Министерство науки и образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
"Московский институт электронной техники"
(МИЭТ)

Отчет по лабораторной работе № 6

Операционные системы

Выполнил: студент ПМ - 31

Мартынова Мария Олеговна

2023 г.

Задание 1

В предыдущей лабораторной работе вы распараллеливали вычисление факториала по модулю с помощью потоков. В этой работе вы пойдете еще дальше: вы распараллелите эту работу еще и между серверами.

Необходимо закончить client.c и server.c:

Клиент в качетсве аргументов командной сроки получает k, mod, servers, где k это факториал, который необходимо вычислить (k! % mod), servers это путь до файла, который содержит сервера (ip:port), между которыми клиент будет распараллеливать соединения.

Сервер получает от клиента "кусок" своих вычислений и mod, в ответ отсылает клиенту результат этих вычислений.

Client.c

lab6/src/client.c

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>

#include <errno.h>
#include <getopt.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <sys/socket.h>
#include <sys/types.h>

#include "multmodule.h"

struct Server {
  char ip[255]; // адрес компьютера
  int port; // индентификатор сокета
};

struct Args{
  struct Server s;
  int* result;
  int begin;
  int end;
  int mod;
};

pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;

bool ConvertStringToUI64(const char *str, uint64_t *val) {//конвертация из строки в число
  char *end = NULL;
  unsigned long long i = strtoull(str, &end, 10);
  if (errno == ERANGE) {
    fprintf(stderr, "Out of uint64_t range: %s\n", str);
    return false;
  }

  if (errno != 0)
    return false;

  *val = i;
  return true;
}

void ConnectServer(void *_args) { //функция для тредов
    struct Args *args = (struct Args *)_args;

    struct hostent *hostname = gethostbyname(args->s.ip);
    if (hostname == NULL) {
      fprintf(stderr, "gethostbyname failed with %s\n", args->s.ip);
      exit(1);
    }
```

```
56
57      struct sockaddr_in server; //содержится адрес сокета и его принцип передачи данных
58      server.sin_family = AF_INET;// принцип передачи данных (через файл или объект ядра)
59      server.sin_port = htons(args->s.port); // перестраивание данных под свою архитектуру
60      server.sin_addr.s_addr = *((unsigned long *)hostname->h_addr);// записывание айпи-адреса
61
62      int sck = socket(AF_INET, SOCK_STREAM, 0); // создание сокета
63      if (sck < 0) {
64        fprintf(stderr, "Socket creation failed!\n");
65        exit(1);
66      }
67
68      if (connect(sck, (struct sockaddr *)&server, sizeof(server)) < 0) { //попытка соединения с сервером
69        fprintf(stderr, "Connection failed\n");
70        exit(1);
71      }
72
73      uint64_t begin = args->begin;
74      uint64_t end = args->end;
75
76      char task[sizeof(uint64_t) * 3];  // строка, содержащая begin, end, mod
77      memcpy(task, &begin, sizeof(uint64_t));
78      memcpy(task + sizeof(uint64_t), &end, sizeof(uint64_t));
79      memcpy(task + 2 * sizeof(uint64_t), &args->mod, sizeof(uint64_t));
80
81      if (send(sck, task, sizeof(task), 0) < 0) {// запись
82        fprintf(stderr, "Send failed\n");
83        exit(1);
84      }
85
86      char response[sizeof(uint64_t)]; // создание массива, в котором будет содержаться ответ
87      if (recv(sck, response, sizeof(response), 0) < 0) {// считываение из массива ответ
88        fprintf(stderr, "Recieve failed\n");
89        exit(1);
90      }
91
92      uint64_t answer = 0;
93      memcpy(&answer, response, sizeof(uint64_t));
94      printf("answer: %llu\n", answer);
95
96      pthread_mutex_lock(&mut);
97      *args->result = (*args->result * answer) % args->mod;
98      pthread_mutex_unlock(&mut);
99
100     close(sck);
101 }
102
103 int main(int argc, char **argv) {
104   uint64_t k = -1;
105   uint64_t mod = -1;
106   char servers[255] = {'\0'};
107
108   while (true) {
109     int current_optind = optind ? optind : 1;
```

