Computer Network
Project 2
Simple TCP-based Message Queue Application
(Difficulty ★★★☆☆)

CSI4106-01

Fall, 2025

Prelim.

Before you do this homework, you must be fully aware of **“Project Policy Notice”**

Goal (you are expected to)

1. Learn a basic socket programming
   - IPv4/TCP only
   - Sockets in Linux Environment
     - **Do not write a code in Windows.**
     - **The two OSs have different socket APIs.**
     - **That means, they are not compatible.**
2. Understand the “Client-Server” architecture
3. How Message Queue Works
4. **Write a simple TCP-based Message Queue application.**

What are these functions?

- **socket**(socket.AF_INET, socket.SOCK_STREAM)
- **setsockopt**(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
- **bind**(("0.0.0.0", 7777))
- **listen**(5)
- **connect**((host, port))
- **accept**()
- **close**()

In case of python

What is Message Queue?

- A Message Queue is an implementation of Message-Oriented Middleware(MOM).

- Our MQ application consists of 3 components.
- 1. Server (MQ) : stores and sends data.
- 2. Producer : produce message (or data) and send it to the server.
- 3. Consumer : read data from the server.

Objectives

- You must write your own code **`server.[py/c]`**
- You must write your own code **`producer.[py/c]`**
- You must write your own code **`consumer.[py/c]`**

- The three programs must take the two or three parameters
  - Server: IP address, port number(producer) and port number(consumer)
  - Producer/Consumer: IP address and port number
  - E.g) `python server.py 127.0.0.1 5000 5001`
  - E.g) `python producer.py 127.0.0.1 5000 task.txt`
  - E.g) `python consumer.py 127.0.0.1 5001`
  - E.g) `./server 127.0.0.1 5000 5001`

How the project works

- `Your project file` should start as following order
  - *Server -> Consumer(or consumers) -> producer*

- *Server* should have a queue that stores messages from *Producer*
- *Producer* gets input from task.txt
  - Each line of the txt is considered as tasks.
- *Consumer* pulls the task from *Server* every second.
  - The tasks are deleted from server after pull, since it is stored in priority queue.
  - Priority queue can be made of standard library(In case of python, library ‘heapq’), or make it by yourself (for example, a custom queue made by using ‘Array’ or 'List')

producer.py
(note this is incomplete)

```
import socket, time, sys
…
host = sys.argv[1]
port = int(sys.argv[2])
task_file = sys.argv[3]
sc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sc.connect((host, port))
tasks = …

for timestamp, priority, task_id, duration in tasks:
    sc.send(f’CREATE {priority} {task_id} {duration}\n’.encode())
…
```

consumer.py
(note this is incomplete)

```
import socket, time, sys
…
host = sys.argv[1]
port = int(sys.argv[2])
worker_name = sys.argv[3]
sc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sc.connect((host, port))
consumer_id = sc.recv(1024).decode().strip()
while True:
  sc.send(b‘REQUEST\n’)
  response = sc.recv(1024).decode().strip()
  …
```

server.py (note this is incomplete)

```
import socket, sys
from threading import Thread
…
def setup_sockets():
…
def producer_worker():
…
def consumer_worker(…):
…
def shutdown():
  is_running = False
…
if __name__ == '__main__':
  setup_sockets()
  producer_thread = Thread(target = producer_worker)
  producer_thread.daemon = True
  producer_thread.start()
  worker_thread = Thread(target = consumer_worker, …)
  worker_thread.daemon = True
  worker_thread.start()
  …
```

```
if __name__ == '__main__':
  …
  try:
     while is_running:
        time.sleep(1)
     except KeyboardInterrupt:
     shutdown()
```

Guidelines – Producer

```
[root@localhost p1]# python3 producer.py 127.0.0.1 8888
test_tasks.txt
[CREATE] 3 task1 3.0
[CREATE] 3 task2 3.0
[CREATE] 3 task3 3.0
[CREATE] 3 task4 3.0
[CREATE] 1 task5 3.0
[CREATE] 2 task6 3.0
[CREATE] 2 task7 3.0
[CREATE] 1 task8 3.0
exit
[root@localhost p1]#
```

test_tasks.txt

```
0.0 3 task1 3.0
0.0 3 task2 3.0
0.0 3 task3 3.0
0.0 3 task4 3.0
2.0 1 task5 3.0
4.0 2 task6 3.0
4.0 2 task7 3.0
6.0 1 task8 3.0
```

Guidelines – tasks.txt

tasks.txt is written with multiple lines of below:

```
timestamp priority task_id duration
timestamp priority task_id duration
timestamp priority task_id duration
timestamp priority task_id duration
```

Timestamp: task created time after the program starts (seconds, actual number)

Priority: 1(HIGH), 2(MEDIUM), 3(LOW)

Task_id: task identifier (string)

Duration: task processing time(seconds, actual number)

Example of tasks.txt

```
0.0 3 task1 3.0
0.0 3 task2 3.0
0.0 3 task3 3.0
0.0 3 task4 3.0
2.0 1 task5 3.0
4.0 2 task6 3.0
4.0 2 task7 3.0
6.0 1 task8 3.0
```

How this works

0 seconds : task 1 2 3 4 is created and sent to server
2 seconds : task 5 is created and sent
4 seconds : task 6 7 is created and sent
6 seconds : task 8 is created and sent, producer is closed.

