Young-ki Hwang
youngki@top.cis.syr.edu
April 30. 1996
project

Contents

1 Project Description

1.1 Title

Multi-user chat in the internet.

1.2 Description

You can use the *TALK* program and talk to one person at a time, or you can use *YTALK* and talk to one or more people at the same time. What these programs have in common is that your conversation are private , like a telephone call. You control who you talk to and only those people can take part in the conversation.
But a vast number of people can gather in one place and talk to every other people in the *Multi-User Chat* environment. Whatever you say can be heard by everyone else in your group. You can hear what everyone says.

2 Class Hierachy

Figure 1 shows the class hierachy which is used in the project.

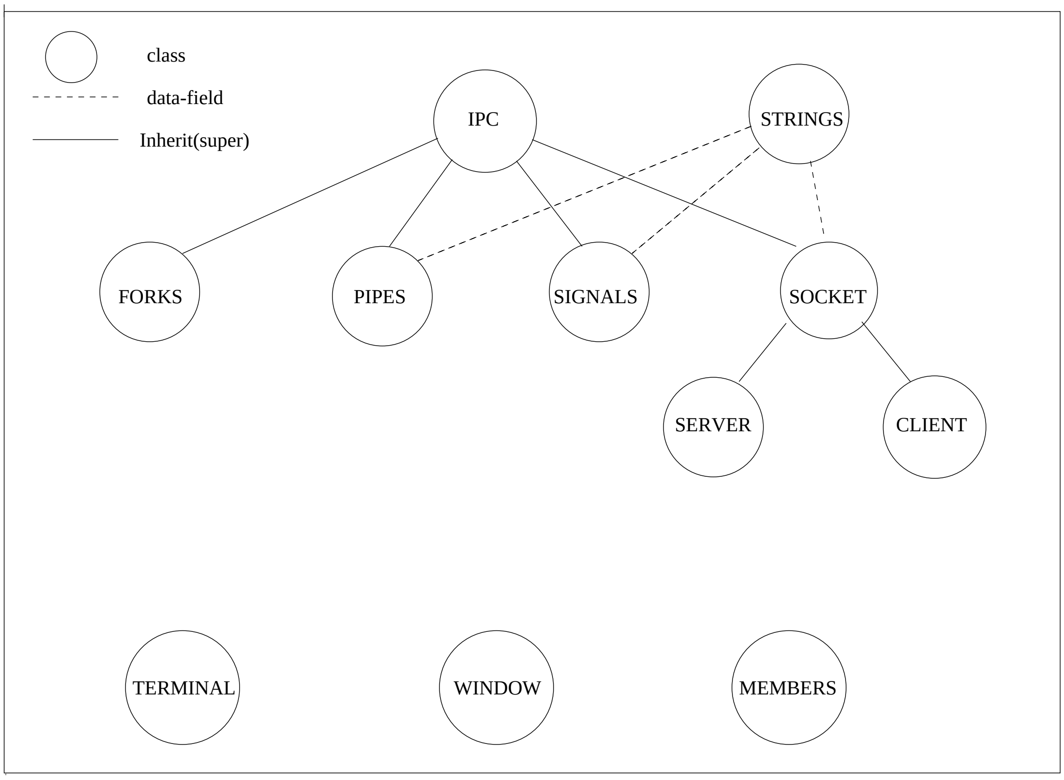

Figure 1: class hierachy

2.1 IPC class

IPC class is an abstract class that defines communication between processes. Socket, pipes, signals, forks classes inherit the IPC class.

2.2 SOCKET class

Socket is the API(Application Program Interface) to the communication protocol. Socket class defines methods for communication between networks.

1. **private data**: socketfd.
2. **receive(char *buffer)** : Receive data into "buffer" from socketfd(private data).
3. **receive(int sockfd, char *buffer)** : Receive data into "buffer" from "sockfd".

4. **send(char *buffer)** : Send “buffer” into the socketfd.

5. **send(int sockfd, char *buffer)** : Send “buffer” into the “sockfd”.

6. **get_sockfd()** : Return the socketfd.

7. **close_sockfd()** : Close the socketfd.

2.3 PIPES class

Pipe is the communication channel between parent and child processes. Pipe class defines methods for communication between parent and child processes.

1. **private data**: p[2]. p[0] is for read and p[1] is for write data.

2. **receive(char *msg, char *id)** : Receive “msg” and “id” from p[0](private data).

3. **receive(int rp, char *msg, char *id)** : Receive “msg” and “id” from “rp”.

4. **send(char *msg, char *id)** : Send “msg” and “id” into p[1](private data).

5. **send(int wp,char *msg, char *id)** : Send “msg” and “id” into “wp”.

6. **send(char *msg)** : Send “msg” into p[1](private data).

7. **send(int wp ,char *msg)** : Send “msg” into “wp”.

