OS 2016

Homework2:

Shared Memory and Mailbox

(Due date: 2016/11/10 23:59:59)

Requirements

1. Implement Mailbox(Message Queue) APIs

```
mailbox_t mailbox_open(int id);
int mailbox_unlink(int id);
int mailbox_close(mailbox_t box);
int mailbox_send(mailbox_t box, mail_t *mail);
int mailbox_recv(malibox_t box, mail_t *mail);
int mailbox_check_empty(mailbox_t box);
int mailbox_check_full(mailbox_t box);
```

   - The APIs prototypes are fixed
   - You should not modify the function names or the meaning of the parameters and return values
   - The APIs must be implemented by the Linux built-in **POSIX** based shared memory mechanism
     - E.g., ***shm_open, shm_unlink, mmap, munmap, close***

2. Create a simple chatroom based on the Mailbox APIs
   - With three types of mail: JOIN, BROADCAST and LEAVE
   - Must support non-blocking I/O

Bonus

- Other mail types
  - e.g., “WHISPER” to sent private message to other clients, “LIST” to list online users
- Reliable protocol design
  - e.g., “ACK” mail
- Mechanism to prevent mail spoofing
  - Similar to **email spoofing** problem
  - Clients may send the mail with fake id to spoof the server
- Priority mailbox design
  - Mail has different priorities
  - Higher priority mails will be received before lower priority ones

Bonus(cont.)

- Multiple chatrooms
  - A client can switch to another chatroom
  - A client can only receive “BROADCAST” mail from other clients in the same chatroom
- Other challenging issues
  - Make the mailbox API as a dynamic shared library
  - Variable-sized mail
  - etc.

Mailbox Interface

Open a mailbox object

```
mailbox_t mailbox_open(int id);
```

- ***mailbox_open()*** creates and opens a new, or opens an existing mailbox object
  - Returns **NULL** on failure
- ***id*** specifies the name of the shared memory object
  - Ex: mailbox_open(12) will open the mailbox object: "/dev/shm/**__mailbox_12**" by calling **shm_open()**

```
typedef void *mailbox_t;
```

- ***mailbox_t*** can be any type you want, but cast to a void pointer

Unlink and Close the mailbox

```
int mailbox_unlink(int id);
```

- ***mailbox_unlink()*** removes a mailbox object

```
int mailbox_close(mailbox_t box);
```

- ***mailbox_close()*** only closes the link to mailbox
  - so the program is no longer refers to the mailbox, and the mailbox still exists
- You must actually release any resources created by ***mailbox_open()***
- On success, return 0; on failure, return -1

Send or Receive Mails

```
int mailbox_send(mailbox_t box, mail_t *mail);
int mailbox_recv(malibox_t box, mail_t *mail);
```

- ***mailbox_send()*** adds the mail to a mailbox
- ***mailbox_recv()*** receives a mail from a mailbox
- On success, return 0; on failure, return -1

About Mail

- Mail object is fixed size, defined as follows:

```
#define SIZE_OF_SHORT_STRING 64
#define SIZE_OF_LONG_STRING 512

typedef struct __MAIL {
    int from;
    int type;
    char sstr[SIZE_OF_SHORT_STRING];
    char lstr[SIZE_OF_LONG_STRING];
} mail_t;
```

  - ***from*** is the sender's mailbox id, ***type*** is the type of mail
  - According to the type of the mail, you will put a C-style string in the short string(***sstr***) and/or the long string(***lstr***)

Check Mailbox Status

```
int mailbox_check_empty(mailbox_t box);
int mailbox_check_full(mailbox_t box);
```

