上海海事大学

Linux操作系统大作业

学 院： 信息工程学院

专 业： 网络工程

班 级： 网络191

姓 名： 徐艺芸

指导老师： 宋安军

完成时间： 2021年 12月

基于epoll模型的高并发多人网络聊天室代码清单

* 服务端：

|  |
| --- |
| 头文件 |
| #include<stdio.h>  #include<string.h>  #include<stdlib.h>  #include<unistd.h>  #include<fcntl.h>  #include<string.h>  #include<pthread.h>  #include<sys/epoll.h>  #include<ctype.h>  #include<sys/socket.h>  #include<arpa/inet.h>  #define Port 7777  #define MAX\_EVENTS 1024  typedef void (\*call\_back)(int,void\*); |

|  |
| --- |
| 登陆用户信息 |
| **typedef** struct user  **{**  char usr\_id**[**8**];** // 用户ID 五位 UID  char usr\_name**[**256**];** // 用户名  char usr\_key**[**40**];** // 用户密码  int st; // 是否在线 0---- 离线 1---- 在线  **}**user\_msg**;**  user\_msg Users**[**MAX\_EVENTS**];** // 已注册的所有用户信息  int user\_num**;** // 已注册用户信息的数量 |

|  |
| --- |
| 描述监听的文件描述符的相关信息的结构体 |
| typedef struct myevent\_s  {  int fd; // 监听的文件描述符  int events; // 对应监听的事件 EPOLLIN / EPOLLOUT  call\_back fun; // 回调函数  void \*arg; // 上面回调函数的参3  int status; // 是否在监听红黑树上, 1 --- 在, 0 --- 不在  char buf[BUFSIZ]; // 读写缓冲区  int len; // 本次从客户端读入缓冲区数据的长度  long long last\_active\_time; // 该文件描述符最后在监听红黑树上的活跃时间  user\_msg um; // 用户登陆的信息  int log\_step;  // 标记用户位于登陆的操作 0-- 未登陆 1 --- 输入账号 2 ---- 输入密码 3----- 成功登陆 4 --- 注册用户名 5 ------ 输入注册的密码 6 ------- 再次输入密码验证  }myevent\_s; |

|  |
| --- |
| 数组模拟双链表写法, 链表保存当前在线的用户 |
| int g\_efd; // 监听红黑树的树根  myevent\_s g\_events[MAX\_EVENTS + 1]; // 用于保存每个文件描述符信息的结构体的数组  #define ONLINE\_MAX 100000  int online\_fd[ONLINE\_MAX], l[ONLINE\_MAX], r[ONLINE\_MAX], fd\_pos[ONLINE\_MAX], idx, online\_num;  // 链表进行初始化  void list\_init()  {  r[0] = 1, l[1] = 0; // 下标0 表示链表左端点, 下标1表示链表右端点  idx = 2;  for(int i = 3; i < ONLINE\_MAX; ++i) fd\_pos[i] = -1;  } |

|  |
| --- |
| 往链表中插入一个在线用户 |
| void list\_push(int fd)  {  fd\_pos[fd] = idx;  online\_fd[idx] = fd;  r[idx] = r[0], l[idx] = 0;  l[r[0]] = idx, r[0] = idx++;  online\_num++; // 在线人数++  } |

|  |
| --- |
| 从在线链表中删除一个用户的文件描述符 |
| void list\_del(int fd) // 在线链表中删除fd的文件描述符  {  int k = fd\_pos[fd];  l[r[k]] = l[k];  r[l[k]] = r[k];  online\_num--; // 在线人数--  } |

|  |
| --- |
| 函数声明 |
| void cb\_read(int, void\*); // 服务器端读事件  void cb\_write(int, void\*); // 向在线用户发送数据写事件  void logout(int cfd, void \*); // 用户登出事件  void login\_menu(int cfd , void\* arg); // 登陆界面读事件  void login(int, void\*); // 输入账号登陆  void register\_id(int, void\*); // 注册新账号  void get\_uid(myevent\_s\*); // 获取一个未注册的UID  void load\_usermsg(); // 从文件加载已经注册过的用户信息  char ms1[] = "与服务器建立连接, 开始进行数据通信 ------ [OK]\n"  " epoll服务器聊天室测试版 \n"  " (1)匿名聊天 (2)登陆 (3) 注册 \n"  ">>> "; |

|  |
| --- |
| 出错处理函数 |
| void sys\_error(const char \*str)  {  perror(str);  exit(1);  } |