8. **get_readpipe()** : Return the p[0].

9. **get_writepipe()** : Return the p[1].

10. **close_readpipe()** : Close the p[0].

11. **close_writepipe()** : Close the p[1].

2.4 SIGNALS class

Signal is a event notification between processes. Signals class defines methods for communication between processes.

1. **private data**:

2. **receive(int sig, void((*func)()))** : When Receiving “sig”, run the (*func).

3. **send(int pid, int sig)** : Send “sig” to process which is “pid”.

2.5 FORKS class

Fork is only way to make a new process. The fork creates a copy of the process that was executing. The process that exectued the fork is called parent process and the new process is called the child process. Forks class defines how to make a new process.

1. **private data**:
2. **make_child()** : Make a new child process.
3. **wait_child_termination()** : Parent process waits until child terminate.

2.6 SERVER class

Server class inherits the socket class. Server class defines the methods which should be done in the server part.

1. **private data**: newsockfd, server_address.
2. **initialize()** : Server register its well-known address with the system. Server port #, machine address(server_address) and network protocol type are initialized.
3. **ready_connection()** : Make a server to be ready for connection from the client.
4. **make_connection()** : Accept a connection from the client.
5. **get_newsodkfd()** : Return the newsockfd(private data).
6. **close_newsodkfd()** : Close the newsockfd(private data).

2.7 CLIENT class

Client class inherits the socket class. Client class defines the methods which should be done in the Client part.

1. **private data**: server_address.
2. **initialize()** : Client set up the information about the server. Server port #, server machine address(server_address) and network protocol type are initialized.
3. **make_connection()** : Connect to the server.
4. **get_id()** : Get a id of the client.

2.8 MEMBERS class

Members class maintain the all the members who are joining currently.

1. **private data**: table[].
2. **insert(char *id)** : Insert a member who is "id" into the table.
3. **remove(char *id)** : Remove a member who is "id" from the table.
4. **Get_numofmember()** : Return the total # of members joinging currently.
5. **Get_sockfd(char *id)** : Return the sockfd which "id" is using.
6. **Get_sockfd(int index)** : Return the sockfd which is located in the "index"th position in the table.
7. **Get_readpipe(int index)** : Return the p[0] which is located in the "index"th position in the table.
8. **Get_writepipe(int index)** : Return the p[1] which is located in the "index"th position in the table.
9. **check_exist(char *id)** : Check whether a member who is "id" is already exist.

2.9 STRINGS class

String class defines methods about manipulating the strings.

1. **private data**:
2. **copy(char *str1, char *str2)** : Copy "str2" to "str1".
3. **compare(char *str1, char *str2)** : Compare "str2" with "str1".
4. **scan(char *buf, char *str)** : Get a first string in the "buf" into "str".
5. **leftshift(char *str, int count)** : Shift a string to left for "count" times.
6. **concatenate(char *str1, char *str2)** : Concatenate "str2" to "str1".

2.10 TERMINAL class

Terminal class handles the terminal type data, set up the terminal.

1. **makeecho()** : Allow window to echo.
2. **makenoecho()** : Allow window to noecho.
3. **makenl()** : Make window to be nl mode.
4. **makenonl()** : Make window to be nonl mode.
5. **makecbreak()** : Make window to be cbreak mode.
6. **makenocbreak()** : Make window to be nocbreak mode.
7. **makeraw()** : Make window to be raw mode.
8. **makenoraw()** : Make window to be noraw mode.
9. **store()** : Save the terminla type data.
10. **restore()** : Restore the terminal type data.

2.11 WINDOW class

Window class handles the window on the terminal screen.

1. **window(int line, int col, int start_y, int start_x)** : Make a new window with "line" and '"col" size.
2. **makescroll()** : Make the window to be scrollable.
3. **clearall()** : Wipe out the window.
4. **cleartobottom()** : Wipe out the window fromt the current position to bottom.
5. **cleartoeol()** : Clear the current line to end of line.
6. **drawborder(char a, char b)** : Draw the border around the window.
7. **getstring(char *str)** : Get a string from the window.
8. **putstring(char *str)** : Put a string onto the window.
9. **getcharacter()** : Get a character from the window.

10. **putcharacter(char ch)** : Write a character onto the window.

11. **deletecharacter()** : Delete a character.

12. **display()** : Refresh the window.

13. **destroy()** : Remove a window.

14. **moveto(int y, int x)** : Move the cusor to (y,x) position on the window.

15. **get_cur_coordinate()** : Get a current cusor position on the window.

3 Algorithm

3.1 Process Control

Figure 2 shows how to control the processes between server and clients.