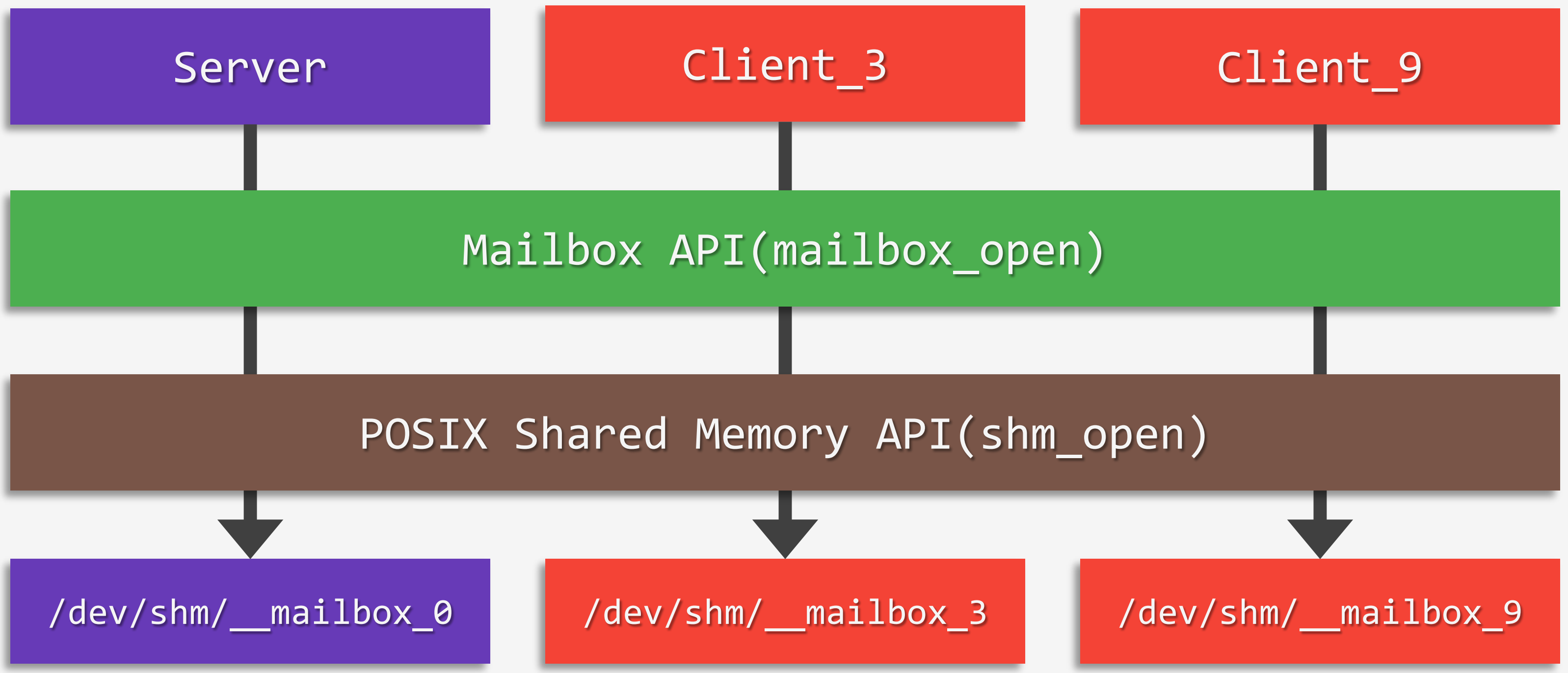
- To check whether the mailbox is empty/full or not
  - If true, return 1; if false, return 0
  - On failure, return -1

- Call ***mailbox_check_empty()*** before ***mailbox_recv()***
  - and call ***mailbox_check_full()*** before ***mailbox_send()***