|  |
| --- |
| 重新设置监听事件 |
| void event\_set(myevent\_s \*ev, int fd, int events, call\_back fun, void \*arg3)  {  ev->fd = fd ;  ev->events = events;  ev->fun = fun;  ev->arg = arg3;  } |

|  |
| --- |
| 添加监听事件到树上 |
| void event\_add(int epfd, myevent\_s \*ev)  {  struct epoll\_event tep;  tep.data.ptr = ev;  tep.events = ev->events;  if(epoll\_ctl(epfd, EPOLL\_CTL\_ADD, ev->fd, &tep) == -1)  printf("fail: epoll\_ctl add fd: %d, events is %d\n",ev->fd, ev->events);  else  ev->status = 1, ev->last\_active\_time = time(NULL);  } |

|  |
| --- |
| 将事件从监听红黑树上摘除 |
| void event\_del(int epfd, myevent\_s \*ev)  {  ev->status = 0;  epoll\_ctl(epfd, EPOLL\_CTL\_DEL, ev->fd, NULL);  } |

|  |
| --- |
| 关闭与客户端通信的文件描述符 |
| void close\_cfd(int cfd, myevent\_s \*ev)  {  char str[BUFSIZ];  event\_del(g\_efd, ev);  close(cfd);  sprintf(str, "the client fd: %d is close\n", ev->fd);  write(STDOUT\_FILENO, str, strlen(str));  return;  } |

|  |
| --- |
| 监听新的客户端建立连接 |
| void cb\_accept(int lfd, void \* arg)  {  struct sockaddr\_in client\_addr;  socklen\_t client\_addr\_len = sizeof client\_addr;  int cfd = accept(lfd, (struct sockaddr\*)&client\_addr, &client\_addr\_len);  if(cfd == -1) sys\_error("accept error");  int i = 0;  for(i = 0; i < MAX\_EVENTS && g\_events[i].status != 0; ++i);  if(i == MAX\_EVENTS)  {  printf("the client num is max\n");  return;  }  struct myevent\_s \*ev = &g\_events[i];  int flag = fcntl(cfd, F\_GETFL); // 设置为非阻塞状态  flag |= O\_NONBLOCK;  fcntl(cfd, F\_SETFL, flag);  event\_set(ev, cfd, EPOLLIN | EPOLLET, login\_menu, ev); // 建立新连接后,事件设定为登陆界面程序  event\_add(g\_efd, ev);  printf("the new client ip is %s, the client port is %d\n", // 服务器端打印客户端的地址信息  inet\_ntop(AF\_INET, &client\_addr.sin\_addr.s\_addr, ev->buf, sizeof ev->buf),  ntohs(client\_addr.sin\_port));  write(cfd, ms1, sizeof ms1);  } |

|  |
| --- |
| 登陆界面 |
| void login\_menu(int cfd , void\* arg)  {  myevent\_s \*ev = (myevent\_s\*) arg;  char \*buf = ev->buf;  int ret = read(cfd, buf, BUFSIZ);  if(ret <= 0)  {  close\_cfd(cfd, ev);  return;  }  if(buf[0] == '1') // 匿名用户登陆  {  sprintf(ev->um.usr\_name, "匿名用户 %ld", time(NULL)) ; // 设置匿名登陆的用户名  strcpy(ev->um.usr\_id, "00000"); // 所有匿名用户的账号为00000  // 加入到聊天回调然后监听  ev->log\_step = 3; // 表示为已登录状态  list\_push(cfd); // 加入当前的在线列表  sprintf(buf,">用户: %s 已登录,当前在线人数为 %d\n\n>>>", ev->um.usr\_name, online\_num);  ev->len = strlen(buf);  char s[] = "----------------------epoll聊天室测试版-----------------\n";  write(cfd, s, sizeof s);  write(cfd, buf, ev->len);  // 重新设定监听事件为写, 写的内容为向当前在线用户发送XXX已登录的信息  event\_del(g\_efd, ev);  event\_set(ev, cfd, EPOLLOUT , cb\_write, ev);  event\_add(g\_efd, ev);  }  else if(buf[0] == '2') // 账号UID登陆  {  ev->log\_step = 1;  strcpy(buf, "请输入登陆的UID:");  write(cfd, buf, strlen(buf));  // 将事件设定为登陆的回调  event\_del(g\_efd, ev);  event\_set(ev, cfd, EPOLLIN | EPOLLET , login, ev);  event\_add(g\_efd, ev);  }  else // 注册  {  strcpy(buf, "注册账号\n###请输入注册的用户名(中文/英文, 注意不要包含特殊符号): ");  write(cfd, buf, strlen(buf));  ev->log\_step = 4; // 标记为进行注册状态的准备输入注册的用户名  // 将事件监听设置为注册的回调  event\_del(g\_efd, ev);  event\_set(ev, cfd, EPOLLIN | EPOLLET, register\_id, ev);  event\_add(g\_efd, ev);  }  } |

