This coursework relates to all learning outcomes on the module: Have a thorough understanding of the principles and issues involved in systems programming and be able to extend and implement low level functionality. Understand the issues involved in IPC at both local and network levels and be able to design and implement a multi-process communicating system. Conduct a detailed analysis of an existing low-level system and give recommendations to improve resilience and reliance against malicious and accidental interference. Produce solutions to systems level problems in a POSIX environment and be able to give recommendations of possible approaches for the other environments. Task This coursework involves software design and implementation of a piece of software at the systems level with appropriate considerations for design, robustness, and security. Your program must be implemented in C. It is recommended that you write the program under MINIX 3, but you may use any UNIX version provided that you use only POSIX standard calls in your program. Design and implement a data-sharing system for processes in which: Programs can upload and download blocks of data using the OS’s interprocess communication facilities. Clients can share their stored data blocks (to read and/or edit) with other clients. The service stores and maintains the data and provides security. The means by which the data is stored (in memory, on disk, or in combination) is up to you, provided it can be retrieved quickly when needed. The clients for your program would be other programs, not user clients. Therefore, you should develop an API rather than a user interface, which can be offered as a C interface library. You should carefully consider the structure of your API and produce documentation for it. Local clients that share secure data will identify themselves via a shared secret. There is no need to implement “accounts” or related functionality. The service must be implemented in your own code. Delegating the functionality to another service (that is already part of the OS or otherwise) will result in no marks. Marking scheme This exercise is worth 40% of the total marks for the module. Your report (excluding references) has to be less than 3000 words. Submit your report via Moodle by 5pm on the Friday of Week 12. Your report should include your source code as an appendix, which does not count towards the word count. It will be looked at to verify that statements made in your report are true and have been acted on or arrived at through experimentation. Statements about your implementation that are not supported by your code will lose marks. This is individual work and is subject to the University’s rules on plagiarism. This applies to both the report and the source code. Your report should include the following sections. The percentages below are percentages of this coursework, not the whole module (ie, they add up to 100%, not 40%): Introduction and requirements analysis. Starting from the requirements above, add the details of what will be necessary to make the system workable [15%] and secure [15%}. Some questions you might like to think about: How will you enable a client to find out if your service is running and to find out how to connect to it if it is? How will you enable a client to know that the service is actually your service, and not another program pretending to be it in order to steal secure data? How will programs that share data be able to identify the data that is shared with them? If programs allow other programs to write data, how will they know if another program has changed the data since they wrote it? How will you store the shared data? How will your program, with the help of the OS, prevent unauthorised processes reading it from the storage? How will you manage the limited capacity of storage? If you decide to use both memory and disk, how will you decide which? When will you remove shared data? If left forever, RAM and disk space will eventually run out. If a process crashes, how will you know when its data should be removed? What should happen to shared data when the computer is rebooted or switched off? Design analysis and implementation describing how you will implement your answers to the questions in the previous section, including: The architecture of your system (eg, the interprocess communication method used, the storage methods used, any internal threading) [10%] The system design for the server, including its main components, how they are implemented, and any communication between them [15%] The design of the API, describing the features offered and the documentation, and with consideration of the difficulty of porting to/from other shared data APIs [15%] Testing: Test plan for your system, including both intended use and edge cases, including attacks, and results of the testing and your responses. [20%] Research an existing data-sharing service and compare it to yours in terms of functionality, robustness and security. What could you learn from it? How compatible is your service with its? [10%]