CAN201 Introduction to Networking

Networking Project

Contribution to Overall Marks	**40%**
Submission Deadline of Part I	**17 Nov. 2023, 23:59**
Submission Deadline of Part II	**18 Dec. 2023, 23:59**
Type	**Team coursework**
Learning Outcome	**[A] [B] [C] [D]**

How the work should be submitted?

- **SOFT COPY ONLY!**
- Every team leader must submit the work through Learning Mall.

Specification of Part I (20% of overall marks)

File uploading and downloading should be one of the most important network-based applications in our daily life. This part of the networking project aims to use Python Socket programming to implement a client-side application for file uploading and downloading based on a given protocol.

The examiner will define and release the protocol description and the server-side application on Learning mall. However, the released server-side code might have some syntax bugs. You should firstly fix all the bugs and run the server-side code. Then, you should implement the client-side application using Python and test your code using the server-side application.

The details are listed as follows:

Task 1: Server code debug and set up
Fix the existing syntax bugs of the code for the server-side application and run the server-side code. “Server is ready” should be displayed on the terminal window when all the bugs are fixed.

Task 2: Get authorization from the server-side
Your application should login to the server using the rule defined in the protocol. A “Token” will be returned if you login successfully. This token will be used for all the following tasks in this part.

Task 3: Upload a file to the server
For uploading a file, your application should firstly apply for this uploading operation using the required information of the file. An uploading plan will be returned, which includes the “key” for permission, the block size for this uploading and the total block number. Your application should upload the file block by block until the whole file is uploaded. Then, you should check the status of the file on the server according to the protocol. The MD5 of the file will be included in the status, which could be used to check whether your file is received by the server properly.

Submission:

Codes:

- >= Python 3.6;
- The two python program files, i.e., "server.py" and "client.py".

Project Report:

- A cover page with your full names (pinyin for Chinese student; name on your passport for international student) and student IDs of the whole team;
- 3 ~ 5 pages (including everything such as the reference), single column, 1.5x line space, 2.54cm margins, Serif font[1], font size:12pt;
- PDF format, LaTeX is recommended;
- Including:
  - **Abstract**
  - **Introduction**: project task specification (introduce some background, **do not** copy from this document and use your own words), challenge (identify the research/development problems you are going to address), practice relevance (come up with the potential applications with your proposal), contributions (key points that you did for this coursework).
  - **Related Work**: research papers, technical reports, or similar applications that solve or facilitate network traffic redirection.
  - **Design**: the design of you solution including a C/S network architecture diagram (and you need to describe it using your own words), the workflow of your solution (in particular, the steps of performing authorization, fetching token, uploading file), the algorithm (i.e., the kernel pseudo codes of the authorization and file uploading).
  - **Implementation**: the host environment where you develop the implementation, such as the host CPU, Memory, Operating System, etc. Also, the development softwares or tools, like the IDE, the Python libraries, etc. Further, steps of implementation (e.g., program flow charts), programming skills (OOP, Parallel, etc.) you used, and the actual implementation of the authorization function, file uploading function. In addition, the difficulties you met and how did you solve them.
  - **Testing and Results**: testing environment (can be more or less the same with your host implementation environment), testing steps (the steps of using the developed Python programs to complete the project tasks 1-3, including snapshots), and testing results, i.e., the time used for uploading the whole file, and you should apply figures of bars or curves for showing average performance.
  - **Conclusion**: what you did for this project and any future work for improvement.
  - **Acknowledgement**: individual contribution percentage should be clarified here if the project is a teamwork by using this format: Student1's name (ID) contributes XX% to the project, Student2's name (ID) contributes XX% to the project, Student3's name (ID) contributes XX% to the project, Student4's name (ID) contributes XX% to the project and Student5's name (ID) contributes XX% to the project. If there is no clarification of individual contribution, it is considered that all the individual team contributes the same percentage to the project.
  - **Reference** [IEEE format]

[1] Eg. Times New Roman, Modern No. 20 or Cambria.

Meanwhile, you have to follow the compulsory requirement (no tolerance[2]):

- Only ZIP file is allowed to submit;
- The ZIP file should be named as: CAN201-CW-Part-I-Student1name-Student2name-Student3name-Student4name-Student5name.
- The ZIP file includes two folders, i.e., "Codes" and "Report". The Codes folder includes all the Python files, and the Report folder includes the report file;
- Python files are: server.py, client.py;
- The report file should be named as: Report_Part_I.pdf;

Allowed Python modules:

os, sys, shutil, socket, struct, hashlib, math, tqdm, numpy, threading, multiprocessing, gzip, zlib, zipfile, time.