|  |
| --- |
| 输入账号UID进行登陆 |
| void login(int cfd, void \*arg)  {  myevent\_s \*ev = (myevent\_s\*) arg;  char \*buf = ev->buf;  int ret = read(cfd, buf, BUFSIZ);  if(ret <= 0)  {  close\_cfd(cfd, ev);  return;  }  buf[ret - 1] = '\0';  if(1 == ev->log\_step) // 读取用户输入用户名  {  int id = atoi(buf);  strcpy(ev->um.usr\_id, buf);  char s[100];  if(id > user\_num || id <= 0)  {  sprintf(s, "!用户UID:%s 不存在\n请重新输入账号UID:", buf);  write(cfd, s, strlen(s));  return;  }  if(Users[id].st)  {  sprintf(s, "!用户UID:%s 已登陆\n请重新输入账号UID:", buf);  write(cfd, s, strlen(s));  return;  }  strcpy(buf, "请输入密码:");  write(cfd, buf, strlen(buf));  }  else if(2 == ev->log\_step) // 输入用户密码  {  int id = atoi(ev->um.usr\_id);  strcpy(ev->um.usr\_key, buf);  if(!strcmp(buf, Users[id].usr\_key))  {  strcpy(ev->um.usr\_name, Users[id].usr\_name);  list\_push(cfd); // 将当前的cfd添加进在线列表中  Users[id].st = 1;  sprintf(buf,">用户: %已登录,当前在线人数为 %d\n\n>>>", ev->um.usr\_name, online\_num);  ev->len = strlen(buf);  char s[] = "----------------------epoll聊天室测试版--------------------\n";  write(cfd, s, sizeof s);  write(cfd, buf, ev->len);  // 设定为写事件, 向当前在线用户发送XXX用户已登陆的信息  event\_del(g\_efd, ev);  event\_set(ev, cfd, EPOLLOUT , cb\_write, ev);  event\_add(g\_efd, ev);  }  else  {  strcpy(buf, "密码错误, 请重新输入密码:");  write(cfd, buf, strlen(buf));  return;  }  }  ev->log\_step++;  } |

|  |
| --- |
| 注册账号 |
| void register\_id(int cfd, void \*arg)  {  myevent\_s \*ev = (myevent\_s\*) arg;  char \*buf = ev->buf;  int ret = read(cfd, buf, BUFSIZ);  if(ret <= 0)  {  close\_cfd(cfd, ev);  return;  }  buf[ret - 1] = '\0';  if(4 == ev->log\_step) // 输入注册的用户名  {  strcpy(ev->um.usr\_name, buf);  strcpy(buf, "请设定账号的密码: ");  write(cfd, buf, strlen(buf));  }  else if(5 == ev->log\_step) // 输入用户密码  {  strcpy(ev->um.usr\_key, buf);  strcpy(buf, "请再次输入密码: ");  write(cfd, buf, strlen(buf));  }  else if(6 == ev->log\_step) // 验证两次用户的密码  {  if(strcmp(ev->um.usr\_key, buf))  {  strcpy(buf, "两次密码输入不一致, 请重新输入:");  write(cfd, buf, strlen(buf));  return;  }  get\_uid(ev);  sprintf(buf, "注册成功, 你的账号uid: %s 用户名为%s, 现在重新返回登陆界面 \n\n", ev->um.usr\_id,ev->um.usr\_name);  user\_num++;  strcpy(Users[user\_num].usr\_id, ev->um.usr\_id);  strcpy(Users[user\_num].usr\_name, ev->um.usr\_name);  strcpy(Users[user\_num].usr\_key, ev->um.usr\_key);  ev->log\_step = 0;  write(cfd, buf, strlen(buf));  write(cfd, ms1, sizeof ms1);  // 注册完账号, 重新返回登陆界面的程序进行监听  event\_del(g\_efd, ev);  event\_set(ev, cfd, EPOLLIN | EPOLLET, login\_menu, ev);  event\_add(g\_efd, ev);  return;  }  ev->log\_step++;  } |

