Operating System Lab 03

Task1

Write the server program, named as: server.c

Idea

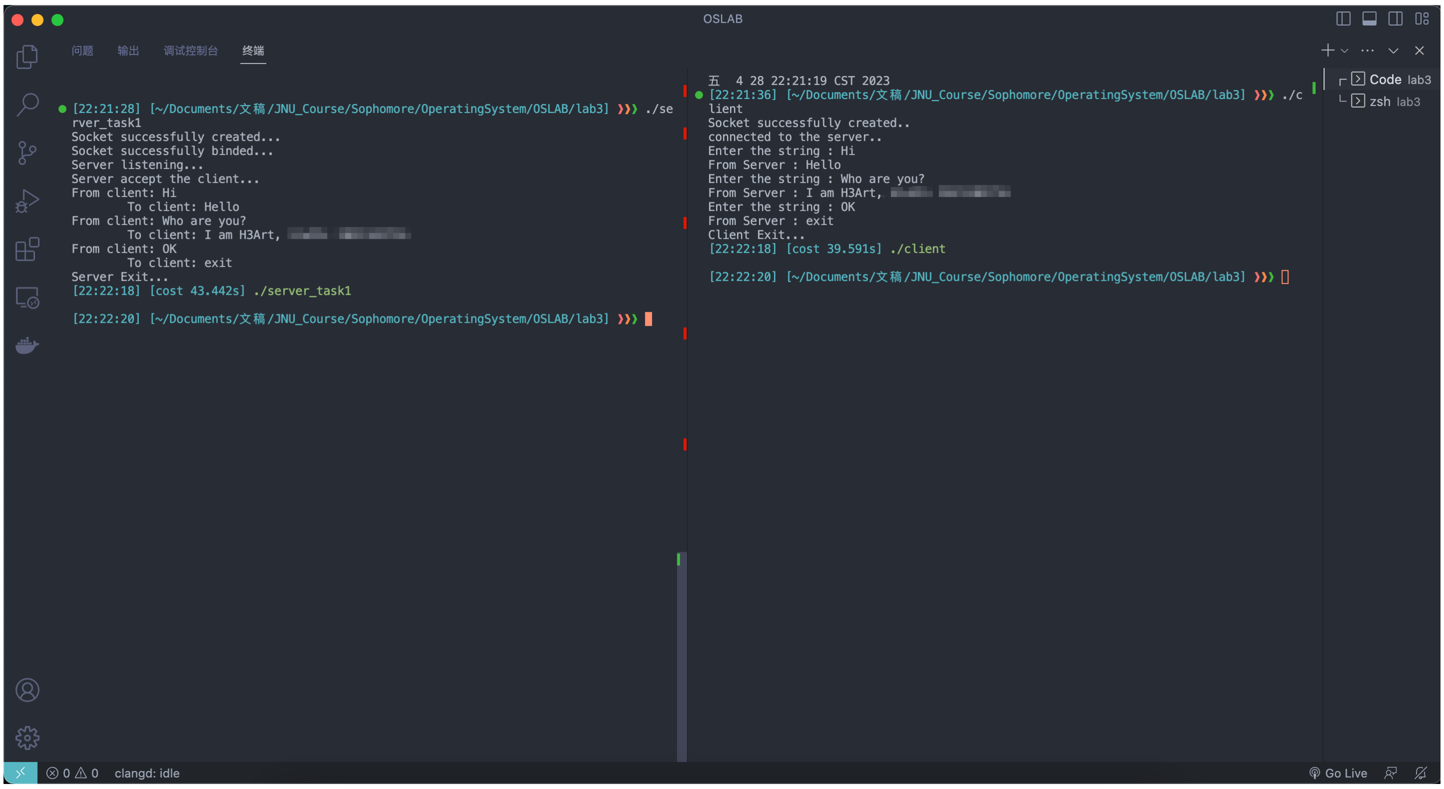
- The basic step of **establish connection** is:
  - create a socket identifier: `socket()`
  - set the socket protocol content: set the `struct sockaddr_in`
  - bind the socket identifier and protocol content: `bind()`
  - set the server to the listening state: `listen()`
  - accept the connection request from the client: `accept()`
- For **data exchange**, server needs to read the client's data and store them in the server's buffer, then print them. After that, server cleans the buffer and write its own message in it then send them to client.