Chatroom Design

Simple Chatroom

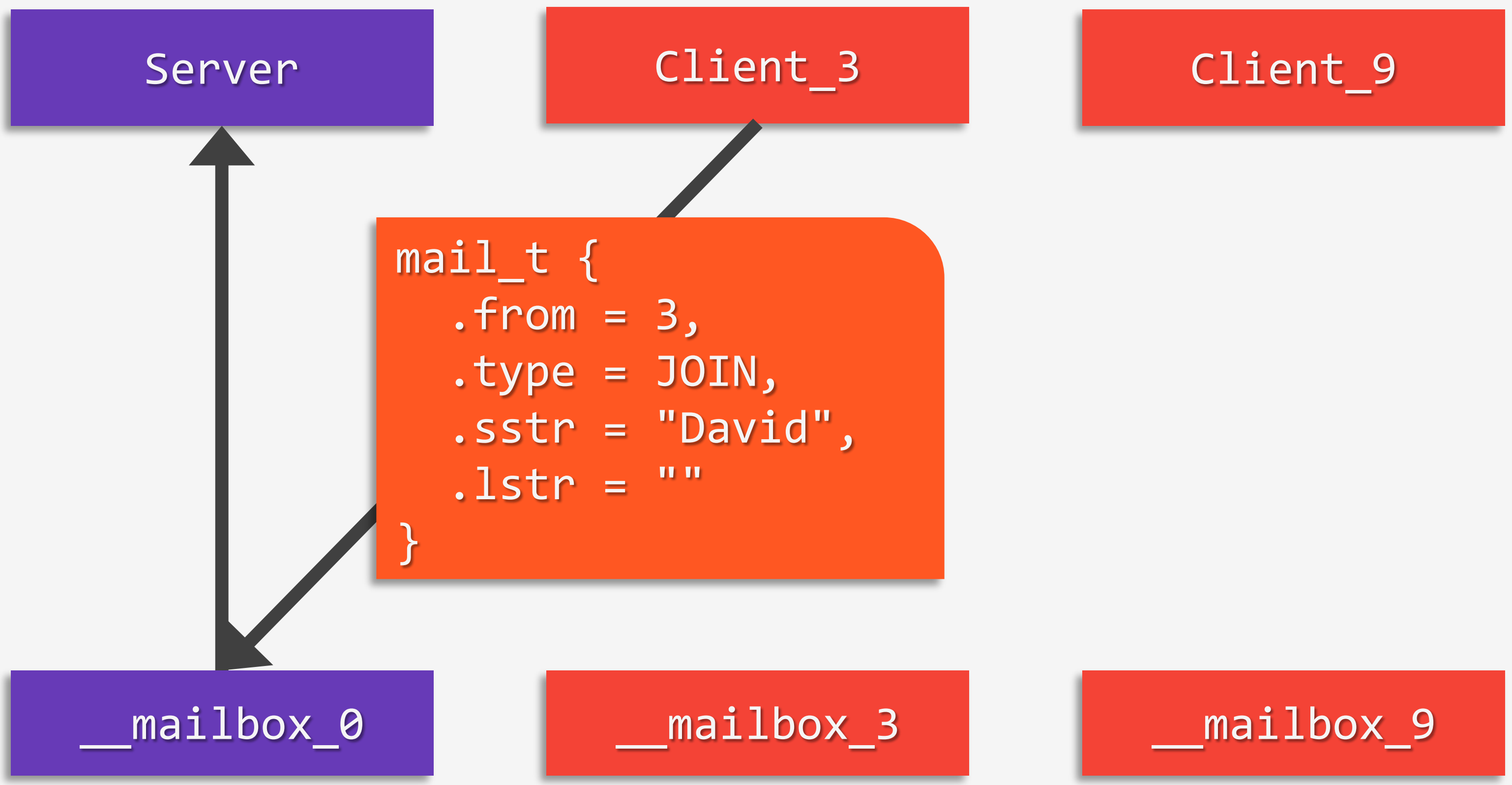
- Single server and multiple clients
- Client’s mail must be sent to server’s mailbox
    - Any mail send/receive between clients is illegal
- Server’s mailbox id fixed to 0



Mail – Join(1)

- Client sent “JOIN” mail to join into the chatroom
- Server must maintain a id-name map for online clients

