Naive implementation of a chat room

- C/C++ implementation of a chat room based on socket programming
- A demonstration of a simple client/server (CS) model

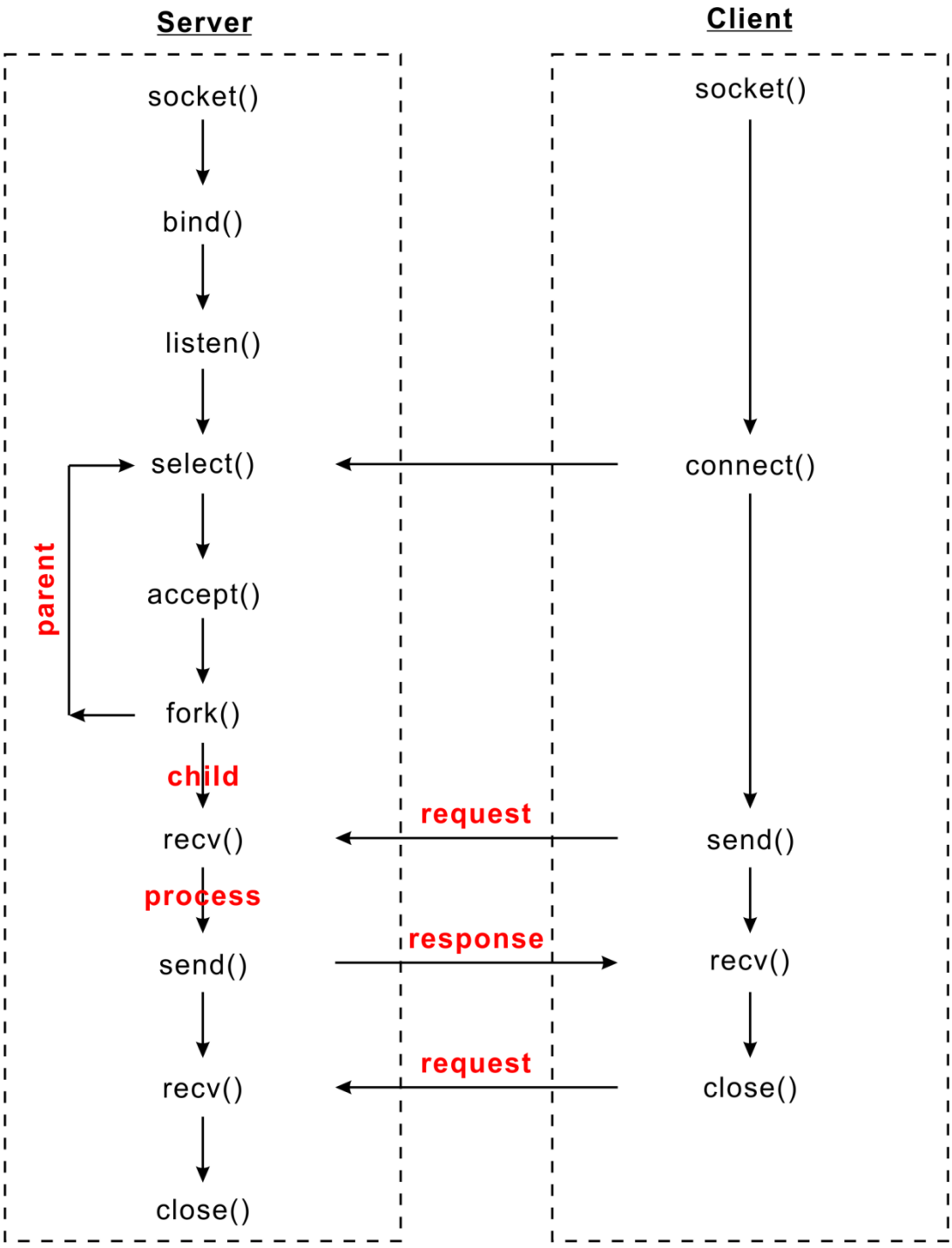

Fig. 1 The client/server model.

- ✧ Event handling pattern – server
  - ◆ Concurrency handling – multiprocessing (Fig. 2).
  - ◆ The main/parent process listens to the port for possible connection requests.
  - ◆ For each user, the parent process creates one child process to monitor the read/write tasks for this user.
  - ◆ For each user, the parent process creates one timer to record the inactive time duration. The timers are managed in a sorted linked list and delete inactive users periodically.
  - ◆ The parent process handles signals from the kernel: SIGCHLD, SIGTERM, SIGINT, SIGALRM etc. The signals are monitored by epoll_wait() via a local pipe (sig_pipe) just like I/O events.
  - ◆ The parent and child process communicate via pipes (Fig. 3).
  - ◆ One child process deals with one client socket: send() and recv() data from and to the client.
  - ◆ The child processes share a chunk of memory (read-only) via shm_open() and mmap() (Fig. 4) and thus the parent process only passes the index of the sender and other users read the data from corresponding shared memory.

