PETRUCCI Titouan - 2210009

Creating a Basic File Transfer Protocol

using the C programming language

University of Sciences and Technology of Hanoi

Contents

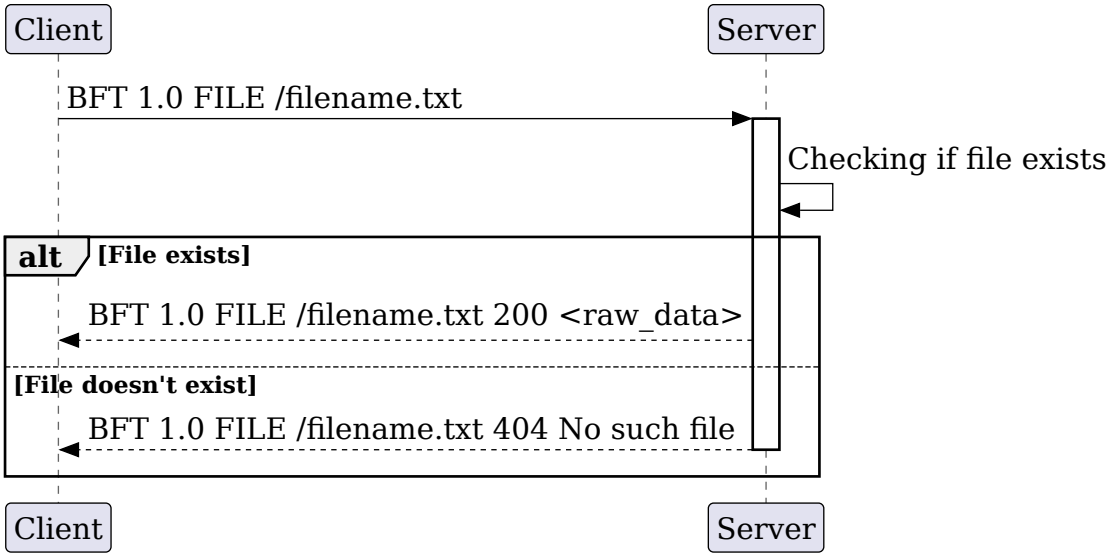
I/ Protocol

To create the protocol, we need to establish what are the real needs

I.1/ Requirements

- Clients would be able to download files from the server
- Clients can download multiple files in a row
- Protocol need to support upgrade (if we want to upgrade it)

I.2/ Diagram



Error code 200 for success and 404 for non-existant data are taken from the HTTP protocol, because it works well and everyone knows them.