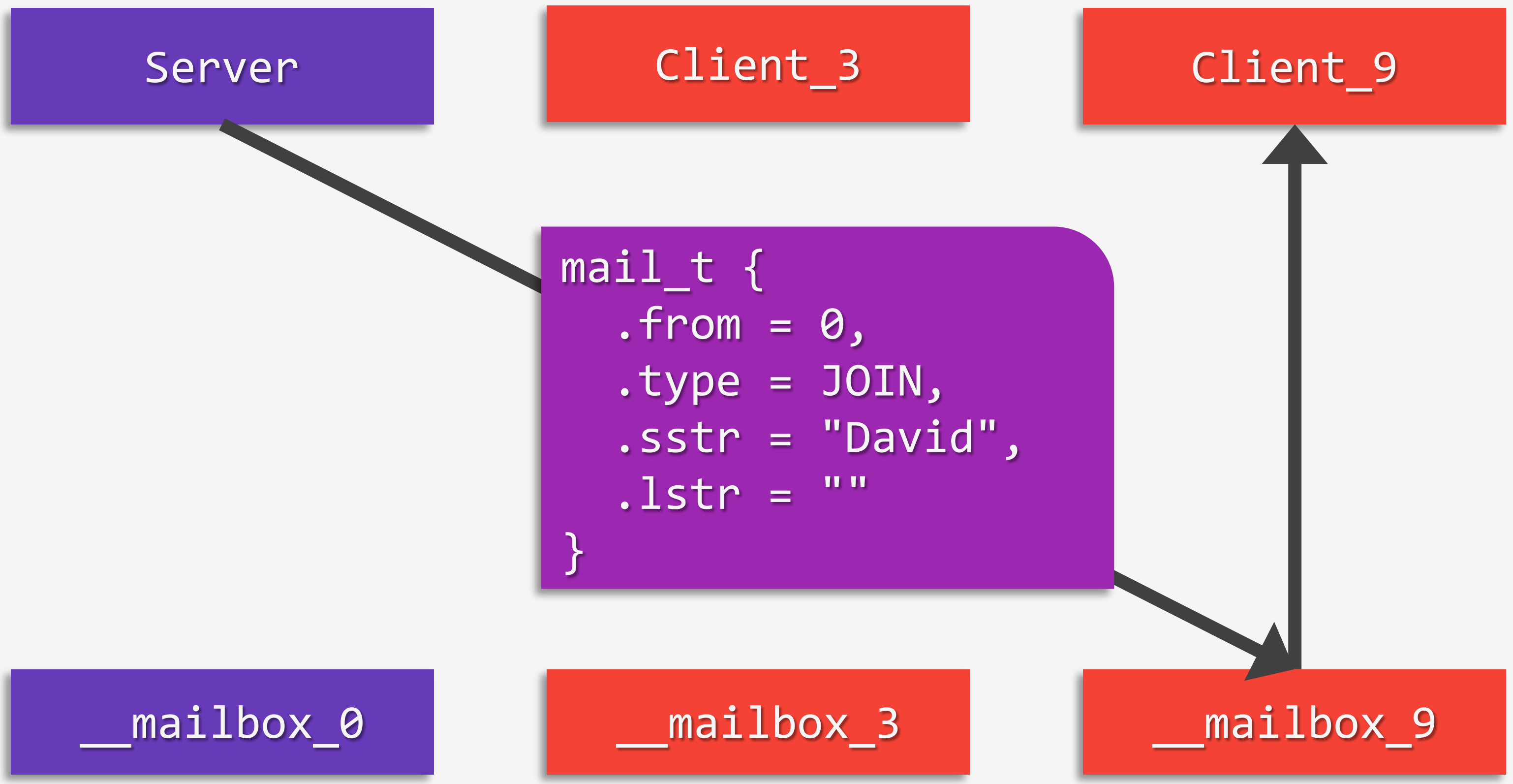
3	David
9	Candy



```
mail_t {
   .from = 3,
   .type = JOIN,
   .sstr = "David",
   .lstr = ""
}
```

Mail – Join(2)

