PA3

SK UNIVERSITY

Update

- **2024-05-23**
  - Added the error codes for Login.
  - Added instructions on how to run server and client
    - » i.e., `./server port` and `./client address port [file]`

PA3

- **Goal**
  - Design and implement a reservation server that supports many clients
  - Server manages the reservation procedure for each client.
  - Client manages a user's seat by sending a query, containing the user's desired action, to the server.
    - » Important Note: Terms "user" and "client" are different.
  - Deadline: 2024-06-16 (Sun) 23:59

Restrictions

- **Restrictions**
  - Project must be compiled and executed in a Linux environment.
  - IP address and the socket are set up in the same way we have done in the lab.
  - Dynamically allocated resources should be freed before the program terminates.
    - » Resources include files, memory, threads, locks, and child processes.
  - This is a personal assignment.
    - » You can discuss the task together, but you must write the source code by yourself.

Restrictions

- **Restrictions**
  - Assignment submission time is based on the time the latest submission was submitted on iCampus.
  - The total score is 100 points, and 10 points are deducted per day (max 4 days).
  - Project files (i.e. Makefile and source code) should be in a directory called pa3, which should be compressed into a "student_id.tar.gz" tarball and must be submitted to iCampus.
  - The report is uploaded separately.
    - » This report contains the design and implementation of your project, and it should be a PDF with the name format, "student_id.pdf“

Restrictions

- **Restrictions**
  - When compiled, it should generate a server program called “pa3_server” and a client called “pa3_client”.
  - The most recently submission will be used to score PA3.
  - The following code contains the expected output when extracting the assignment.
    - » If the following code does not work, your project might not be graded correctly.

```
$ tar xvzf 20XXXXXXXX.tar.gz
$ cd pa3
$ make
$ ls | grep pa3
pa3_server pa3_client  (other files_with pa3)
```

Number of Cores

- **Starting this week, we will be working with concurrent applications**
  - Number of cores should be increased as much as possible (i.e., somewhere within the green, yellow range in VirtualBox). At least 4 cores is recommended.
  - The number of cores is referred to as NUM_CORES.

Number of Cores

- Getting number of cores in C:
  - » Based on https://stackoverflow.com/a/74744791, which contains multiple approaches as well.
  - » For this approach, we get the set of cores the application can run on using `sched_getaffinity`, and we store it in cpu_set.
    - » Needs _GNU_SOURCE to be defined (either in Makefile or C code)
    - » sched_getaffinity accepts pid, cpu_set size, and the set/mask.
  - » Then, we use CPU_COUNT_S to count the number of cores in cpu_set.

```c
#ifndef _GNU_SOURCE
    #define _GNU_SOURCE
#endif

#include <sched.h>
int get_num_cores() {
 cpu_set_t cpu_set;
 sched_getaffinity(0, sizeof(cpu_set), &cpu_set);
 return CPU_COUNT_S(sizeof(cpu_set), &cpu_set);
}
```

Prerequisites

- **Install the following packages**
  - sudo apt install libargon2-dev

- **In the Makefile**
  - add “-D_GNU_SOURCE” in the gcc options
  - add “-largon2” at the end of your linking command
    - » (i.e. `gcc file.c –largon2` or `gcc file.o –largon2`)

User

- Identified with a positive unsigned integer (user ID)
- A user can only use one client.
  - » If User 1 is currently logged in through a client, other attempts to log in as User 1 from another client will be rejected.
- A client can only accommodate one user.
  - » The client can only send queries of user 1 until user 1 successfully logs out.
- A user associated with a client is called an active user.
- A client associated with a user is an active client.

Server

- Total number of seats managed by the server is 256.
  - » Seat number is from 1 to 256.
- Each seat should be managed by the server using at least one synchronization mechanism and one synchronization variable.
- To lessen the burden on your system, the server uses NUM_CORES of threads.
  - » However, the number of active users can be larger than the number of threads. To handle this, we will use a thread pool.
  - » Here, we have a queue of tasks to process, and each thread will take a task from that queue.
  - » Implementing a thread-safe queue is actually the exercise for the last week.