Guidelines – Server

```
[root@localhost p1]# python3 server.py 127.0.0.1 8888 8889
[Consumer1 connected]
[1 consumers online]
[CREATE] 3 task1 3.0
[CREATE] 3 task2 3.0
[CREATE] 3 task3 3.0
[CREATE] 3 task4 3.0
[ASSIGN] task1 - consumer1
[CREATE] 1 task5 3.0
[ASSIGN] task5 - consumer1
...
[Consumer 1 disconnected]
[0 consumers online]
^C
exit
[root@localhost p1]#
```

Producer created Events and sent here

Guidelines – Consumer

```
[root@localhost p1]# python3 consumer.py 127.0.0.1 8889
Consumer1 connected
[REQUEST]
[COMPLETE] task1
[REQUEST]
[COMPLETE] task5
[REQUEST]
[COMPLETE] task6
[REQUEST]
[COMPLETE] task7
…
[COMPLETE] task4
[REQUEST]
NOTASK
[REQUEST]
NOTASK
…
^C
exit
[root@localhost p1]#
```

Task5 was completed first because it had a higher priority than task 2

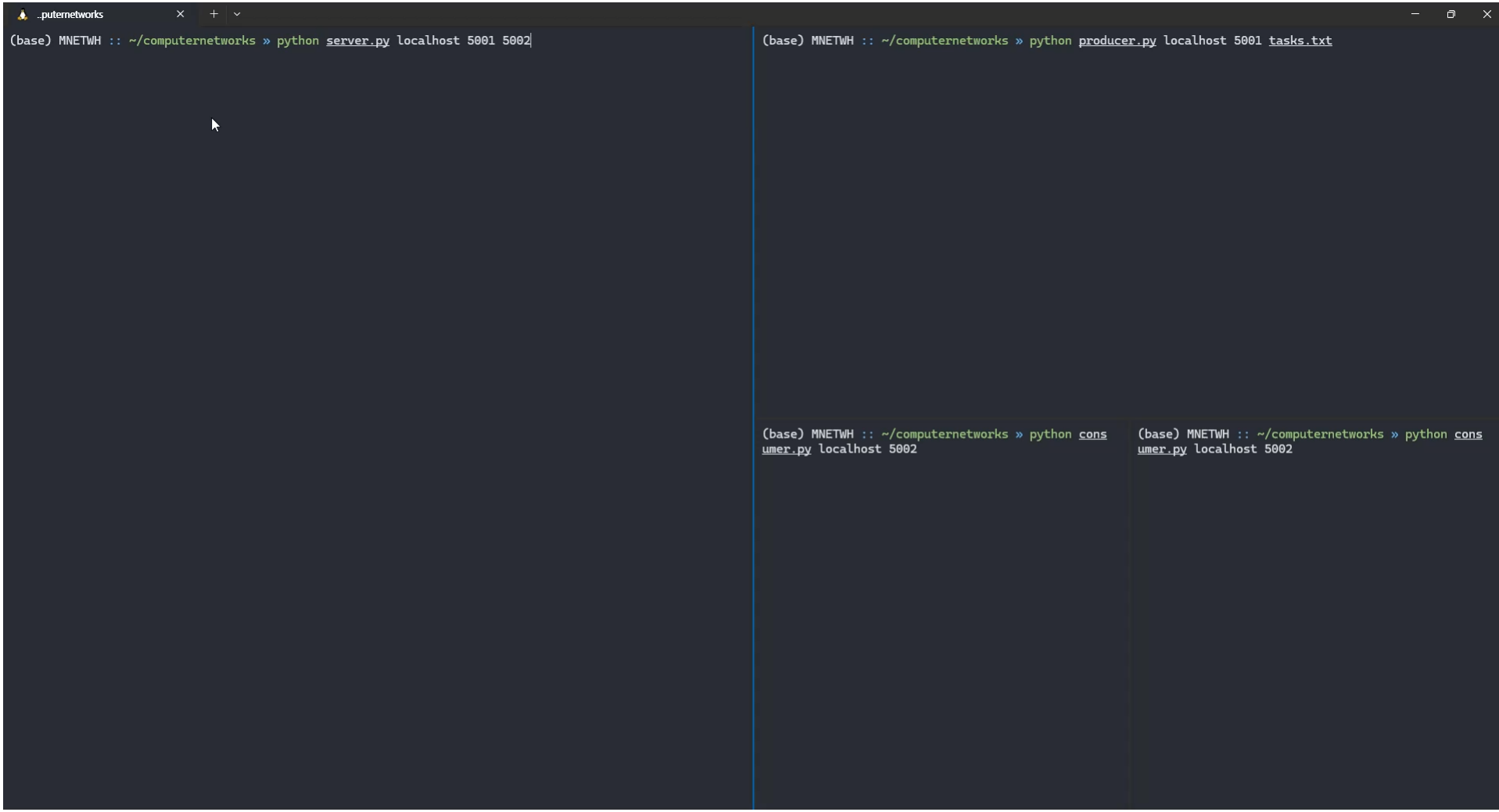
If there is no task, keep requesting after 1 second.