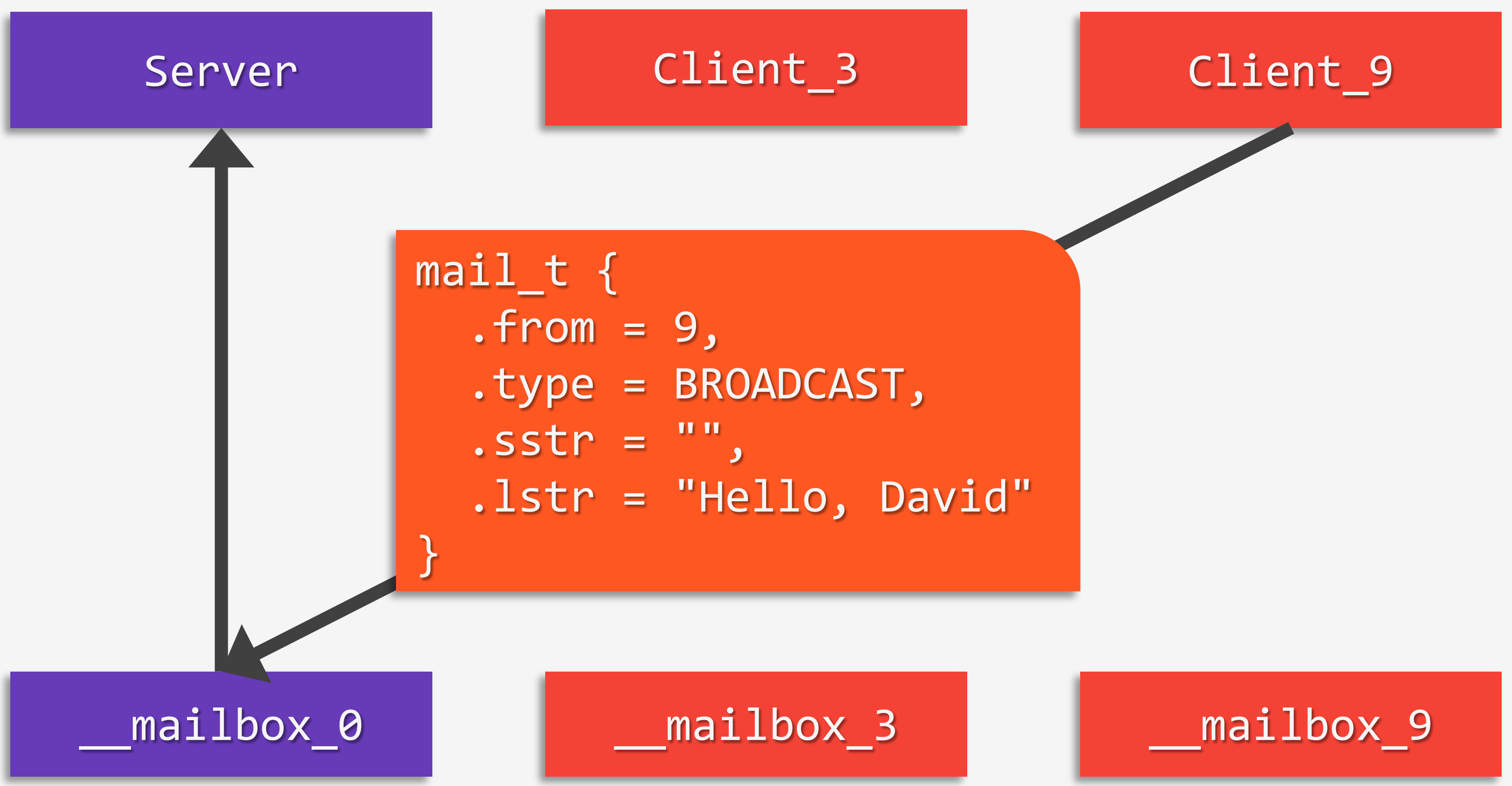
- Server will inform other clients with **alter** “JOIN” mail
  - So the client will only receive names rather than ids



```
mail_t {
  .from = 0,
  .type = JOIN,
  .sstr = "David",
  .lstr = ""
}
```

Mail – Broadcast(1)

