Assignment 4 - ProcChat

ProcChat

Due date 23:59 Tuesday 18 May 2021 (local Sydney time)

ProcChat

You are to develop a localised chat server that will support a number of clients through named pipes (FIFOs). The server and client applications will communicate through a fixed sized binary protocol. The server will manage a global named pipe for establishing connections, after a connection has been established it will construct a separate read and write named pipes for the client to utilise. The client will communicate to the server over the named pipes. The server must be able to read from all clients asynchronously.

Global Process

The global server is involved in facilitating the initial connection from clients. The initial message from the client will be an identifier prefix for the named pipes to be constructed. The client will read and write to the respective pipes that are post fixed with `_RD` and `_WR`.

This process will be responsible for creating the separate client handler daemons responding to each client as a separate process. These processes will need to listen to all messages sent from the client and write to back to the client and any other client's named pipe to relay messages from client to others.

The global event pipe goes by the name `gevent`, this pipe should only be read by the global process but can be written to by any other process.

Client-Handler

A client-handler is spawned when a client is connected, a client-handler will be relegated to a specific domain. A domain is simply a folder relative to the current working directory.

The Protocol

The system uses a binary message protocol to facilitate communication between the server, client-

handler and client. The server and client interaction is simply for acknowledgement and setting up the named pipes. The main protocol will be used between the client-handler and the client. The client-handler is spawned by the global server process to handle a client and will communicate with the client.

The binary message is broken up two parts, the first part being the type of message and the second being contents related to the type. The type will determine how to interpret the contents in the second part. Each message is 2048 bytes.

Under each message type in the following sections, the binary representation will be presented. The type is at the start of the message and is 2 bytes in size.

Anatomy of a message:

```
+-------------+--------------------+
|type: 2 bytes| contents: 2046 bytes|
+-------------+--------------------+
```

GlobalProcess-Client Protocol

- CONNECT `<identifier>` `<domain>`

Type Decimal: 0

Type Binary: 00000000 00000000

The client will connect to the server, the server will construct two named pipes the identifier. The client is expected to connect to the named pipes after sending a well formed connect message to the server.

The identifier part of the message is maximum 256 characters ASCII encoded. The domain component will map to a folder relative to the current working directory.

ClientHandler-Client Protocol

- SAY `<message>`

Type Decimal: 1

Type Binary: 00000000 00000001

Packet Layout:

Type: 2bytes

Message: 1790bytes, ASCII characters

The client will send the SAY command to the client-handler, the client-handler will need to relay this to all other clients that are connected to same domain using the RECEIVE message. The contents

maximum is 1790 characters.

- SAYCONT `<message>` `<termination>`

Type Decimal: 2

Type Binary: 00000000 00000010

Packet Layout:

Type: 2bytes

Message: 1789bytes, ASCII characters

Termination: 1byte, 255 indicates termination

The client will send the SAYCONT command to the client-handler, this is a variation on SAY in which the server will relay the message as a RECVCONT. SAY command typically sends one message within 2046 bytes of its message contents. SAYCONT reserves the 2046th byte as a termination byte for the message's contents, when the termination byte represents the value 255, the message is considered complete. The maximum message is 1789 characters.

The buffering of the SAYCONT messages will occur on client side. Note this detail when testing your code.

- RECEIVE `<identifier>` `<message>`

Type Decimal: 3

Type Binary: 00000000 00000011

Packet Layout:

Type: 2bytes

Identifier: 256bytes

Message: 1790bytes

This message is sent from the client-handler to all other clients (excludes sender). When a client has sent a `SAY` message to the client-handler, the client-handler relays the message along with the identifier to all other clients in the domain. The contents (2046 bytes) where the first 256 bytes are reserved for the identifier and the 1790 bytes afterwards is the message.

- RECVCONT `<identifier>` `<message>` `<termination>`

Type Decimal: 4

Type Binary: 00000000 00000100

Packet Layout:

Type: 2bytes

Identifier: 256bytes

Message: 1789bytes Termination: 1byte

Similar to SAYCONT, the client handler will be sending the message in chunks to the client. The message will contain an identifier (who sent it) and send termination byte.

The buffering of the RECVCONT messages will occur on client side. Note this detail when testing.

- DISCONNECT

Type Decimal: 7

Type Binary: 00000000 00000111

Packet Layout:

Type: 2bytes

This message is sent from the client to the client-handler. This will tell the client-handler that the FIFOs can be removed and the handler can be terminated.

Extension

- PING

Type Decimal: 5

Type Binary: 00000000 00000101

Packet Layout:

Type: 2bytes

The PING message is sent from the client-handler to the client. This is to check that the client is still alive in the event the process crashed. This extension requires some form of asynchronous IO to be implemented. The PING message is sent every 15 seconds from the server.

- PONG

Type Decimal: 6

Type Binary: 00000000 00000110

Packet Layout:

Type: 2bytes

The PONG message is sent from the client to the client-handler. This is the response to the PING message being sent from the server to check if it is still alive.

Daemon Alive

Each client-handler will send a signal to the global process under `SIGUSR1` to indicate that it is shutting down. The global process can use this information, to clean up the process before the child becomes a zombie process.

As an extension, you are required to implement PING and PONG messages that will check that the client is still alive. Every 15 seconds, the client-handler will send a PING message to a client, requiring the client to respond to the message. with PONG.

Restrictions

You are prohibited from using threads within this assignment. Communication between users in the chat room must be facilitated via the client handler. The client handler will be a separate process from the global process.

- No VLAs
- No Excessive CPU usage
- No Zombie Processes (clean up your processes when you can)

If your program does not adhere to these restrictions, your submission will receive 0.

Make sure you thoroughly test your program and construct a mini-client program to interact with your server.

Marking Criteria

The following is the marking break, each point contributes a portion to the total 15% of the assignment. You will receive a result of zero if your program fails to compile.

- 5% Test Cases - Your Program must pass public, private and hidden test cases, to achieve the maximum number of points awarded for this section.
- 6% Solution Discussion - You will need to answer questions from a COMP2017 teaching staff member regarding your implementation. You will be required to attend zoom session with COMP2017 teaching staff member after the code submission deadline. A reasonable attempt will need to be made, otherwise you will receive zero for the assessment. Failure to attend will result in zero for the assessment. In this session, you will be asked to explain:
    - How your solution communicates between the client-handler and client?
    - How are you handling erroneous messages?
    - How is your solution handling client failure?
    - Answer further questions regarding your test cases, code style and knowledge.
- 2% Extension - Your client-handler must be able to detect when a client is no longer alive. This

must be a clear use of non-blocking IO and multiplexing.

- 2% Test Cases - Your solution must include a suite of test cases that should be automated and executable with the given make script. Please make sure your test suite outputs the result of each test case in a human readable format.

Academic Declaration

By submitting this assignment, you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.