II/ Basic Implementation

II.1/ Server

```c
#include <sys/socket.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>


void handle_request(int connfd);
void *client_handler(void *arg);
void init_files_inventory();
int file_exists(const char *filename);
void send_file(int connfd, const char *filename);

char **FILES = NULL;
int FILES_COUNT = 0;
pthread_mutex_t files_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```c
int main(int argc, char *argv[]) {
  int sfd, newconn;
  struct sockaddr_in hostinfo = {
    .sin_family = AF_INET,
    .sin_addr.s_addr = INADDR_ANY
  };
  socklen_t addrlen = sizeof(hostinfo);
  pthread_t thread_id;

  if (argc < 2) {
        perror("No port specified\n");
        exit(EXIT_FAILURE);
  }

  int port = atoi(argv[1]);

  if (port == 0) {
    perror("Bad Port Specified\n");
    exit(EXIT_FAILURE);
  }

  hostinfo.sin_port = htons(port);

  if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("Unable to init socket\n");
    exit(EXIT_FAILURE);
  }

  int opt = 1;
  // completely optional but useful so i did it (little flex too)
  // reuse the same port even if the server close (the syscall take too much time
and i'm lazy sorry)
  if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0) {
    perror("setsockopt failed\n");
    exit(EXIT_FAILURE);
  }

  if (bind(sfd, (struct sockaddr *) &hostinfo, sizeof (hostinfo)) < 0) {
    perror("Port already in use or reserved\n");
    exit(EXIT_FAILURE);
  }

  if (listen(sfd, 3) < 0) {
    perror("Can't listen on port");
    exit(EXIT_FAILURE);
  }

  init_files_inventory();

  printf("port: %d!!\n", port);
  printf("files: %d\n", FILES_COUNT);

  while (1) {
    if ((newconn = accept(sfd, (struct sockaddr *) &hostinfo, &addrlen)) < 0) {
```

```
      perror("Accept failed\n");
      continue;
    }
    int *threadparam;
    if ((threadparam =malloc(sizeof (int))) == NULL) {
      perror("Unable to allocate memory\n");
      close(newconn);
      continue;
    }
    *threadparam = newconn;

    printf("new connection\n");

    if (pthread_create(&thread_id, NULL, client_handler, threadparam) != 0) {
      perror("Failed to create thread\n");
      close(newconn);
      continue;
    }

    pthread_detach(thread_id);
  }

  close(sfd);
  return 0;
}


void init_files_inventory() {
  DIR *dir;
  struct dirent *entry;
  struct stat st;
  int capacity = 10; // i wanted to do a proper argument when you launch the server
but finally i decided to do it automatcially

  if ((FILES = malloc(capacity * sizeof (char *))) == NULL) {
    perror("Failed to allocate memory for files list\n");
    exit(EXIT_FAILURE);
  }


  if ((dir = opendir("."))== NULL) {
    perror("Failed to open current directory\n");
    exit(EXIT_FAILURE);
  }

  while ((entry = readdir(dir)) != NULL) {
    if (stat(entry->d_name, &st) == 0 && S_ISREG(st.st_mode)) {
      if (FILES_COUNT >= capacity) {
        capacity *= 2;
        FILES = realloc(FILES, capacity * sizeof(char *));
        if (FILES == NULL) {
          perror("Failed to reallocate memory\n");
          exit(EXIT_FAILURE);
        }
      }
```

```
      FILES[FILES_COUNT] = strdup(entry->d_name); //NOTE: strdup supremacy > strcpy
      if (FILES[FILES_COUNT] == NULL) {
        perror("Failed to duplicate filename\n");
        exit(EXIT_FAILURE);
      }
      printf("file: %s\n", FILES[FILES_COUNT]);
      FILES_COUNT++;
    }
  }

  closedir(dir);
}

// i decided to not use <boolean.h> because kinda a waste of time for preoject like
that
int file_exists(const char *filename) {
  pthread_mutex_lock(&files_mutex);
  for (int i =0; i < FILES_COUNT; i++) {
    if (strcmp(FILES[i], filename) == 0) {
      pthread_mutex_unlock(&files_mutex);
      return 1;
    }
  }
  pthread_mutex_unlock(&files_mutex);
  return 0;
}


void handle_request(int connfd) {
  // i could make a linked list but i prefer to believe that people don't name their
files with a whole sentence
  // it's just a proof of concept so it's okay
  char buffer[4096], protocol[16], command[16], version[5], filename[256];
  int n;

  n = read(connfd, buffer, sizeof(buffer) - 1);
  if (n <= 0) {
    return;
  }
  buffer[n] = '\0';

  if (sscanf(buffer, "%s %s %s %s", protocol, version, command, filename) != 4) {
    char *error = "BFT 1.0 ERROR 400 Bad Request\n";
    write(connfd, error, strlen(error));
    return;
  }

  if (strcmp(protocol, "BFT") != 0 || strcmp(command, "FILE") != 0) {
    char *error = "BFT 1.0 ERROR 400 Bad Request\n";
    write(connfd, error, strlen(error));
    return;
  }

  char *clean_filename = filename;
  if (filename[0] == '/') {
    clean_filename = filename + 1; // lazy move but efficient
```

```
  }

  printf("request: %s\n", clean_filename);

  if (!file_exists(clean_filename)) {
    char response[512];
    snprintf(response, sizeof(response), "BFT 1.0 FILE %s 404 No such file\n",
filename);
    write(connfd, response, strlen(response));
    printf("file not found: %s\n", clean_filename);
    return;
  }

  send_file(connfd, clean_filename);
}


void send_file(int connfd, const char *filename) {
  FILE *fp;
  char buffer[4096];
  size_t bytes_read;
  char header[512];
  struct stat st;

  fp = fopen(filename, "rb");
  if (fp == NULL) {
    char response[512];
    snprintf(response, sizeof(response), "BFT 1.0 FILE /%s 404 No such file\n",
filename);
    write(connfd, response, strlen(response));
    return;
  }

  stat(filename, &st);
  snprintf(header, sizeof(header), "BFT 1.0 FILE /%s ", filename);
  write(connfd, header, strlen(header));

  while ((bytes_read = fread(buffer, 1, sizeof(buffer), fp)) > 0) {
    write(connfd, buffer, bytes_read);
  }

  fclose(fp);
  printf("file sent: %s \n", filename);
}


void *client_handler(void *arg) { //my thread
  int connfd = *((int *)arg);
  free(arg);

  handle_request(connfd);

  close(connfd);
  printf("connection closed\n");
```

```
  return NULL;
}
```

