多并发服务器

```c
int main(void)
 {
         listen_fd = socket(AF_INET,SOCK_STREAM,0);
         printf("listen fd =%d\n",listen_fd);
         serverTe_addr.sin_family = AF_INET;
         serverTe_addr.sin_port = htons(server_port);
         serverTe_addr.sin_addr.s_addr = INADDR_ANY;
         bind(listen_fd,(struct sockaddr *)&serverTe_addr,sizeof(serverTe_addr));
         listen(listen_fd,20);

         //select
         fd_set read_fd,all_fd,rset_fd;//all_fd用来暂存fd的集合
         Max_fd = listen_fd;//记录当前最大的fd
         FD_ZERO(&all_fd);//fd集合中表示fd的位（bit）全部置0
         FD_SET(listen_fd,&all_fd);//将listen_fd置于all_fd中

         while(1)
          {
          read_fd = all_fd;
         ready_numFd = select(Max_fd+1,&read_fd,NULL,NULL,NULL);
          printf("select success\n");
          if(ready_numFd < 0)
          {
              perror("select err:");
             exit(1);
          }
          printf("ready_numFd = %d\n",ready_numFd);
          if(FD_ISSET(listen_fd,&read_fd))//返回值为1说明listen_fd监听到了连接,这里用read_f
          {
          //printf("0000000000000\n");
          server_addr_len = sizeof(server_addr);
          connect_fd = accept(listen_fd,(struct sockaddr *)&server_addr,&server_addr_len
          printf("connect success\n");
          read_n = read(connect_fd,buf,sizeof(buf));for(i = 0; i< read_n; i++)
          {
              buf[i] = toupper(buf[i]);
          }
          write(connect_fd,buf,read_n);
```

```
            write(STDOUT_FILENO,buf,read_n);
            close(connect_fd);

```

epoll

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/epoll.h>
#include <unistd.h>
#include <sys/types.h>

#define Port 6669
#define IPADDRESS  "127.0.0.1"
#define LISTEN_NUM 30
#define FDSIZE 1000
#define EPOLLEVENTS 100
static int socket_bind(const char IP,int PORT);
static int add_event(int epoll_fd,int fd,int state);
static void do_epoll(int listenfd);
static void handle_event(int epoll_fd,int num,struct epoll_event * events,int listenfd);
static void hand_accept(int epollfd,int listenfd);
static int add_event(int epoll_fd,int fd,int state);

int socket_bind(const char IP,int PORT)
{
    struct sockaddr_in server_addr;
    int lfd;
    int ret;
    lfd = socket(AF_INET,SOCK_STREAM,0);
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = INADDR_ANY;
    ret = bind(lfd,(struct sockaddr*)&server_addr,sizeof(server_addr));
    if(ret == -1){
```

```c
        perror("bind err!");
        exit(1);
    }
    ret = listen(lfd,LISTEN_NUM);
if(ret == -1){
        perror("listen err!");
        exit(1);
    }
    return lfd;
}
static void do_epoll(int listenfd)
{
    int epoll_fd;
    struct epoll_event events[EPOLLEVENTS];
    int num;
    epoll_fd = epoll_create(FDSIZE);
    if(epoll_fd == -1 ){
        perror("epoll err!");
        exit(1);
    }
    add_event(epoll_fd,listenfd,EPOLLIN);
    for( ;; )
    {
       num = epoll_wait(epoll_fd,&events,EPOLLEVENTS,-1);
       handle_event(epoll_fd,num,events,listenfd);
    }
}

static void handle_event(int epoll_fd,int num,struct epoll_event * events,int listenfd)
{
    int i;
    int fd;
    for(i=0;i<num;i++)
    {
    fd = events[i].data.fd;
    if((fd == listenfd) && (events[i].events == EPOLLIN))
        hand_accept(listenfd,listenfd);
    //else if(events[i].event == EPOLLIN)
        //handle_read();
   // else if(events[i].event == EPOLLOUT)
```

```
        //handle_write();
    }
}
static void hand_accept(int epollfd,int listenfd)
{
    int clifd;
    struct sockaddr_in cli_sockaddr;
    socklen_t socklen;
    socklen = sizeof(cli_sockaddr);
    memset(&cli_sockaddr,0,sizeof(cli_sockaddr));
    clifd = accept(listenfd,(struct sockaddr*)&cli_sockaddr,&socklen);
   // clifd = accept(listenfd,NULL,socklen);
    if(clifd == -1){
        perror("accept err");
     //  exit(1);
    }
    else{
        printf("client ip is %s port is %d \n",inet_ntoa(cli_sockaddr.sin_addr),cli_sock
    }
    add_event(epollfd,clifd,EPOLLIN);
}

static int add_event(int epoll_fd,int fd,int state)
{
    struct epoll_event events;
    events.data.fd = fd;
    events.events = state;
    epoll_ctl(epoll_fd,EPOLL_CTL_ADD,fd,&events);
}
int main()
{
    int listenfd;
    listenfd = socket_bind(IPADDRESS,Port);
    do_epoll(listenfd);
    return 0;
}


```