|  |
| --- |
| 获取一个未注册的uid |
| void get\_uid(myevent\_s \*ev)  {  char str[10];  user\_msg \*p = &ev->um;  sprintf(str, "%05d", user\_num + 1);  strcpy(ev->um.usr\_id, str);  FILE \*fp = fopen("/mnt/c/socket/user\_msg", "a+");  if(fp == NULL)  {  write(ev->fd, "error\n", 6);  fprintf(stderr, "get\_uid open file error\n");  }  fprintf(fp,"%s %s %s\n", p->usr\_id, p->usr\_name, p->usr\_key); // 将新注册的用户信息写入保存用户信息的文件中  fflush(fp); // 刷新缓冲区, 将内容写入到文件中  } |

|  |
| --- |
| 写事件 ---> 向当前在线用户发送信息 |
| void cb\_write(int cfd, void \*arg)  {  char str[BUFSIZ];  myevent\_s \*ev = (myevent\_s\*)arg;  if(ev->len <= 0)  {  logout(cfd, ev);  close\_cfd(cfd, ev);  return;  }  for(int i = r[0]; i != 1; i = r[i]) // 遍历当前的在线链表, 向在线用户发送  {  if(online\_fd[i] == cfd) continue;// 发送数据给服务器的客户端一方并不需要发送  write(online\_fd[i], ev->buf, ev->len);  }  if(ev->log\_step == 3) write(cfd, "\n>>>", 4); // 界面的优化,与主要逻辑无关  // 执行完一次事件之后--> 从树上摘下 ---> 重新设定要监听事件 ---> 重新挂上树监听  event\_del(g\_efd, ev);  event\_set(ev, cfd, EPOLLIN | EPOLLET, cb\_read, ev);  event\_add(g\_efd, ev);  } |

|  |
| --- |
| 读事件 -----> 服务器接收的客户端发来的信息 |
| void cb\_read(int cfd, void \*arg)  {  char str[BUFSIZ],str2[1024];  myevent\_s \*ev = (myevent\_s \*) arg;  int ret = read(cfd, str, sizeof str);  if(ret <= 0)  {  logout(cfd, ev);  close\_cfd(cfd, ev);  return;  }  str[ret] = '\0';  sprintf(str2, "from client fd: %d receive data is :", cfd);  if(ret > 0) write(STDOUT\_FILENO, str2, strlen(str2));  write(STDOUT\_FILENO, str, ret); // 将客户端发来的数据在服务器端进行打印  sprintf(ev->buf, "(%s):%s\n>>>", ev->um.usr\_name, str); // 格式化客户端发来的数据 --- 数据处理  ev->len = strlen(ev->buf);  // 此时服务器端接收客户端发来的数据并进行数据,然后发送给其他的在线用户,故此时事件要重新设定为写事件  event\_del(g\_efd, ev);  event\_set(ev, cfd, EPOLLOUT, cb\_write, ev);  event\_add(g\_efd, ev);  } |

|  |
| --- |
| 登出操作 ---> 必须是登陆上之后进行登出才调用 |
| void logout(int cfd, void \*arg)  {  myevent\_s \*ev = (myevent\_s\*)arg;  char str[1024];  list\_del(cfd); // 从在线列表中删除  ev->log\_step = 0; // 标记为登出  Users[atoi(ev->um.usr\_id)].st = 0; // 用户信息中将其标记为离线状态  sprintf(str, "已退出聊天室, 当前在线人数为%d\n", online\_num);  sprintf(ev->buf, "(%s) %s\n>>>", ev->um.usr\_name, str);  ev->len = strlen(ev->buf);  cb\_write(cfd, ev); // 手动调用向其他用户发送XXX用户登出的信息  } |

|  |
| --- |
| 将保存用户信息的文件载入程序 |
| void load\_usermsg() {  FILE \*fp = fopen("/mnt/c/socket/user\_msg", "r"); // 打开保存已注册用户信息的文件  if(fp == NULL) sys\_error("load error");  while(!feof(fp))  {  user\_num++;  fscanf(fp,"%s %s %s", Users[user\_num].usr\_id, Users[user\_num].usr\_name, Users[user\_num].usr\_key); // 将文件的内容加载到保存用户信息的结构体数据中  Users[user\_num].st = 0; // 初始化所有用户为离线状态  }  user\_num--;  /\* for(int i = 1; i <= user\_num; ++i) \*/  /\* fprintf(stderr,"uid:%s,name:%s,key:%s\n", Users[i].usr\_id, Users[i].usr\_name, Users[i].usr\_key); \*/  } |