Code Structure

```c
#include <arpa/inet.h> // For inet_addr()
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h> // For Unix/Linux system function: read()/write()/close()

#define MAX 80
#define PORT 8080
#define SA struct sockaddr

void data_xchg(int clientfd) {
  char buff[MAX];
  int n = 0;
  for (;;) {
    // Read data from client and store in buffer
    read(clientfd, buff, sizeof(buff));
    printf("From client: %s", buff);

    // Clear the buffer, prepare to write from server
    bzero(buff, sizeof(buff));

    // Read data from stdin and write them into buffer
    n = 0;
    printf("\tTo client: ");
    while ((buff[n++] = getchar()) != '\n')
      ;
    // Send the data from buffer to socket
    write(clientfd, buff, sizeof(buff));

    if ((strncmp(buff, "exit", 4)) == 0) {
      printf("Server Exit...\n");
      return;
    }
  }
}

int main(int argc, char *argv[]) {
  int sockfd, clientfd;
  struct sockaddr_in serv_addr;
  struct sockaddr_in cli_addr;
  unsigned int cli_addr_len = sizeof(cli_addr);

  // Create socket
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd == -1) {
    printf("Socket creation failed...\n");
    exit(EXIT_FAILURE);
  } else
    printf("Socket successfully created...\n");

  // Initialize each byte with 0
  bzero(&serv_addr, sizeof(serv_addr));
  bzero(&cli_addr, sizeof(cli_addr));

  // Assign IP, port
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
  serv_addr.sin_port = htons(PORT);

  // Bind the socket file descriptor and the IP port
  if (bind(sockfd, (SA *)&serv_addr, sizeof(serv_addr)) < 0) {
    printf("Socket bind failed...\n");
    exit(EXIT_FAILURE);
  } else {
    printf("Socket successfully binded...\n");
  }

  // Set the server to listening state
  if (listen(sockfd, 1) < 0) {
    printf("Socket listen failed...\n");
    exit(EXIT_FAILURE);
  } else {
```

```c
    printf("Server listening...\n");
  }

  // Accept the connection from client program
  clientfd = accept(sockfd, (struct sockaddr *)&cli_addr, &cli_addr_len);
  if (clientfd < 0) {
    printf("Server accept the client failed...\n");
    exit(EXIT_FAILURE);
  } else {
    printf("Server accept the client...\n");
  }

  data_xchg(clientfd);

  close(sockfd);

  return 0;
}
```

Execution Results



```
● [22:21:28] [~/Documents/文稿/JNU_Course/Sophomore/OperatingSystem/OSLAB/lab3] ❯❯❯ ./se
rver_task1
Socket successfully created...
Socket successfully binded...
Server listening...
Server accept the client...
From client: Hi
        To client: Hello
From client: Who are you?
        To client: I am H3Art, [redacted]
From client: OK
        To client: exit
Server Exit...
[22:22:18] [cost 43.442s] ./server_task1

[22:22:20] [~/Documents/文稿/JNU_Course/Sophomore/OperatingSystem/OSLAB/lab3] ❯❯❯
```

```
五  4 28 22:21:19 CST 2023
● [22:21:36] [~/Documents/文稿/JNU_Course/Sophomore/OperatingSystem/OSLAB/lab3] ❯❯❯ ./c
lient
Socket successfully created..
connected to the server..
Enter the string : Hi
From Server : Hello
Enter the string : Who are you?
From Server : I am H3Art, [redacted]
Enter the string : OK
From Server : exit
Client Exit...
[22:22:18] [cost 39.591s] ./client

[22:22:20] [~/Documents/文稿/JNU_Course/Sophomore/OperatingSystem/OSLAB/lab3] ❯❯❯
```

Task2

The above implementation can only perform **single-process communication**, that is to say, each time can only make a client connect to server for data communication, which obviously does not meet the basic requirements of the server.