Server

- Server should check if the user is logged in when the user attempts to book a seat.
- Server should send a response, containing the response code and the data, to the client after it receives a query from the client.
- If the server receives an action not known to the server, it should print “Action ACTION is unknown.”, where ACTION is the action given by the user.

- Running server: ./pa3_server port

Client

- **Two ways to interact with client**
  - File: file is passed as a command-line argument
    - » This file contains one or more requests/queries
      - » One line = request
    - » When all the requests in the file are processed, the client automatically sends a termination request to the server.
  - Interactive: REPL
    - » Client runs on infinite loop until it has to terminate (“0 0 0”)
  - Running client: ./pa3_client ip_address port [file]

Data

SK UNIVERSITY

Structures

- **You can use the following structures for sending data from the server and client.**
  - Request is sent by the client and received by the server.
  - Response is sent by the server and received by the client.

```
struct Request {
    uint32_t user;
    uint32_t size;
    uint8_t action;
    uint8_t* data;
};
```

```
struct Response {
    uint32_t code;
    uint32_t size;
    uint8_t* data;
};
```

Sending Data

- **To send data with unknown length, we use TLV (type-length-value) or LTV (length-type-value)**
  - Client sends one message to the server.
  - In the server,
    - » The associated thread locks the mutex to prevent other threads from accessing the client's data.
    - » Once locked, the thread can get the type and length.
    - » We can the read this data using this length.
    - » Afterwards, unlock the mutex.
- **The format of the data is a sequence of bytes (uint8_t[]), which can be safely casted using memcpy.**

Server

```c
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

typedef struct {
  uint32_t type;
  uint32_t length;
  uint8_t* value;
} tlv_t;
int recv_tlv(int fd, tlv_t* tlv) {
  uint8_t header[sizeof(tlv->type) + sizeof(tlv->length)];
  if (read(fd, header, sizeof(header)) == 0) {
    return -1;
  }
  memcpy(&(tlv->type), header, sizeof(tlv->type));
  memcpy(&(tlv->length), header + sizeof(tlv->type), sizeof(tlv->length));
  if (tlv->length == 0) {
    tlv->value = NULL;
    return 0;
  }
  tlv->value = (uint8_t*)malloc(tlv->length);  // Cast to uint8_t*
  if (read(fd, tlv->value, tlv->length) == 0) {
    return -1;
  }
  return 0;
}
```

Server

```c
int server_fd, client_fd;

void handle_exit() {
  close(client_fd);
  close(server_fd);
}

int main() {
  struct sockaddr_in server_addr, client_addr;
  socklen_t addr_len = sizeof(client_addr);

  server_fd = socket(AF_INET, SOCK_STREAM, 0);
  memset(&server_addr, 0, sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(31415);
  server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
    perror("bind");
    exit(1);
  }

  listen(server_fd, 5);

  printf("Server is listening...\n");
  client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &addr_len);
  atexit(handle_exit);
```

Server

```
  while (1) {
    tlv_t tlv;
    if (recv_tlv(client_fd, &tlv) == -1) {
      printf("Connection was terminated\n");
      break;
    } else {
      printf("Received - Type: %u, Length: %u, Value: ", tlv.type, tlv.length);
      switch (tlv.type) {
        case 0:
          printf("%p\n", tlv.value);
          break;
        case 1:
          // Cast uint8* -> char*
          printf("%s\n", (char*)tlv.value);
          break;
        case 2:
          // Cast uint8* -> int* then dereference to get int
          printf("%d\n", *(int*)tlv.value);
          break;
        default:
          break;
      }

      free(tlv.value);
    }
  }

  close(client_fd);
  close(server_fd);
  return 0;
}
```

Client

```c
#include <netinet/in.h>
#include <readline/history.h>
#include <readline/readline.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

typedef struct {
  uint32_t type;
  uint32_t length;
  uint8_t* value;
} tlv_t;

void send_tlv(int fd, tlv_t* tlv) {
  size_t total_size = sizeof(tlv->type) + sizeof(tlv->length) + tlv->length;
  uint8_t* buffer = malloc(total_size);

  memcpy(buffer, &(tlv->type), sizeof(tlv->type));
  memcpy(buffer + sizeof(tlv->type), &(tlv->length), sizeof(tlv->length));
  memcpy(buffer + sizeof(tlv->type) + sizeof(tlv->length), tlv->value,
         tlv->length);

  write(fd, buffer, total_size);

  free(buffer);
}
```