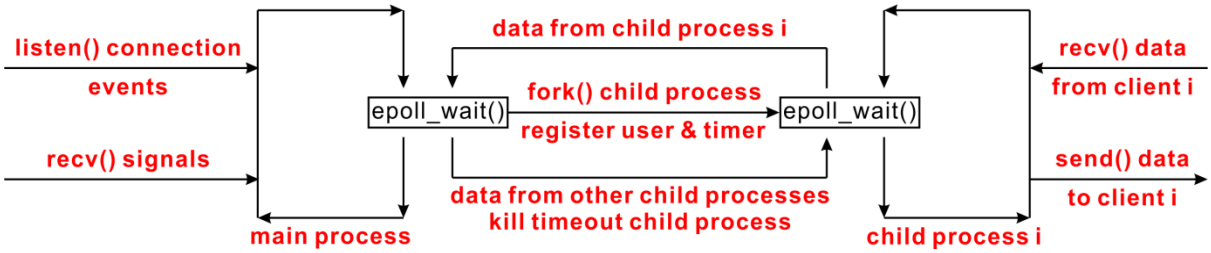

Fig. 2 The event handling pattern of the server.

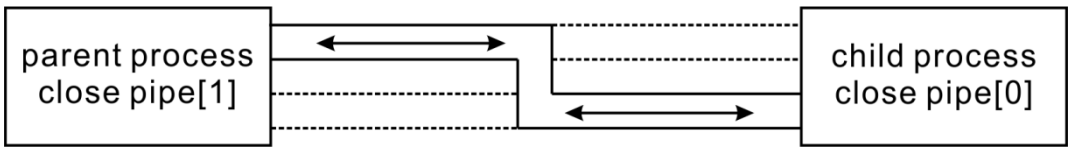

Fig. 3 Communication between the parent and child process.

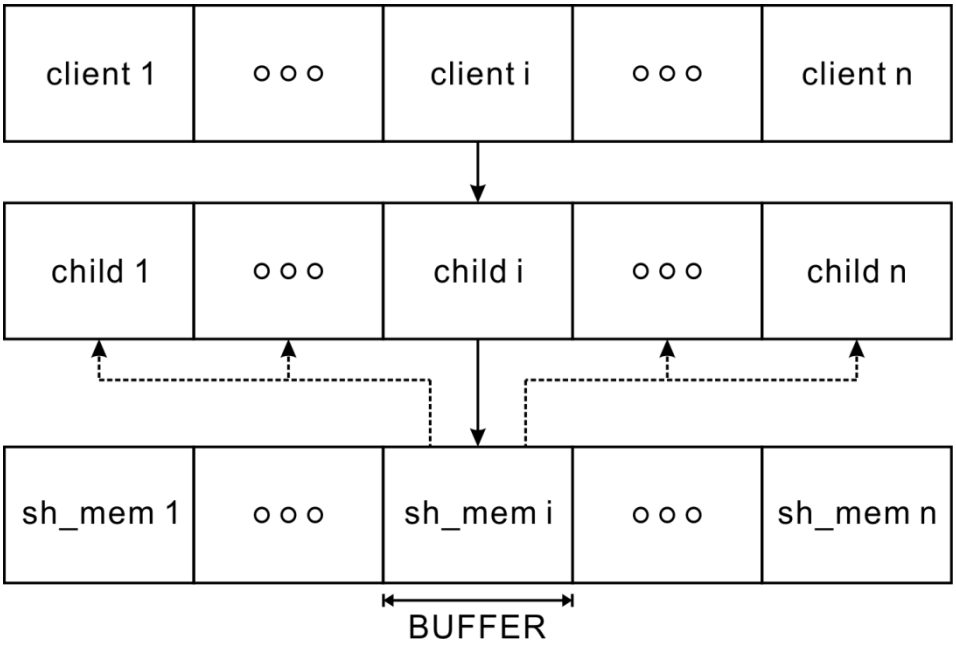

Fig. 4 Data transfer through clients.