We can modify the code on the server side by creating a child process each time after a client connection is successfully accepted by the server, and letting the child process handle the read and write data while the parent process continues to listen and accept.

Idea

- Modify the `accept()` part and surround it with an infinite loop to receive the multi-client connection requests
- Every time when accept a connection request from new client program, we `fork()` a child process to perform the **data exchange**
- In the process of using `fork()` to achieve the communication between multi-client and server, I found that the alternate execution of multi-client will destroy the order of standard output(in a sense, this is a kind of **concurrent bug**), resulting in the output of the server does not match the actual situation
- I modified the program to use **threads** to implement, and added a **mutex** mechanism to ensure that the input and output streams will not be interrupted

Code Structure

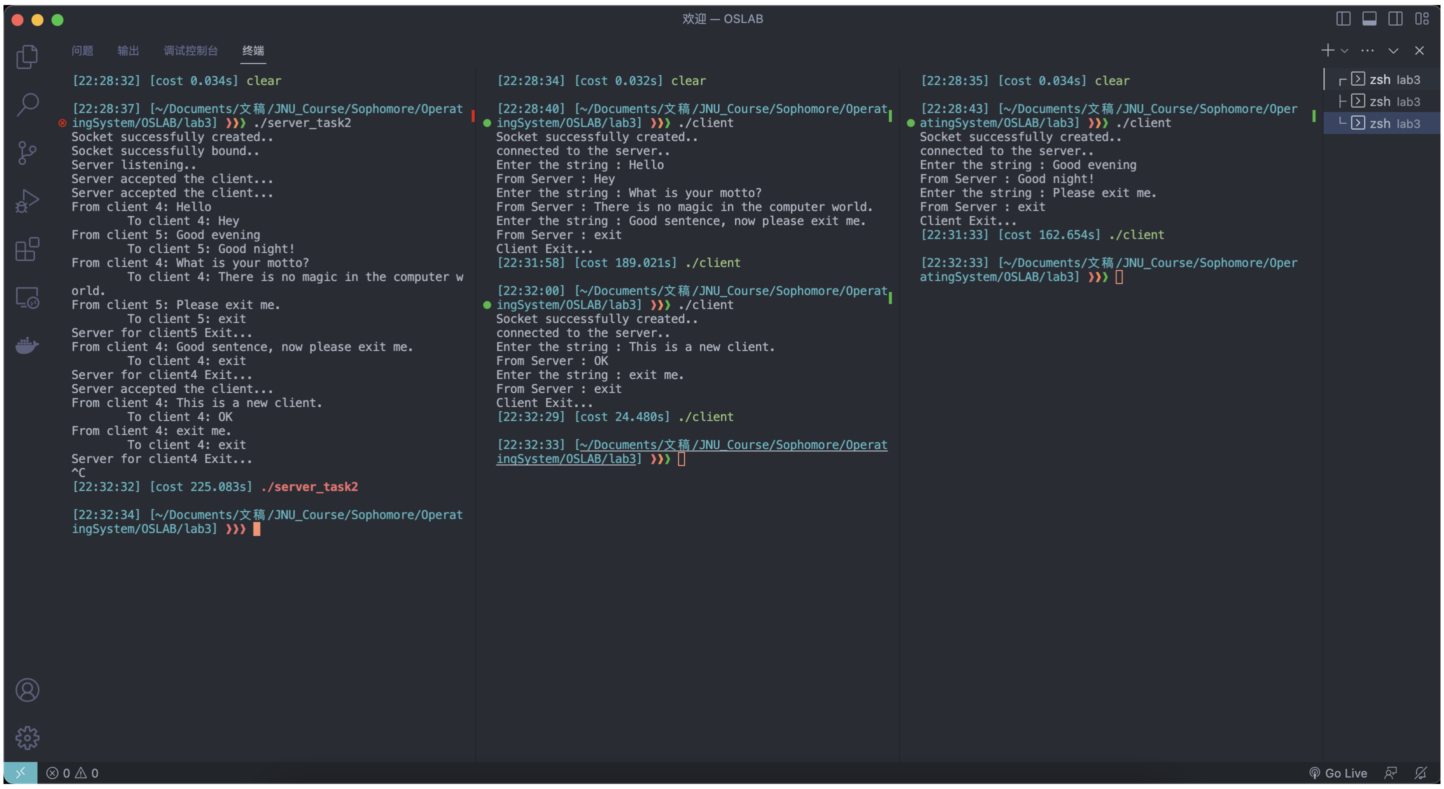
- part of `fork()` version:

```
// Accept the connection from client program
for (;;) {
  clientfd = accept(sockfd, (SA *)&cli_addr, &cli_addr_len);
  if (clientfd < 0) {
    printf("Server accept the client failed...\n");
    exit(EXIT_FAILURE);
  } else {
    printf("Server accept the client...\n");
  }

  pid_t pid = fork();
  if (pid == 0) {
    if (data_xchg(clientfd) == 0) {
      close(clientfd);
      return 0;
    }
  } else if (pid < 0) {
    fprintf(stderr, "Fork failed...\n");
    exit(EXIT_FAILURE);
  }
}
```