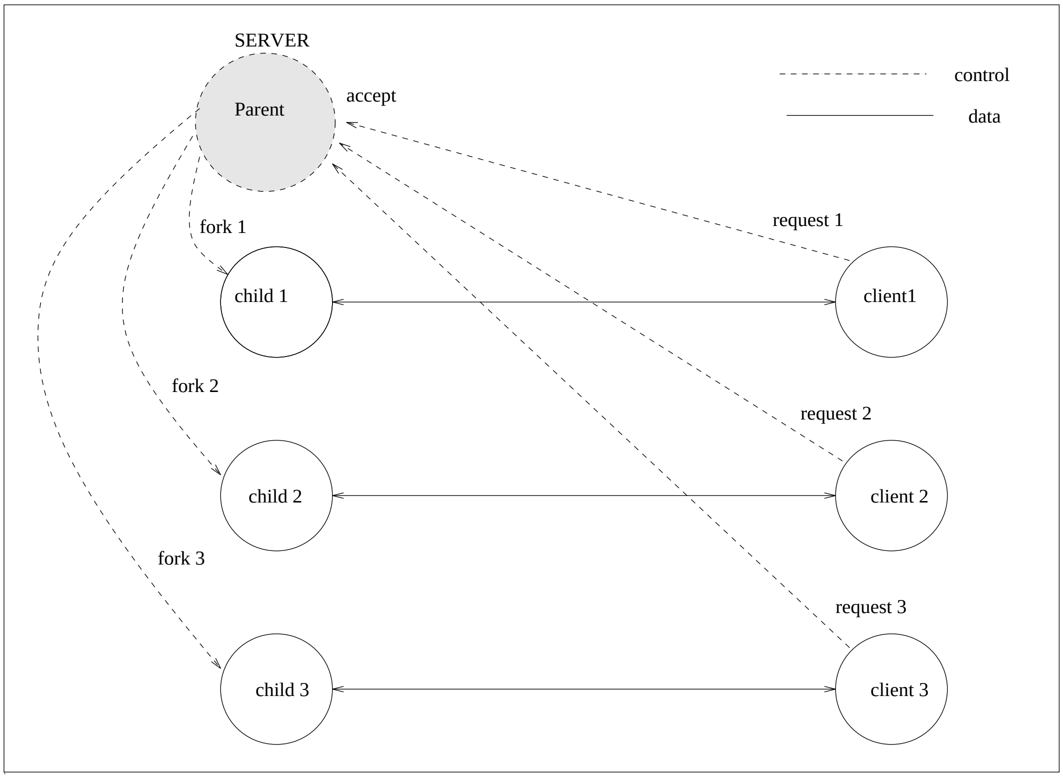

Figure 2: Process Control Diagram

When a server receives a connection request from a client, the server accepts it. After then, it forks and makes a new child process. The new child process communicate with the client. Every communication is done between the client and new child process.

3.2 Data Control

Figure 3 shows how to control the data between server and clients.

Client sends data to server. A child process correspoding to the client gets data, sends it to the parent process. After the parent process gets data from a child process, he sends data to every client who is joining currenltly. The parent process always maintains a list of all members.