✧ Event handling pattern – client

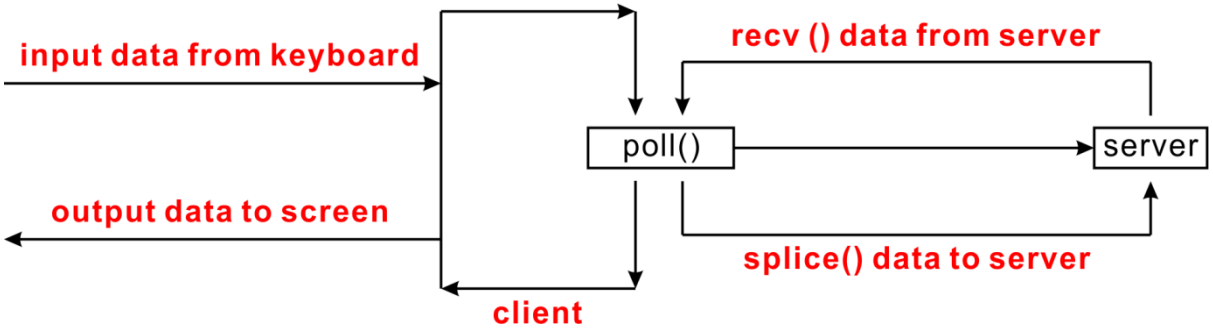

Fig. 5 The event handling pattern of the client.

✧ I/O multiplexing: epoll_wait()/poll()

wait for more than one descriptor to be ready

- network sockets (listening/connection sockets)
- signals
- standard input

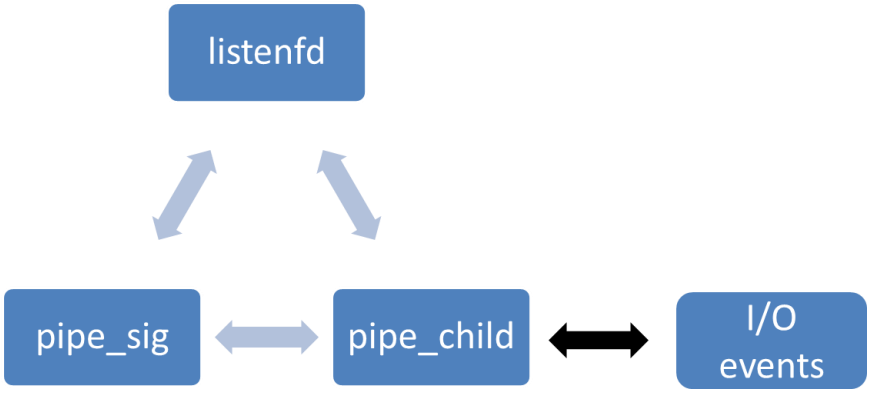

Fig. 6 I/O multiplexing in the server.

✧ The clients are stored in an array and corresponding process ids are stored in an unordered_map.

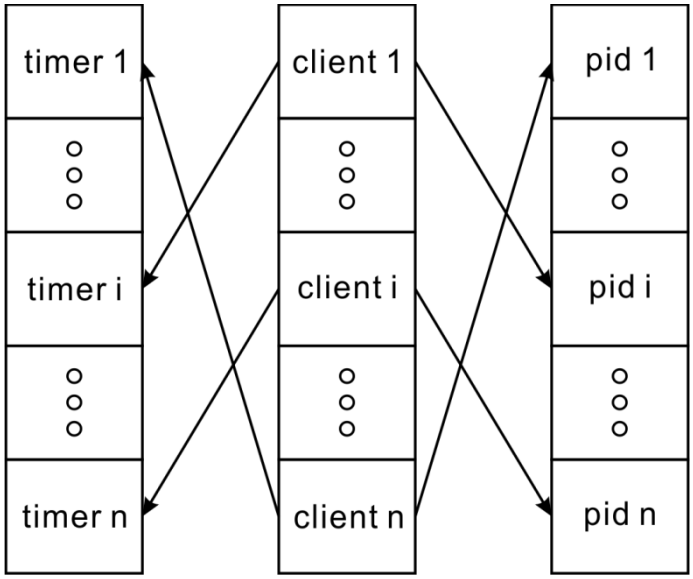

Fig. 7 Storage of client data, timers and process ids.