```
110 |
111 ▼   static struct option options[] = {{"k", required_argument, 0, 0},
112                                        {"mod", required_argument, 0, 0},
113                                        {"servers", required_argument, 0, 0},
114                                        {0, 0, 0, 0}};
115
116     int option_index = 0;
117     int c = getopt_long(argc, argv, "", options, &option_index);
118
119     if (c == -1)
120       break;
121
122 ▼   switch (c) {
123 ▼   case 0: {
124 ▼     switch (option_index) {
125       case 0:
126         if(!ConvertStringToUI64(optarg, &k))
127           exit(1);
128         break;
129       case 1:
130         if(!ConvertStringToUI64(optarg, &mod))
131           exit(1);
132         break;
133       case 2:
134         memcpy(servers, optarg, strlen(optarg));
135         break;
136       default:
137         printf("Index %d is out of options\n", option_index);
138       }
139     } break;
140
141     case '?':
142       printf("Arguments error\n");
143       break;
144     default:
145       fprintf(stderr, "getopt returned character code 0%o?\n", c);
146     }
147   }
148
149 ▼   if (k == -1 || mod == -1 || !strlen(servers)) {
150     fprintf(stderr, "Using: %s --k 1000 --mod 5 --servers /path/to/file\n",
151             argv[0]);
152     return 1;
153   }
154
155   unsigned int servers_num; // количество серверов, к которым будем обращаться
156   FILE* f = fopen(servers, "r");
157   if(!f) exit(1);
158   fscanf(f, "%d ", &servers_num); // считываем количество серверов (в txt количество серверов и их порты)
159   struct Server *to = malloc(sizeof(struct Server) * servers_num);// указатель на массив структур, содержащие адреса серверов
160
161 ▼   for(int i = 0; i < servers_num; i++) { //заполнение массива всех серверов
162     fscanf(f, "%d ", &to[i].port);
163     memcpy(to[i].ip, "127.0.0.1", sizeof("127.0.0.1")); // копирование строки в переменную, содержащую айпи
164   }
165   fclose(f);
166
```

```
166
167   pthread_t threads[servers_num];
168   struct Args args[servers_num];
169   int factorial = 1;
170
171 ▼   for(int i = 0; i < servers_num; i++) { // то что передается на каждый тред
172     args[i].s = to[i]; //адрес для связи с сервером
173     args[i].result = &factorial; // адрес, где хранится результат работы серверов
174     args[i].begin = i * k / servers_num + 1;
175     args[i].end = i == servers_num - 1 ? k : (i + 1) * k / servers_num;
176     args[i].mod = mod;
177 ▼   if(pthread_create(&threads[i], NULL, ConnectServer, (void *)&args[i])) {
178       printf("Error: pthread_create failed!\n");
179       exit(1);
180     }
181   }
182
183   for(int i = 0; i < servers_num; i++) // дожидается завершения тредов
184     pthread_join(threads[i], NULL);
185   free(to);
186
187   printf("Result: %d\n", factorial);
188
189   return 0;
190 }
```

Server.c

lab6/src/server.c

```c
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <getopt.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <sys/socket.h>
#include <sys/types.h>

#include "pthread.h"
#include "multmodule.h"

struct FactorialArgs {
  uint64_t begin;
  uint64_t end;
  uint64_t mod;
};

uint64_t Factorial(const struct FactorialArgs *args) {
  uint64_t ans = 1;

  int i;
  for(i = args->begin; i <= args->end; i++)
    ans = (ans * i) % args->mod;

  return ans;
}

void *ThreadFactorial(void *args) {// функция для тредов
  struct FactorialArgs *fargs = (struct FactorialArgs *)args;
  return (void *)(uint64_t *)Factorial(fargs); // возвращение результата факториала
}

int main(int argc, char **argv) {
  int tnum = -1; // количество тредов на сервере
  int port = -1;

  while (true) {
    int current_optind = optind ? optind : 1;

    static struct option options[] = {{"port", required_argument, 0, 0},
                                      {"tnum", required_argument, 0, 0},
                                      {0, 0, 0, 0}};

    int option_index = 0;
    int c = getopt_long(argc, argv, "", options, &option_index);

    if (c == -1)
      break;
```

```
54
55      switch (c) {
56      case 0: {
57        switch (option_index) {
58        case 0:
59          port = atoi(optarg);
60          if (port < 0)
61              exit(1);
62          break;
63        case 1:
64          tnum = atoi(optarg);
65          if (tnum < 0)
66              exit(1);
67          break;
68        default:
69          printf("Index %d is out of options\n", option_index);
70        }
71      } break;
72
73      case '?':
74        printf("Unknown argument\n");
75        break;
76      default:
77        fprintf(stderr, "getopt returned character code 0%o?\n", c);
78      }
79    }
80
81    if (port == -1 || tnum == -1) {
82      fprintf(stderr, "Using: %s --port 20001 --tnum 4\n", argv[0]);
83      return 1;
84    }
85
86    int server_fd = socket(AF_INET, SOCK_STREAM, 0); // создание сокета
87    if (server_fd < 0) {
88      fprintf(stderr, "Can not create server socket!");
89      return 1;
90    }
91
92    struct sockaddr_in server; //содержится адрес сокета и его принцип передачи данных
93    server.sin_family = AF_INET;
94    server.sin_port = htons((uint16_t)port);
95    server.sin_addr.s_addr = htonl(INADDR_ANY); //будет работать с любым айпи адресом
96
97    int opt_val = 1;
98    setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt_val, sizeof(opt_val));// убирание "залипание" порта
99
100   int err = bind(server_fd, (struct sockaddr *)&server, sizeof(server)); //связь слушающего сокета с параметрами
101   if (err < 0) {
102     fprintf(stderr, "Can not bind to socket!");
103     return 1;
104   }
105
```