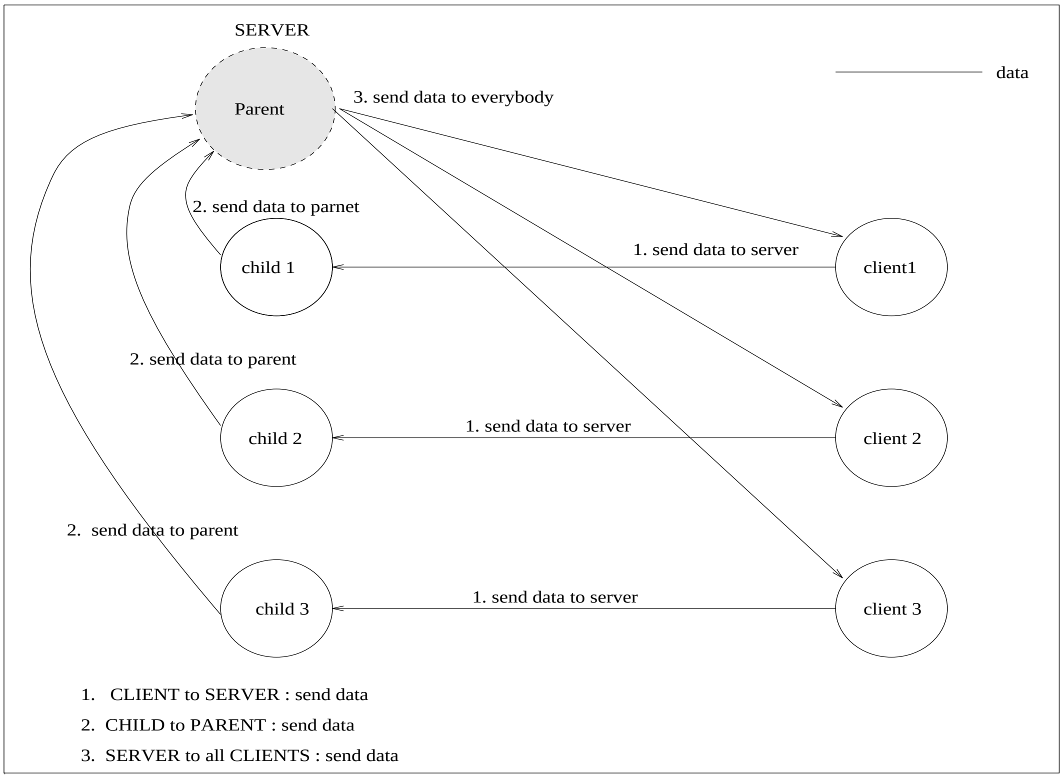

Figure 3: Dataflow Control Diagram

4 Manual

4.1 Server Part

1. Run the server program. You shoul know the IP address of the machine you are using.
   prompt % irc-server machine_address
   e.g. prompt % irc-server 128.230.25.13

4.2 Client Part

1. Connect to the SERVER machine. You should know the IP address of the server machine.
   prompt% irc-client server_address
   e.g. prompt% irc-client 128.230.25.13

2. Enter your ID which you want, if you see following message.
   prompt% Enter your ID :

3. After get into the Chat program, you will see the following screen.

   Figure 4 shows the user interface for client.

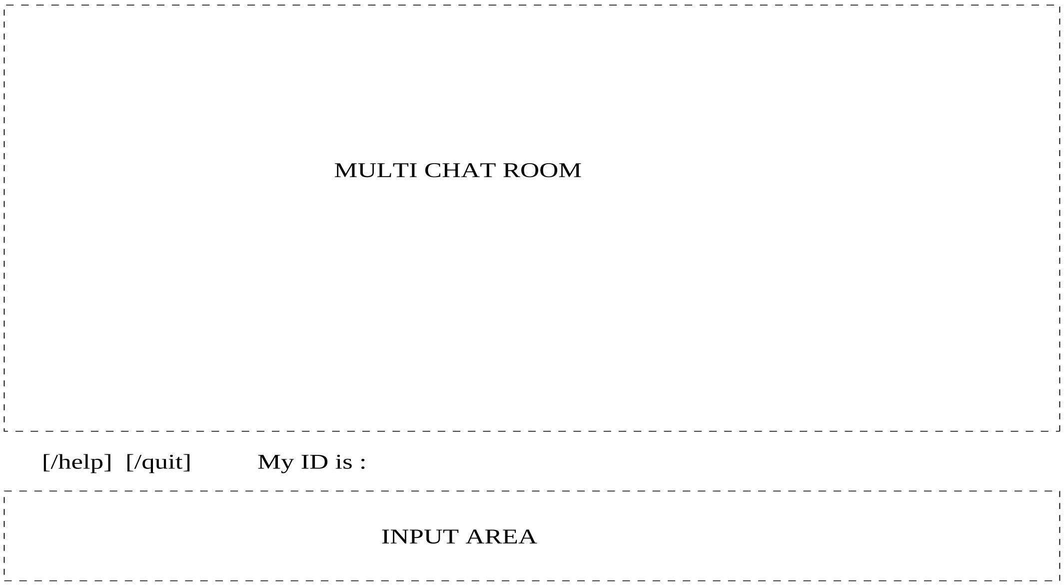

Figure 4: User Interface of Client

Bottom part of the window is for the input. Top part of the window is for the Chat room.

4.3 Command

1. **/help** : show the commands.
2. **/quit** : exit the program.
3. **/nick new_name** : Change the current ID to "new_name".
4. **/ID message** : Send the following "message" only to the person who is "ID".
5. **/ID1/ID2 ... /IDN message** : Send the following "message" to N people who are ID1, ID2 ... , IDN.

5 Source code

Please refer to attached source code. It consists of 3 parts.

- **Header Files.**
  1. ipc.h : ipc class definition.
  2. fork.h : forks class definition.
  3. pipe.h : pipes class definition.
  4. signal.h : signals class definition.
  5. socket.h : socket class definition.
  6. server.h : server class definition.
  7. client.h : client class definition.
  8. terminal.h : terminal class definition.
  9. window.h : window class defintion.
  10. member.h : members class definition.
  11. string.h : strings class definition.

- **Methods Files.**
  1. member.cc : members class methods.
  2. client.cc : client class methods.
  3. pipe.cc : pipe class methods.

4. signal.cc : signals class methods.
5. socket.cc : socket class methods.
6. server.cc : server class methods.
7. win.cc : window class methods.
8. string.cc : strings class methods.

- **Main Program.**
  1. irc-server.cc : main program of server part.
  2. irc-client.cc : main program of client part.