II.2/ Client

```
#include <sys/socket.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <string.h>


void request_file(int sockfd, const char *filename);


int main(int argc, char *argv[]) {
  int sockfd;
  struct sockaddr_in servinfo;
  char *host;
  int port;
  char *filename;

  if (argc < 4) {
    fprintf(stderr, "Usage: %s <host> <port> <filename>\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  host = argv[1];
  port = atoi(argv[2]);
  filename = argv[3];

  if (port == 0) {
    perror("Bad Port Specified\n");
    exit(EXIT_FAILURE);
  }

  servinfo.sin_family = AF_INET;
  servinfo.sin_port = htons(port);

  if (inet_pton(AF_INET, host, &servinfo.sin_addr) <= 0) {
    perror("Invalid address\n");
    exit(EXIT_FAILURE);
  }

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("Unable to create socket\n");
    exit(EXIT_FAILURE);
  }

  if (connect(sockfd, (struct sockaddr *)&servinfo, sizeof(servinfo)) < 0) {
    perror("Connection failed\n");
    exit(EXIT_FAILURE);
  }

  printf("Connected to %s:%d\n", host, port);
```

```
  request_file(sockfd, filename);

  close(sockfd);
  return 0;
}


void request_file(int sockfd, const char *filename) {
  char request[512];
  char buffer[4096];
  int n, total_bytes = 0;
  FILE *fp = NULL;
  char *clean_filename;
  int header_parsed = 0;

  if (filename[0] != '/') {
    snprintf(request, sizeof(request), "BFT 1.0 FILE /%s\n", filename);
    clean_filename = strdup(filename);
  } else {
    snprintf(request, sizeof(request), "BFT 1.0 FILE %s\n", filename);
    clean_filename = strdup(filename + 1);
  }

  printf("Requesting: %s", request);

  if (write(sockfd, request, strlen(request)) < 0) {
    perror("Failed to send request\n");
    free(clean_filename);
    return;
  }

  while ((n = read(sockfd, buffer, sizeof(buffer))) > 0) {
    if (!header_parsed) {
      char *data_start = NULL;
      char response_header[512];
      int header_len = 0;

      for (int i = 0; i < n; i++) {
        if (i >= 3 &&
            buffer[i-3] == 'F' && buffer[i-2] == 'I' &&
            buffer[i-1] == 'L' && buffer[i] == 'E') {
          int j = i + 1;
          while (j < n && buffer[j] == ' ') j++;
          while (j < n && buffer[j] != ' ') j++;
          if (j < n && buffer[j] == ' ') {
            j++;
            data_start = buffer + j;
            header_len = j;
            break;
          }
        }
      }

      if (data_start == NULL) {
        buffer[n] = '\0';
```

```
        if (strstr(buffer, "404 No such file") != NULL) {
          printf("Error: %s", buffer);
          free(clean_filename);
          return;
        }
        printf("Error: Invalid response from server\n");
        free(clean_filename);
        return;
      }

      strncpy(response_header, buffer, header_len);
      response_header[header_len] = '\0';

      if (strstr(response_header, "404") != NULL) {
        printf("Error: File not found\n");
        free(clean_filename);
        return;
      }

      printf("Receiving file: %s\n", clean_filename);

      fp = fopen(clean_filename, "wb");
      if (fp == NULL) {
        perror("Failed to create output file\n");
        free(clean_filename);
        return;
      }

      int data_len = n - header_len;
      if (data_len > 0) {
        fwrite(data_start, 1, data_len, fp);
        total_bytes += data_len;
      }

      header_parsed = 1;
    } else {
      fwrite(buffer, 1, n, fp);
      total_bytes += n;
    }
  }

  if (fp != NULL) {
    fclose(fp);
    printf("File saved: %s (%d bytes)\n", clean_filename, total_bytes);
  }

  free(clean_filename);
}
```

III/ How to make it work

In this directory, type

```
make test
```

It will automaically create folders and binaries (with debug flags, use make all instead if you don’t want it)

III.1/ Server part

Then go on server test folder

```
cd tests/server_test
```

Create an exemple file

```
echo "Salut!" > hello.txt
```

and finally launch the server

```
./server 2020
```

There you go for the server

III.2/ Client part

In another shell go to the client test folder

then launch the client

```
./client 127.0.0.1 2020 hello.txt
```