- using threads version

```c
#include <arpa/inet.h> // For inet_addr()
#include <netdb.h>
#include <netinet/in.h>
#include <pthread.h> // For pthread_create()/pthread_join()/pthread_mutex_*
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h> // For Unix/Linux system function: read()/write()/close()

#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Define mutex to synchronize stdout access
pthread_mutex_t stdout_mutex;

void *data_xchg_thread(void *arg) {
  int clientfd = *(int *)arg;
  char buff[MAX];
  int n = 0;
  for (;;) {
    // Read data from client and store in buffer
    read(clientfd, buff, sizeof(buff));

    // Acquire the lock to synchronize stdout access
    pthread_mutex_lock(&stdout_mutex);
    printf("From client %d: %s", clientfd, buff);

    // Clear the buffer, prepare to write from server
    bzero(buff, sizeof(buff));
    // Read data from stdin and write them into buffer
    n = 0;
    printf("\tTo client %d: ", clientfd);
    while ((buff[n++] = getchar()) != '\n')
      ;
    // Send the data from buffer to socket
    write(clientfd, buff, sizeof(buff));

    if ((strncmp(buff, "exit", 4)) == 0) {
      printf("Server for client%d Exit...\n", clientfd);
      close(clientfd);
      pthread_mutex_unlock(&stdout_mutex); // Release the lock before returning
      return NULL;
    }

    // Release the lock after finishing stdout access
    pthread_mutex_unlock(&stdout_mutex);
  }
}

int main(int argc, char *argv[]) {
  int sockfd, clientfd;
  struct sockaddr_in serv_addr;
  struct sockaddr_in cli_addr;
  unsigned int cli_addr_len = sizeof(cli_addr);

  // Create socket
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd == -1) {
    printf("Socket creation failed...\n");
    exit(EXIT_FAILURE);
  } else {
    printf("Socket successfully created..\n");
  }

  // Initialize each byte with 0
  bzero(&serv_addr, sizeof(serv_addr));
  bzero(&cli_addr, sizeof(cli_addr));

  // Assign IP, port
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  serv_addr.sin_port = htons(PORT);

  // Bind the socket file descriptor and the IP port
  if ((bind(sockfd, (SA *)&serv_addr, sizeof(serv_addr))) != 0) {
    printf("Socket binding failed...\n");
    exit(EXIT_FAILURE);
  } else {
    printf("Socket successfully bound..\n");
  }

  // Set the server to listening state
  if ((listen(sockfd, 10)) != 0) {
    printf("Listen failed...\n");
    exit(EXIT_FAILURE);
  } else {
    printf("Server listening..\n");
  }

  // Initialize the mutex
  pthread_mutex_init(&stdout_mutex, NULL);

  for (;;) {
    // Accept the data packet from client and verification
    clientfd = accept(sockfd, (SA *)&cli_addr, &cli_addr_len);
    if (clientfd < 0) {
      printf("Server accept failed...\n");
      exit(EXIT_FAILURE);
    } else {
      printf("Server accepted the client...\n");
    }

    // Create a thread to handle data exchange with this client
    pthread_t tid;
    pthread_create(&tid, NULL, data_xchg_thread, &clientfd);

    // Detach the thread to avoid memory leak
    pthread_detach(tid);
  }

  // Destroy the mutex before exiting
  pthread_mutex_destroy(&stdout_mutex);

  close(sockfd);

  return 0;
}
```