Client

```c
int main() {
  int client_fd;
  struct sockaddr_in server_addr;

  client_fd = socket(AF_INET, SOCK_STREAM, 0);
  memset(&server_addr, 0, sizeof(server_addr));

  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(31415);
  server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  if (connect(client_fd, (struct sockaddr*)&server_addr,
      sizeof(server_addr)) == -1) {
    perror("connect");
    exit(1);
  }

  char* line;
  while ((line = readline("Enter type (0, 1, 2): ")) != NULL) {
    if (strlen(line) > 0) {
      add_history(line);
      long long type = strtoll(line, NULL, 10);
      tlv_t tlv = {.type = type};
```

Client

```
      switch (type) {
        // no type
        case 0:
          tlv.type = 0;
          tlv.value = NULL;
          tlv.length = 0;
          break;
        // string
        case 1: {
          char* value = readline("Enter string: ");
          tlv.value = (uint8_t*)value;
          tlv.length = strlen(value) + 1;
          break;
        }         // integer
        case 2: {
          char* value = readline("Enter number: ");
          long long int_value = strtoll(value, NULL, 10);
          tlv.value = (uint8_t*)&int_value;
          tlv.length = sizeof(int_value);
          break;
        }
        default:
          fprintf(stderr, "Invalid type\n");
          free(line);
          continue;
      }
```

Client

```
        // switch-case...
        send_tlv(client_fd, &tlv);
    }

    free(line);
  }
  close(client_fd);
  return 0;
}
```

Actions

SK UNIVERSITY

Actions (Client)

- **There are six types of actions.**

Action ID	Name	Description	Data field
0	Termination	Terminates the connection between the client and the server	0
1	Log in	Attempt to log in	Password
2	Book	Attempt to book a seat	Seat number
3	Confirm booking	Check the number of seats booked by the user	0 or N/A
4	Cancel booking	Cancel the user's booking if the user has booked a seat	Seat number
5	Log out	Log out	N/A (no data passed)

Actions (Server)

- **Response code: server sends 0 on success and the error code on failure.**

Action ID	Name	Data value	Response codes on failure
0	Termination	N/A	1
1	Log in	N/A	1, 2, 3
2	Book	The booked seat number (e.g. 3)	1, 2, 3
3	Confirm booking	List of available/booked seat numbers (e.g. 3,4,5,6)	1
4	Cancel booking	The canceled seat number (e.g. 3)	1
5	Log out	N/A	1

Actions

- **Print format: On success, print to stdout. On failure, print to stderr.**
- **ERROR_MSG can be found in the Errors section.**

Action ID	Name	On success	On failure
0	Termination	Connection terminated.	Failed to disconnect as ERROR_MSG.
1	Log in	Logged in successfully.	Failed to log in as ERROR_MSG.
2	Book	Booked seat SEAT_NUMBER.	Failed to book as ERROR_MSG.
3	Confirm booking	"Booked the seats seats1, seats2, etc.." or "Did not book any seats."	Failed to confirm booking as ERROR_MSG.
4	Cancel booking	Canceled seat SEAT_NUMBER.	Failed to cancel booking as ERROR_MSG.
5	Log out	Logged out successfully.	Failed to log out as ERROR_MSG.
<Other>	Unknown action	-	Action <Other> is unknown.

Action 0: Termination

- **Termination: Terminates connection and client**
  - If requests are passed using a file and not STDIN, the client automatically sends the termination request once all actions have been performed.
  - Otherwise, the user must send the request directly to the server.
  - Before sending this request, first check if a user is currently logged in.
    - » If so, send a logout request before sending this request.
    - » Assume that the logout request does not fail.

Action 0: Termination

- **Termination: Terminates connection and client**
  - The client terminates once it receives 0 as the response code from the server.
  - Before terminating the program, print "Connection terminated" to stdout in the client.