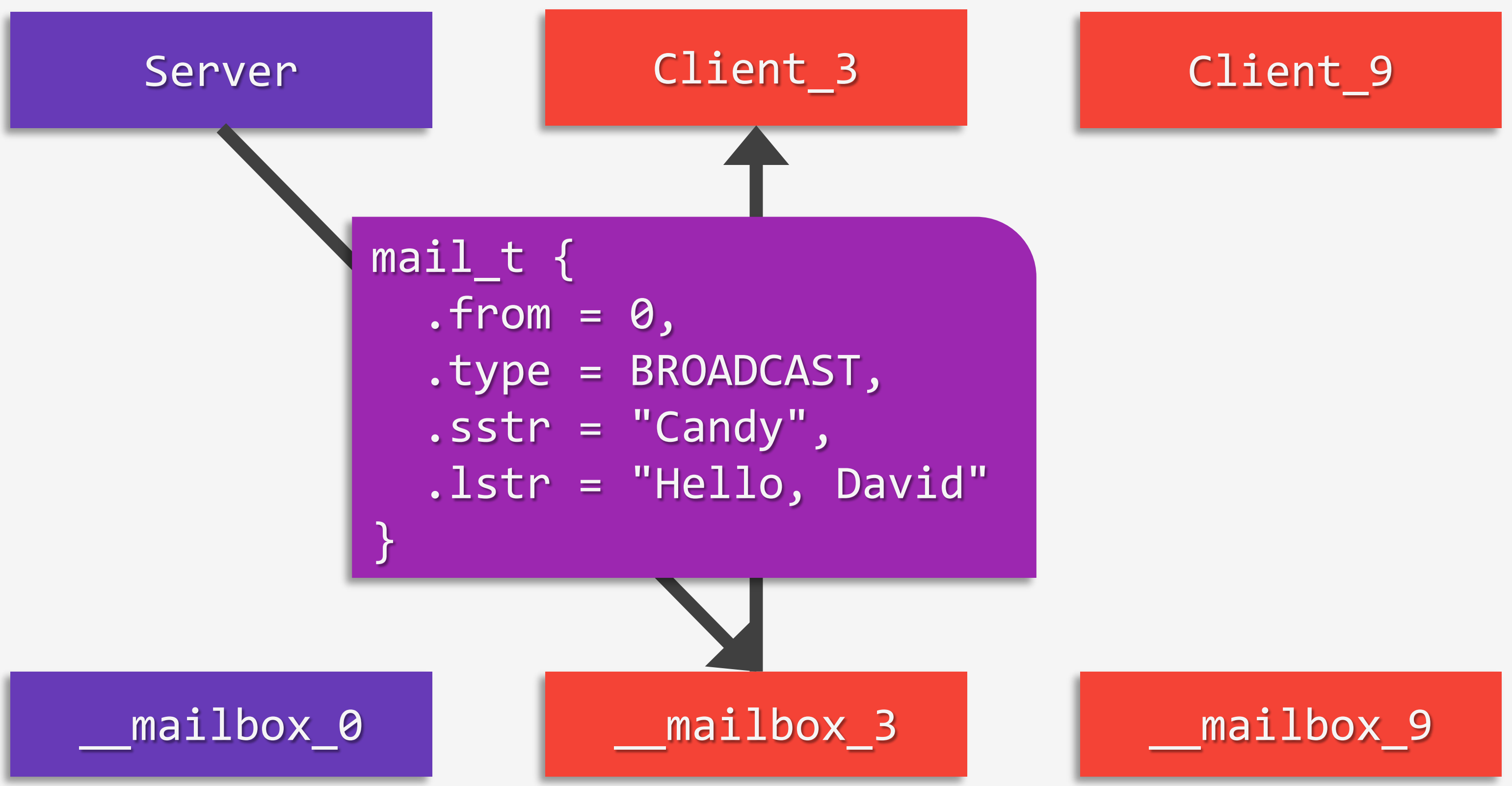
- After Join into the chatroom, client may sent “BROADCAST” mail to chat with others



```
mail_t {
  .from = 9,
  .type = BROADCAST,
  .sstr = "",
  .lstr = "Hello, David"
}
```

Mail – Broadcast(2)