Marking Criteria

The following marking scheme is for the team, and every team member shall contribute to the project. Also, several specific rules should be followed:

1. Every team should use the "ACKNOWLEDGMENT" section of the IEEE template to describe the individual contribution(s) using the following format: Student1's name (ID) contributes XX% to the project, Student2's name (ID) contributes XX% to the project, Student3's name (ID) contributes XX% to the project, Student4's name (ID) contributes XX% to the project and Student5's name (ID) contributes XX% to the project.
2. If there is no clarification about the individual contributions, it is considered that every team member in the same team has the same contribution percentage and will have the same mark of the CW project.
3. The individual contribution must be in a range: for a 5-person team, it must be 10% - 30% (15% and 30% are included); for a 4-person team, it must be 15% - 35% (15% and 35% are included). If any individual contribution percentage of a team is out of the range (e.g., a 5-person team has the contributions like: 60%, 10%, 10%, 10%, 10%), the team may go through a review by the module leader about the contribution discrepancy.
4. The algorithm for calculating individual mark as follows:
    a. Assuming the 3-person team's mark is *m*, student1 contributes *x*%, student2 contributes *y*% and student3 contributes *z*%, student4 contributes *u*%, student5 contributes *v*%.
    b. The student who gets the most contribution will get mark *m*.
    c. Student 1's mart will be $x/max(x,y,z,u,v)*m$.
    d. Student 2's mart will be $y/max(x,y,z,u,v)*m$.
    e. Student 3's mart will be $z/max(x,y,z,u,v)*m$.
    f. Student 4's mart will be $u/max(x,y,z,u,v)*m$.
    g. Student 5's mart will be $v/max(x,y,z,u,v)*m$.

Report (50%)

Marking Criteria	Item	Mark
Contents (40%)	Abstract	3%

[2] It means that if you do not follow the compulsory requirement, your work will be marked as zero.

	Introduction	5%
	Related Work	4%
	Design	8%
	Implementation	7%
	Testing and Results	7%
	Conclusion	3%
	Reference	3%
Typography (5%)	Report structure, style, and format	5%
Writing (5%)	Language	5%

Marking Scheme:

1. Contents (40%)
   1.1. Abstract (3%)
   - Good (3%)
   - Appropriate (1-2%)
   - No abstract (0%)

   1.2. Introduction (5%)
   - Excellent (5%)
   - Lack of necessary parts (1%-4%)
   - No introduction (0%)

   1.3. Related Work (4%)
   - Sufficient (4%)
   - Not enough (1%-3%)
   - No introduction (0%)

   1.4. Design (8%)
   - Excellent: adequate and accurate figures and text description (8%)
   - Reasonable: clear figures and text description (4%-7%)
   - Incomplete: unclear figures and text description (1%-3%)
   - No design (0%)

   1.5. Implementation (7%)
   - Excellent: sufficient details of implementation (7%)
   - Reasonable: clear description of implementation (4%-6%)
   - Incomplete: unclear description of implementation (1%-3%)
   - No implementation (0%)

   1.6. Testing and Results (7%)
   - Excellent: sufficient testing description, correct experimental results using figures with clear text description and analysis (7%)
   - Acceptable: clear testing description, appropriate experimental results using figures with acceptable text description and analysis (3%-6%)
   - Incomplete: lack of testing description, experimental results with figures, or text description and analysis (1%-2%)
   - No testing and results (0%)

   1.7. Conclusion (3%)
   - Excellent conclusion (3%)
   - Acceptable conclusion (1%-2%)

- No conclusion (0%)

1.8. Reference (3%)
- Excellent reference with the correct IEEE format (3%)
- Incorrect or inconsistent reference format (1%-2%)
- No reference (0%)

2. Typography (5%)
- Beautiful and clear typography: 5%
- Acceptable typography: 2%-4%
- Bad typography: 0% ~ 1%

3. Writing (5%)
- Accurate and concise language: 3%-5%
- Unclear and confusing language: 1% ~ 2%

Codes (50%)

Program testing steps:

1．Debug for server code:

Run server.py and "Server is ready" should be displayed on the terminal window.

2．Authorization:

Run client.py -server_ip xxx.xxx.xxx -id xxxxxx -f <path to a file> and f"Token: {token}" should be displayed on the terminal window.

3．File uploading:

After step 2, your code should upload the file which is indicated in the command line (less than 1MB) to the server. The progress should be printed on the terminal window.

Marking Scheme:

Step 1 (15%)
- No bugs and run properly (15%)
- Fix each bug, get 2% (2~12%)
- No bug is fixed (0%)

Step 2 (15%)
- Print the right "Token" (15%)
- Print something but not the right "Token" (6~14%, no future test for step 3)
- Print nothing but can run (1~5%, no future test for step 3)
- Cannot run (0%, no future test for step 3)

Step 3 (20%)
- The server received the right file (checked by MD5) (20%)
- The server received a file but not the right file (10% ~ 19%)
- The server received nothing (0% ~ 9%)