|  |
| --- |
| 主函数 |
| int main(int argc, char \*argv[])  {  list\_init(); // 初始化链表  load\_usermsg(); // 加载已注册用户的信息  int lfd = socket(AF\_INET, SOCK\_STREAM, 0); // 创建监听套接字  if(lfd == -1) sys\_error("socket error");  // 设置端口复用  int opt = 1;  if(setsockopt(lfd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof opt) == -1) sys\_error("setsockopt error");  // 绑定客户端的地址ip和端口号 , 设置监听上限  struct sockaddr\_in server\_addr;  server\_addr.sin\_family = AF\_INET;  server\_addr.sin\_port = htons(Port);  server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  if(bind(lfd, (struct sockaddr\*)&server\_addr, sizeof server\_addr) == -1) sys\_error("bind error");  if(listen(lfd, 128) == -1) sys\_error("listen error");  // 创建epoll的监听树根  g\_efd = epoll\_create(MAX\_EVENTS + 1);  if(g\_efd == -1) sys\_error("epoll\_create error");  for(int i = 0; i < MAX\_EVENTS; ++i) g\_events[i].status = 0; // 将数组各个位置标记为空闲状态  myevent\_s \*lev = &g\_events[MAX\_EVENTS];  // 将lfd的监听事件设定为与客户端建立连接  event\_set(lev, lfd, EPOLLIN, cb\_accept, lev);  event\_add(g\_efd, lev);  struct epoll\_event eps[MAX\_EVENTS + 1];  int check\_active = 0;  char t[] = "-----------------------server start-----------------------[ok]\n";  write(STDOUT\_FILENO, t, sizeof t);  while(1)  {  // 若客户端经过一段时间没有与服务器进行数据通信, 主动关闭cfd  long long now\_time = time(NULL);  for(int i = 0 ; i < 100; ++i, check\_active++)  {  if(check\_active == MAX\_EVENTS) check\_active = 0;  if(g\_events[check\_active].status != 1) continue;  if(now\_time - g\_events[check\_active].last\_active\_time >= 600)  {  char \*buf = g\_events[check\_active].buf;  sprintf(buf,"-----------------太长时间未操作,已与服务器断开连接--------------------\n");  write(g\_events[check\_active].fd, buf, strlen(buf));  if(g\_events[check\_active].log\_step == 3) logout(g\_events[check\_active].fd, &g\_events[check\_active]);  close\_cfd(g\_events[check\_active].fd, &g\_events[check\_active]);  }  }  int num = epoll\_wait(g\_efd, eps, MAX\_EVENTS + 1, 500);  for(int i = 0; i < num; ++i)  {  myevent\_s \*ev = (myevent\_s\*)eps[i].data.ptr;  ev->fun(ev->fd, ev->arg);  }  }  close(lfd);  return 0;  } |

* 客户端代码

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<unistd.h>  #include<fcntl.h>  #include<string.h>  #include<pthread.h>  #include<sys/socket.h>  #include<arpa/inet.h>  void sys\_error(const char \*str)  {  perror(str);  exit(1);  }  int main(int argc, char \*argv[])  {  char buf[BUFSIZ];    int cfd = socket(AF\_INET,SOCK\_STREAM, 0);  struct sockaddr\_in server\_addr;  server\_addr.sin\_family = AF\_INET;  server\_addr.sin\_port = htons(7777);  inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr.s\_addr);  int ret = connect(cfd, (struct sockaddr\*)&server\_addr,sizeof server\_addr);  if(ret == -1 ) sys\_error("connet error");  int flag = fcntl(STDIN\_FILENO, F\_GETFL);  flag |= O\_NONBLOCK;  fcntl(STDIN\_FILENO, F\_SETFL, flag);  flag = fcntl(cfd, F\_GETFL);  flag |= O\_NONBLOCK;  fcntl(cfd, F\_SETFL, flag);  while(1)  {  ret = read(STDIN\_FILENO, buf, sizeof buf);  if(ret > 0) write(cfd, buf, ret);  if(!strncmp(buf,"exit",4)) break;  int ret = read(cfd, buf, sizeof buf);  if(ret > 0) write(STDOUT\_FILENO, buf, ret);  }  close(cfd);  return 0;  } |