- Server then broadcast an alter "BROADCAST" mail to other clients

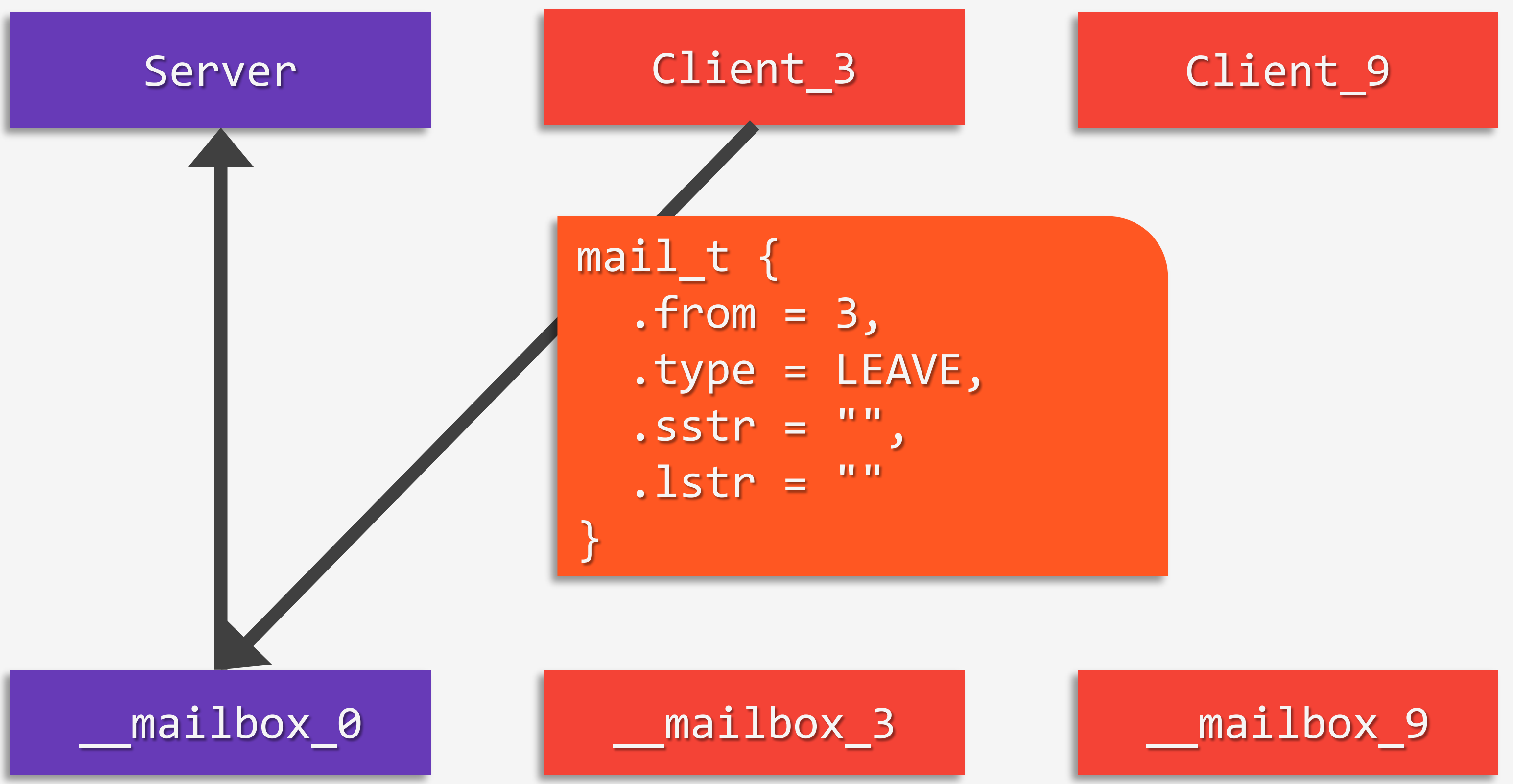


```
mail_t {
   .from = 0,
   .type = BROADCAST,
   .sstr = "Candy",
   .lstr = "Hello, David"
}
```

Mail – Leave(1)

