Chat Server

Chat Server allows multiple clients to exchange text messages using their Unix shell terminals. It also allows to broadcast the messages to all online users e.g. in case of any emergency. It supports the following set of functionalities:

(1) Create new user
(2) Login and logout
(3) Show all users and Show all online users
(4) Send/Receive Chat Requests to online users
(5) Exchange text messages (chatting)
(6) Stop an ongoing chat session
(7) Broadcast Message (Send messages to all users)
(8) Stop an existing broadcast
(9) Get/Set User Profile Information (additional functionality supported)

High-level Design

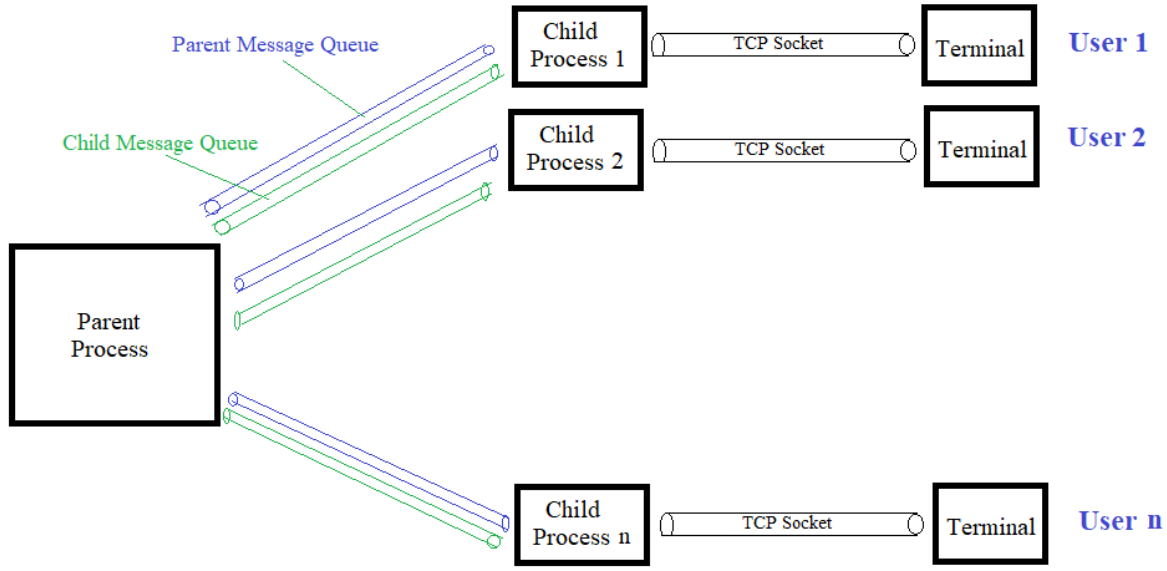

Figure 1. High-level Design of Chat Server

The high-level design of Chat Server can be seen in Figure 1 above. The Server consists of a parent process and several child processes. Each child process is supposed to manage one client user session. The child process communicates with the client user terminal using a **connected TCP Socket**. The inter-process communication (IPC) between parent process and child process is via a pair of **message queues**.

So for each client user session, a new child process is created to manage the same and for each child process a pair of message queues is created for communication with the parent process. The two message queues are called:

(1) **Parent Message Queue**: The parent process reads from this message queue and the child process writes into it. It is shown in blue colour in the above diagram.
(2) **Child Message Queue**: The child process reads from this message queue and the parent process writes into it. It is shown in the green colour in the above diagram.

Low-level Design

The Chat Sever parent process maintains two separate linked lists to store the information about the users of Chat Server:

(1) **masterUserList**: This linked list stores information about all the users created in the Chat Server. It does not matter whether the user is currently online or not. As soon as a new user is created in the Chat Server, its information is stored in the masterUserList.

(2) **connUserList**: This linked list stores information about all the online users in the Chat Server. As soon as a user logs into the Chat Server, its information is stored in the connUserList. Similarly when the user logs out of the Chat Server, this information is removed from the connUserList.

The data structures for masterUserList and connUserList are shown below:

```
typedef struct
{
    char username[MAX_USERNAME_LEN];
    char password[MAX_PASSWORD_LEN];
    int  age;
    char gender;    // Use 'M' or 'F'
    char description[MAX_DESCRIPTION_LEN];
} User;

typedef struct UserNode
{
    User user;
    int pid;
    struct UserNode *next;
} UserNode;

// masterUserList contains a list of all the chat users in the system.
// parent process at chat server will keep an active record of it all
// the time.
UserNode *masterUserList = NULL;

// connUserList contains a list of users which are currently online.
// parent process at chat server will keep an active record of it all
// the time.
UserNode *connUserList = NULL;
```

The parent process keeps a record of all the parent message queue ids and child message queue ids in two C arrays. These are defined globally as follows:

```
// Create container for parent message queue ids.
int parentmsgqids[MAX_CONNECTIONS];

// Create container for child message queue ids.
int childmsgqids[MAX_CONNECTIONS];
```

In the parent process, the main() function simply performs the following tasks:

(1) Create a TCP socket using socket() call. Make it as non-blocking by setting appropriate flags in fcntl() call.

(2) Run an infinite loop to check for incoming connections using accept() system call. This is non-blocking. There can be two cases:

   (a) If there is an incoming connection request, accept() will return a connected socket i.e. "connfd" in which case, code will create parent message queue and child message queue. Further, it will call fork() to create a child process and handle the client session by calling handleClient() function.
   (b) If there is no incoming connection request, accept() will return -1. In that case, we simply check for any messages in all the parent message queues by calling handleMessageFromChild() function in a while loop.

Some of the important functions used by the parent process are as follows:

(1) **handleMessageFromChild()**: It checks for any incoming message in the parent message queue. If there is a valid message there, it checks the type of message in a switch..case and takes necessary action accordingly.
(2) **createNewUser()**: It creates a new user if it already doesn't exists and adds it to the masterUserList.
(3) **deleteUser()**: If the user exists, it removes the user from masterUserList.
(4) **login()**: If user is not online i.e. not already in connUserLsit, this function will add this user to connUserList.
(5) **logout()**: If user is online i.e. present in connUserList, this function will remove this user from connUserList.

Some of the important functions used by the child process are as follows:

(1) **handleClient()**: It is used by the child process to manage a connected client user session. It further calls two functions handleInitialSteps() and handleConnectedUserSteps() to do this.
(2) **handleInitialSteps()**: It will show initial options to the user such as create new user, login existing user and terminate the session.
(3) **handleConnectedUserSteps()**: It will show the options to a logged-in user such as show all users, show all online users, start chat, broadcast message, show user profile info and logout.
(4) **handleChatting()**: It is used by the child process to exchange chat messages with the other user session.