Guidelines – 2 Consumers

```
[root@localhost p1]# python3
consumer.py 127.0.0.1 8889
Consumer1 connected
[REQUEST]
[COMPLETE] task1
[REQUEST]
[COMPLETE] task5
[REQUEST]
[COMPLETE] task8
[REQUEST]
[COMPLETE] task7
[REQUEST]
NOTASK
[REQUEST]
NOTASK
```

```
[root@localhost p1]# python3
consumer.py 127.0.0.1 8889
Consumer2 connected
[REQUEST]
[COMPLETE] task2
[REQUEST]
[COMPLETE] task3
[REQUEST]
[COMPLETE] task6
[REQUEST]
[COMPLETE] task4
[REQUEST]
NOTASK
[REQUEST]
NOTASK
```

Guidelines – Example video



We will test your code as follows.

- OS: Ubuntu Linux
- Language: Python3 / C language
- Your codes can
  - Run with **custom IP and Port** (10pts)
  - Work perfectly during evaluation **with multiple consumer** (30pts)
  - Be **terminated** by Ctrl+C (5pts)
  - The programs will be terminated by following order : consumer.py -> server.py
  - **Close the sockets** (We will check this by **`netstat`**) (5pts)

Your report must include

- **Introduction/Reference** (3pts)
  - Software environment, programming language you used, version, reference and so on.
- **Flow chart or Diagram** (10pts)
  - Must show the logic of your program
  - Focus on describing how your client and server work.
- **At least 3 snapshots** which prove your codes are working well. (5pts)

Your report must include (cont'd)

- **Logical explanations block by block in detail**. (20pts)
  - It is different with brief comments in your source code!!!
  - In your report, write what the blocks do and why you implemented those functions.
- **All explanations of the 7 functions** in "What are these functions?" slide. (12pts)

You will get 0 points if you…

- Copy your friend's codes
  - + Change a little bit of them.
  - + Wish that TAs don't catch that.

- Use a 3rd-party API or codes.

Helpful Keywords

- Socket
- Thread / Multithread
- Message Queue
- Producer-Consumer Pattern
- Python standard libraries

Deliverable

- **Only one zip file of "YourID_2.zip"**
  - **If your ID is 2022147123,**
  - **'2022147123_2.zip' should be your deliverable file name.**
- **In the zip file only the four files must be included without any folder**
  - **report.pdf**
  - **server.py or server.c**
  - **consumer.py or consumer.c**
  - **producer.py or producer.c**
    - **if you use C language, include compile.sh as well**
    - **Files with different format/name will be a deduction [2pts each]**

- **DUE DATE**

  13/Nov/2024 23:59:59 KST

  No exception for exceeding deadline

- **Delay Policy**

  -33%pts for ~14/Nov 23:59:59

  -66%pts for ~15/Nov 23:59:59

  -100%pts for 16/Nov 00:00:00~

Score Policy: *Max. 100pts*

1	Not submitted / not working / missing files	**0 pts**
2	Overdue ➔ Delay	**-33% pts/day**
3	**The rules or directions whose scores are not specified/ are not followed**	-10 pts/rule
4	**Any 3rd party framework is used**	**0 pts**
5	**Plagiarizing / Over-implementation (Any kinds of Suspicion of Code-copy)**	**0 pts**
6	**Impolite Report / Lack of Comments**	**0 pts / -50% pts**

FAQ

Q. Can I use Korean for the report?
-> Yes, you may use either English or Korean.

Q. Can I use C++?
-> Yes, you can use C++ with the C++ standard library, including STL. However, for socket programming, you must use <sys/socket.h>. Additionally, please specify your GCC version and C++ standard (e.g., C++17 or ISO/IEC 14882:2017) in your report.

Q. What environment should I use for testing?
-> We will test your code in Ubuntu 22.04, python 3.10, gcc 9.4. Please specify your testing environment in the report. If the code doesn't work well in our environment, we will try to match our setup to yours (but, you should use Linux socket API)

Q. I see sockets with "TIME_WAIT" status, after closing the program.
-> 'TIME_WAIT' status on client sockets is not an issue.

If your program has issues, please note them in the report.

We will create a Q&A board on LearnUS. If you have any questions, please post them there.