✧ When the clients close the connection actively

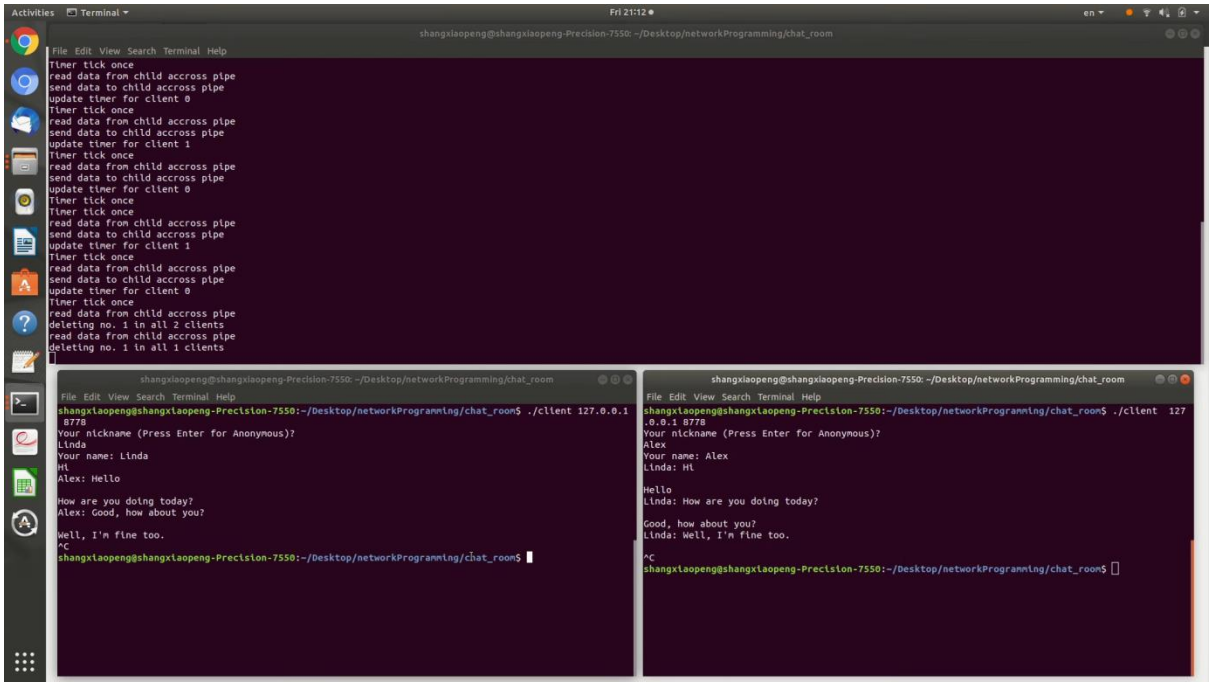

Fig. 8 Screenshot when the clients close the connection actively.

✧ When the clients close the connection passively due to timeout

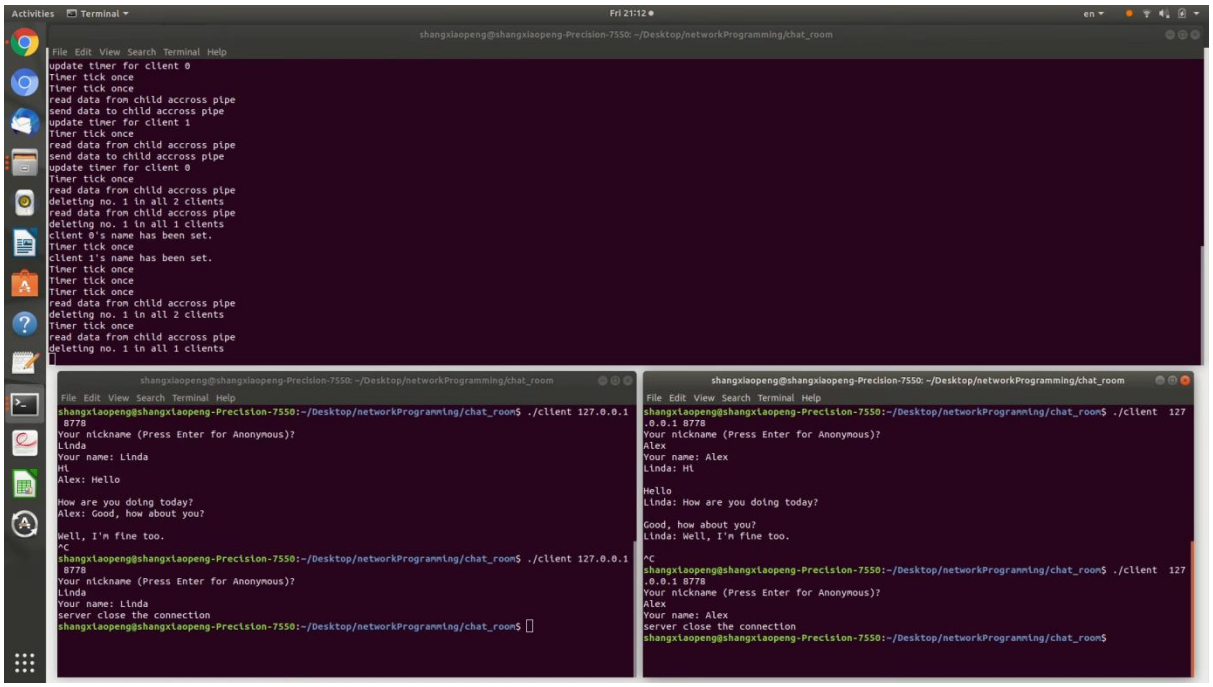

Fig. 9 Screenshot when the clients close the connection passively.