Action 0: Termination

- **Termination**
  - If not successful, print "Failed to disconnect as ERROR_MSG." instead to stderr.
    - » ERROR_MSG here refers to the second column in the Errors section.
- **Errors**

Response code	ERROR_MSG	description
1	arguments are invalid	not all the values of the fields are 0

Action 1: Login

- **Login: assign user as client's active user**
  - Data: a variable-length password.
  - A user who has executed this action successfully is referred to as an active user.
  - A client associated with an active user is referred to as an active client.
  - Action makes the requested user the client's active user if successful.
  - Server does not send any data.

Action 1: Login

- **Storing passwords**
  - Passwords will be stored in /tmp/passwords.tsv.
    - » Create this file if it does not exist, but open in append mode.
    - » Storing passwords in plaintext can be a bad habit, so we will hash the password.
    - » Each line refers to a user and has the following format.
      - » “USER_ID\tHASHED_PASSWORD”
      - » \t is the delimiter
    - » Use the “hash_password” and “validate_password” functions in the following example to obtain HASHED_PASSWORD and to check if the password is correct.
    - » You need to install and link argon2 (see prerequisites).
    - » If user is not in /tmp/passwords.tsv, treat this as registration.

Action 1: Login

- **Safely read tsv file and find the line that contains the user id.**
- **If the user exists,**
  - Extract the hashed_password form that line.
  - Compare two passwords using “validate_password” that is found in the example code.
    - `validate_password(char* password_to_validate, char* hashed_password)`
      - password_to_validate: Where the password passed by the user is located.
      - hashed_password: Where your hashed password is located.
  - Returns 1 if true, 0 if false.

Action 1: Login

- **If the user exists,**
  - Call “hash_password” that is found in the example code.
    - `hash_password(char* password, char* hashed_password)`
      - password: Where your password is located
      - hashed_password: Where your hashed password will be saved.
        - Size: 128
  - Safely append “USER_ID\tHASHED_PASSWORD” to tsv file.

Action 1: Login

```c
#include <argon2.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define MEMORY_USAGE 512
#define SALT_SIZE 16
#define HASH_SIZE 32
#define HASHED_PASSWORD_SIZE 128

void generate_salt(uint8_t* salt) {
  int fd = open("/dev/urandom", O_RDONLY);
  read(fd, salt, SALT_SIZE);
  close(fd);
}
```

Action 1: Login

```
void hash_password(char* password, char* hashed_password) {
  uint8_t salt[SALT_SIZE];
  generate_salt(salt);
  char hash[HASHED_PASSWORD_SIZE];
  argon2id_hash_encoded(2, MEMORY_USAGE, 1, password,
        strlen(password), salt, SALT_SIZE, HASH_SIZE,
        hash, HASHED_PASSWORD_SIZE);

  strcpy(hashed_password, hash);
}

int validate_password(char* password_to_validate,
                      char* hashed_password) {
  if (argon2id_verify(hashed_password, password_to_validate,
                      strlen(password_to_validate)) == ARGON2_OK) {
    return 1;
  } else {
    return 0;
  }
}
```

Action 1: Login

```c
int main() {
  char* input = readline("Enter your actual password: ");
  if (!input) {
    return 0;
  }

  char hashed_password[HASHED_PASSWORD_SIZE];
  hash_password(input, hashed_password);

  uint32_t uid = 1;
  // Now you can save (user id) and hashed_password to a file.
  // ...
```

Action 1: Login

```
  while (1) {
    char* input = readline("Enter a password: ");
    if (!input) {
      break;
    }
    int is_correct = validate_password(input, hashed_password);
    if (is_correct) {
      puts("Correct password!");
    } else {
      puts("Incorrect password!");
    }

    free(input);
  }

  return 0;
}
```

Action 1: Login

- **Errors**

Response code	ERROR_MSG	Description
1	user is active	an active user attempts to log in when already logged in on another client
2	client is active	any user, including the active user, tries to log in on an active client.
3	password is incorrect	a registered user attempts to log in with the wrong passcode

Action 2: Book

- **Book: Books seat for the user**
  - Seat number has to be between 1 and 256
    - » (inclusive, i.e., [1, 256]).
  - Response's data field: booked seat number
  - If successful, print "Booked seat SEAT_NUMBER." on the client, where SEAT_NUMBER is the number of the seat that was booked.
  - If not, print "Failed to book as ERROR_MSG." instead.
    - » ERROR_MSG here refers to the second column in the Errors section.