Execution Results



```
[22:28:32] [cost 0.034s] clear

[22:28:37] [~/Documents/文稿/JNU_Course/Sophomore/Operat
ingSystem/OSLAB/lab3] ❯❯❯ ./server_task2
Socket successfully created..
Socket successfully bound..
Server listening..
Server accepted the client...
Server accepted the client...
From client 4: Hello
        To client 4: Hey
From client 5: Good evening
        To client 5: Good night!
From client 4: What is your motto?
        To client 4: There is no magic in the computer w
orld.
From client 5: Please exit me.
        To client 5: exit
Server for client5 Exit...
From client 4: Good sentence, now please exit me.
        To client 4: exit
Server for client4 Exit...
Server accepted the client...
From client 4: This is a new client.
        To client 4: OK
From client 4: exit me.
        To client 4: exit
Server for client4 Exit...
^C
[22:32:32] [cost 225.083s] ./server_task2

[22:32:34] [~/Documents/文稿/JNU_Course/Sophomore/Operat
ingSystem/OSLAB/lab3] ❯❯❯
```

```
[22:28:34] [cost 0.032s] clear

[22:28:40] [~/Documents/文稿/JNU_Course/Sophomore/Operat
ingSystem/OSLAB/lab3] ❯❯❯ ./client
Socket successfully created..
connected to the server..
Enter the string : Hello
From Server : Hey
Enter the string : What is your motto?
From Server : There is no magic in the computer world.
Enter the string : Good sentence, now please exit me.
From Server : exit
Client Exit...
[22:31:58] [cost 189.021s] ./client

[22:32:00] [~/Documents/文稿/JNU_Course/Sophomore/Operat
ingSystem/OSLAB/lab3] ❯❯❯ ./client
Socket successfully created..
connected to the server..
Enter the string : This is a new client.
From Server : OK
Enter the string : exit me.
From Server : exit
Client Exit...
[22:32:29] [cost 24.480s] ./client

[22:32:33] [~/Documents/文稿/JNU_Course/Sophomore/Operat
ingSystem/OSLAB/lab3] ❯❯❯
```

```
[22:28:35] [cost 0.034s] clear

[22:28:43] [~/Documents/文稿/JNU_Course/Sophomore/Oper
atingSystem/OSLAB/lab3] ❯❯❯ ./client
Socket successfully created..
connected to the server..
Enter the string : Good evening
From Server : Good night!
Enter the string : Please exit me.
From Server : exit
Client Exit...
[22:31:33] [cost 162.654s] ./client

[22:32:33] [~/Documents/文稿/JNU_Course/Sophomore/Oper
atingSystem/OSLAB/lab3] ❯❯❯
```