calls checked correctly	All returns from Sockets API calls checked correctly	
	0 Marks	0.5 Mark	1 Mark	
AA5 (4.2) Total marks				
KU3 (1.3)	Missing makefile	Incomplete makefile	Makefile implemented fully according to requirements	
	0 Marks	1 Mark for each specification	5 Marks	
2.2a	Missing or incomplete data structure	N/A	Complete data structure	
	0 Marks		1 Mark	
2.2b	Missing linked list implementation	Partial answer	Complete and correct linked list implementation	
	0 Marks	2 Marks each for: - working mechanism to add an entry - correct use of memory management	4 Marks	
2.2c	Missing printing feature	Partial answer	Complete implementation	
		1 Mark each for: - implementing the list printing - handling printing an empty list	2 Marks	
AA2 (2.2) Total marks				

Task B				
3.2a	No design submitted	Partial answer	Complete design, with explanation and sample	
	0 Marks	1 Mark each for: - Design and explanation - Sample following the design correctly	2 Marks	
3.2b	Persistence not implemented	Partial answer	Complete implementation	
	0 Marks	1 Mark each for: - Opening file correctly - Appending to the log file - Closing the file correctly - Log file following the design correctly	4 Marks	
3.2c	Less half of returns from File IO API calls checked correctly	At least half of returns from file IO API calls checked correctly	All returns from file IO API calls checked correctly	
	0 Marks	0.5 Mark	1 Mark	
AA3 (3.2) Total marks				
1.4a	Full/Lite versions not implemented	Partial answer	Complete and correct implementation	
	0 Marks	1 Mark each for using pre-compiler directives to implement the two versions 1 Mark for correct building/cleaning commands in makefile 1 Mark for log entry details 1 Mark for proper log counter mechanism	5 Marks	
1.4b	Command line arguments unused	Partial answer	Complete implementation	
	0 Marks	1 Mark each for: - reading the file name argument correctly - defaulting to hits.log when argument is missing (not crashing)	2 Marks	
AA1 (1.4) Total marks				
Task C				
3.3a	Concurrency not implemented	A decent implementation of concurrency was submitted but it is incomplete, has severe bugs or crashes repeatedly.	A solid attempt that implements all the requirements (possibly with minor bugs) i. using pthread (5 marks) ii. using select (6 marks)	
	0 Marks	2.5 Marks	6 Marks	
3.3b	No protection, no report or an incorrect solution.	An incomplete or buggy protection implementation and report.	Protection that implements all the requirements (possibly with minor bugs) and a solid report.	
	0 Marks	2 Marks	4 Marks	
SE3 (3.3) Total marks				