```
106   err = listen(server_fd, 128);
107 ▼ if (err < 0) {
108     fprintf(stderr, "Could not listen on socket\n");
109     return 1;
110   }
111
112   printf("Server listening at %d\n", port);
113
114 ▼ while (true) {
115     struct sockaddr_in client;
116     socklen_t client_len = sizeof(client);
117     int client_fd = accept(server_fd, (struct sockaddr *)&client, &client_len);
118
119 ▼   if (client_fd < 0) {
120       fprintf(stderr, "Could not establish new connection\n");
121       continue;
122     }
123
124 ▼   while (true) {
125       unsigned int buffer_size = sizeof(uint64_t) * 3;
126       char from_client[buffer_size]; // буфер, в который будет записан таск
127       int read = recv(client_fd, from_client, buffer_size, 0); // считываем то, что прислал клиент (таск)
128
129       if (!read)
130         break;
131 ▼     if (read < 0) {
132         fprintf(stderr, "Client read failed\n");
133         break;
134       }
135 ▼     if (read < buffer_size) {
136         fprintf(stderr, "Client send wrong data format\n");
137         break;
138       }
139
140       pthread_t threads[tnum]; //массив, в котором будут храниться айди тредов [количесвто тредов на сервере]
141
142       uint64_t begin = 0;
143       uint64_t end = 0;
144       uint64_t mod = 0;
145       memcpy(&begin, from_client, sizeof(uint64_t));
146       memcpy(&end, from_client + sizeof(uint64_t), sizeof(uint64_t));
147       memcpy(&mod, from_client + 2 * sizeof(uint64_t), sizeof(uint64_t));
148
149       fprintf(stdout, "Receive: %llu %llu %llu\n", begin, end, mod);
150
151       struct FactorialArgs args[tnum];//создаем аргументы для тредов
152 ▼     for (uint32_t i = 0; i < tnum; i++) {
153         args[i].begin = begin + i*(end - begin + 1)/tnum;
154         if (i != 0) args[i].begin++;
155         args[i].end = i == tnum - 1 ? end : begin + (i + 1)*(end - begin + 1)/tnum;
156         args[i].mod = mod;
157
```

```
157
158         if (pthread_create(&threads[i], NULL, ThreadFactorial,
159 ▼                          (void *)&args[i])) {
160           printf("Error: pthread_create failed!\n");
161           return 1;
162         }
163       }
164
165       uint64_t total = 1;
166 ▼     for (uint32_t i = 0; i < tnum; i++) {
167         uint64_t result = 0;
168         pthread_join(threads[i], (void **)&result);
169         total = MultModulo(total, result, mod);
170       }
171
172       printf("Total: %llu\n", total);
173
174       char buffer[sizeof(total)];
175       memcpy(buffer, &total, sizeof(total));
176       err = send(client_fd, buffer, sizeof(total), 0);
177 ▼     if (err < 0) {
178         fprintf(stderr, "Can't send data to client\n");
179         break;
180       }
181     }
182
183     shutdown(client_fd, SHUT_RDWR); //прекращение чтения и записи сокета
184     close(client_fd); // закрытие сокета
185   }
186
187   return 0;
188 }
189
```

Задание 2

Создать makefile для программ клиента и сервер.

lab6/src/makefile ×

```
1  CC=gcc
2  CFLAGS=-I.
3
4  all :
5    $(CC) -pthread -o client client.c -L. -lmulm $(CFLAGS)
6    $(CC) -pthread -o server server.c -L. -lmulm $(CFLAGS)
7
8  rm :
9    rm client server
```

Задание 3

Найти дублирующийся код в двух приложениях и вынести его в библиотеку.

Добавить изменения в makefile.

lab6/src/multmodule.c

```c
#include <stdint.h>
#include "multmodule.h"
uint64_t MultModulo(uint64_t a, uint64_t b, uint64_t mod) {
  uint64_t result = 0;
  a = a % mod;
  while (b > 0) {
    if (b % 2 == 1)
      result = (result + a) % mod;
    a = (a * 2) % mod;
    b /= 2;
  }

  return result % mod;
}
```