Action 2: Book

- **Errors**

Response code	ERROR_MSG	description
1	user is not logged in	user attempts to book when it is not the active user in the client
2	seat is unavailable	seat requested by the active user is already booked by another user (including the user)
3	seat number is out of range	seat number is out of the range.

Action 3: Confirm booking

- **Confirm booking: Check available/booked seats**
  - Response's data field
    - » If request's data field is 0, server returns all available seats.
    - » If request's data field is empty (i.e. length == 0), return seats booked by the user.
  - Client must print the booked seats in ascending order with the following format: "Booked the seats seat1, seat2, etc.." (i.e., "Booked the seats 1, 10, 21.").
  - If the user has not booked any seats, print "Did not book any seats." in the client.

Action 3: Confirm booking

- **Confirm booking**
  - If the action was not processed successfully, print "Failed to confirm booking as ERROR_MSG." instead.
    - » ERROR_MSG here refers to the second column in the Errors section.
- **Errors**

Response code	ERROR_MSG	description
1	user is not logged in	user attempts to confirm booking when it is not the active user in the client

Action 4: Cancel booking

- **Cancel booking: Removes specified seat from user's booked seats.**
  - Seat number has to be between 1 and 256
    - » (inclusive, i.e., [1, 256]).
  - Response's data field: the canceled seat number.
  - If successful, print "Canceled seat number SEAT_NUMBER." on the client, where SEAT_NUMBER is the number of the seat that was booked.
  - If not, print "Failed to cancel booking as ERROR_MSG." instead.
    - » ERROR_MSG here refers to the second column in the Errors section.

Action 4: Cancel booking

- **Errors**

Response code	ERROR_MSG	description
1	user is not logged in	user attempts to cancel a booking when it is not the active user in the client
2	user did not book the specified seat	user attempts to cancel a booking for a seat that has not been reserved or a seat booked by another user
3	seat number is out of range	seat number is out of the range.

Action 5: Log out

- **Log out: Logs the user out**
  - Removes the user's status as active user.
  - Data field: no data sent (length == 0).
  - If successful, print "Logged out successfully." on the client.
  - If not, print "Failed to log out as ERROR_MSG."
    - » ERROR_MSG here refers to the second column in the Errors section.
- **Errors**

Response code	ERROR_MSG	description
1	user is not logged in	user attempts to book when it is not the active user in the client

Examples of client/server

Example

	<Client>	<Terminal>
	[user action data]	
The user has not logged in yet	5 2 12	Failed to book as user is not logged in.
so these queries will fail.	3 5	Failed to log out as user is not logged in.
	1 3	Failed to confirm booking as user is not logged in.
User 7 logged in with 'password'	7 1 password	Logged in successfully.
User 1 is not logged in -> fail	1 3	Failed to confirm booking as user is not logged in.
User 7 booked seat 12	7 2 12	Seat 12 booked.
Seat 1025 is out of range -> fail	7 2 1025	Failed to book as seat number is out of range.
10 is an unknown action -> fail	7 10 5	Action 10 is unknown.
User 7 logged out	7 5 0	Logged out successfully.
User 3 logged in with 'hunter2'	3 1 hunter2	Logged in successfully.
Seat 12 already booked -> fail	3 2 12	Failed to book as seat is unavailable.
User 3 booked seat 21	3 2 21	Seat 21 booked.
User 3 booked seat 22	3 2 22	Seat 22 booked.
User 3 confirms the booking	3 3	Booked the seats 21, 22.
User 3 logs out	3 5	Logged out successfully.
User 7 logs in with the wrong password	7 1 abc	Failed to log in as password is incorrect.

Example

<Client 1>	<Client 2>	Description
5 1 password		
	5 1 password	Failed as user 5 is already logged in on client 1
	3 1 hunter2	
5 2 10		
	3 2 10	Failed as user has already booked seat 10
5 4 10		
	3 2 10	
5 5		
	5 1 hunter2	Failed as user 5's password is password