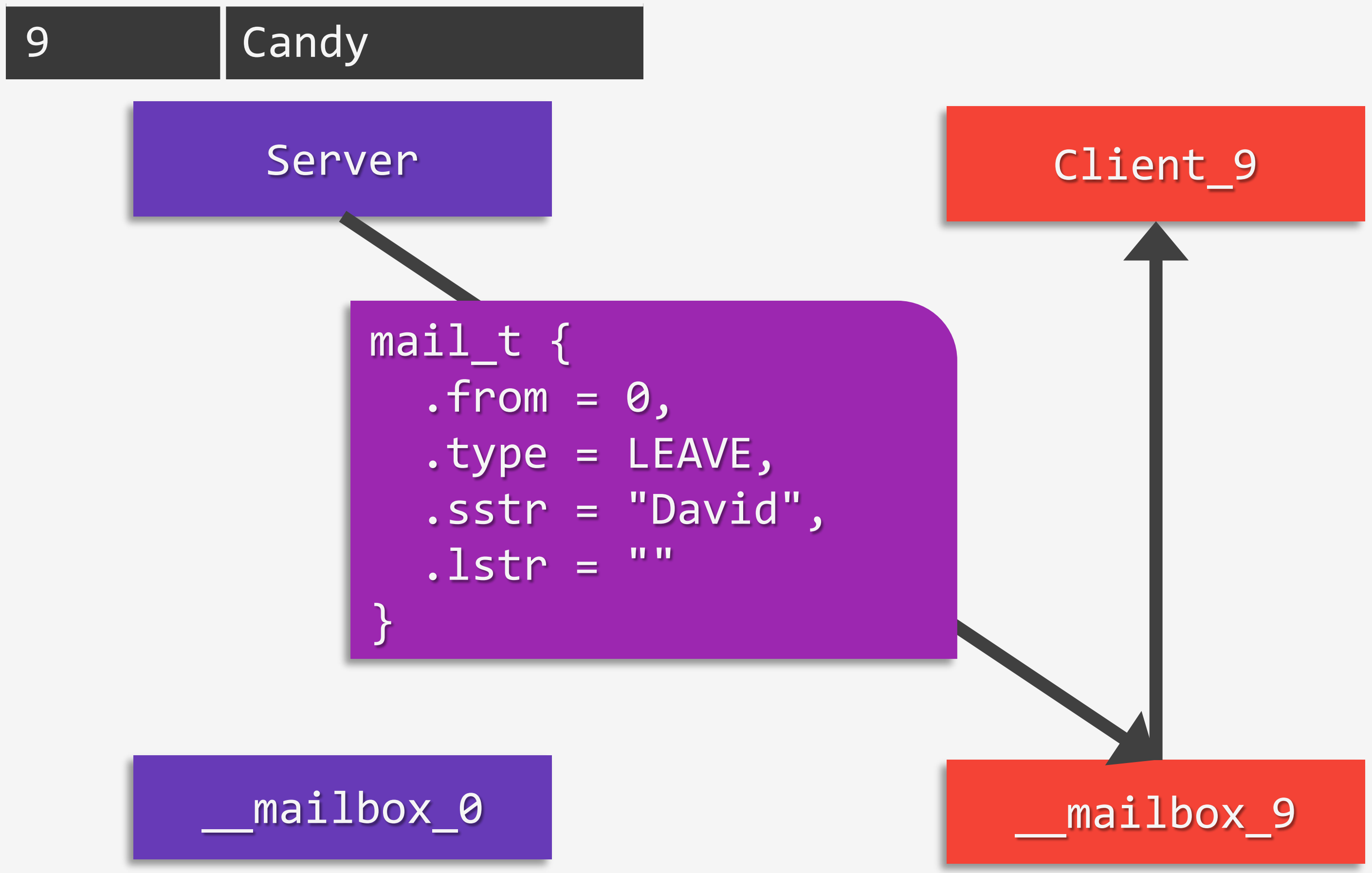
- Client sent “LEAVE” mail to leave the chatroom

~~3~~	~~David~~
9	Candy



```
mail_t {
   .from = 3,
   .type = LEAVE,
   .sstr = "",
   .lstr = ""
}
```

Mail – Leave(2)



```
mail_t {
   .from = 0,
   .type = LEAVE,
   .sstr = "David",
   .lstr = ""
}
```

Non-blocking I/O

- Clients must check user input(stdin) and the mailbox at the same time
- If clients use blocking I/O for stdin, the process will wait until user hit the enter key
  - If user don’t hit the enter key
    - any new mails will lying in the mailbox
    - and client process can’t show it immediately
- ***mailbox_send()*** and ***mailbox_recv()*** are non-blocking as well
- For stdin to support non-blocking I/O
  - Google it!

Reference

- Manual
    - shm_overview
    - fcntl