三种并发思路

1.IO 多路复用

2.多进程

3.多线程

多线程服务器端的基本框架

MutexLock 和 Condition

Thread　ThreadPool

InetAddress　Socket

Select/Poller/Epoller

TcpServer TcpClient

IO 多路复用

select 编写服务器

```
int main(int argc, char **argv) {

    signal(SIGPIPE, SIG_IGN);
    int listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if (listenfd < 0) {
        ERR_EXIT("socket");
    }

    int on = 1;
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on))
< 0)
        ERR_EXIT("setsockopt");
```

```
struct sockaddr_in servaddr;
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(8989);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
socklen_t len = sizeof servaddr;
int ret = bind(listenfd, (struct sockaddr*) &servaddr, len);
if (ret < 0) {
    ERR_EXIT("bind");
}

ret = listen(listenfd, SOMAXCONN);
if (ret < 0) {
    ERR_EXIT("listen");
}

int i;
int client[FD_SETSIZE];
for (i = 0; i < FD_SETSIZE; ++i) {
    client[i] = -1;
}
int maxi = 0;
int maxfd = listenfd;
int nready;
fd_set allset;
fd_set rset;
FD_ZERO(&allset);
FD_ZERO(&rset);
FD_SET(listenfd, &allset);

while (1) {
    rset = allset;
    nready = select(maxfd + 1, &rset, NULL, NULL, NULL);
    if (nready == -1) {
        if (errno == EINTR) {
            continue;
        } else {
            ERR_EXIT("select");
        }
    }
    if (nready == 0) {
        continue;
    }
    if (FD_ISSET(listenfd, &rset)) {
```

```
        struct sockaddr_in peeraddr;
        bzero(&peeraddr, sizeof peeraddr);
        len = sizeof peeraddr;

        //accept
        int peerfd = accept(listenfd, (struct sockaddr*) &peeraddr, &len);
        if (peerfd == -1) {
            ERR_EXIT("accept");
        }
        //加入 clients
        int i;
        for (i = 0; i < FD_SETSIZE; ++i) {
            if (client[i] == -1) {
                client[i] = peerfd;
                if (i > maxi) {
                    maxi = i;
                }
                break;
            }
        }

        //too many
        if (i == FD_SETSIZE) {
            fprintf(stderr, "too many clients\n");
            exit(EXIT_FAILURE);
        }

        //加入 allset
        FD_SET(peerfd, &allset);
        if (peerfd > maxfd) {
            maxfd = peerfd;
        }
        fprintf(stdout, "IP = %s, port = %d\n",
                inet_ntoa(peeraddr.sin_addr), ntohs(peeraddr.sin_port));

        //如果等于零，说明其他 fd 不需要操作
        if (--nready <= 0) {
            continue;
        }

    }

    int i;
    for (i = 0; i <= maxi; ++i) {
```

```
                int peerfd = client[i];
                if (peerfd == -1) {
                    continue;
                }
                if (FD_ISSET(peerfd, &rset)) {
                    char recvbuf[MAXLINE + 1] = { 0 };
                    int ret = readline(peerfd, recvbuf, MAXLINE);
                    if (ret == -1) {
                        ERR_EXIT("readline");
                    }
                    if (ret == 0) {
                        fputs("client close\n", stdout);
                        FD_CLR(peerfd, &allset);
                        client[i] = -1;
                        close(peerfd);
                        continue;
                    }
                    fprintf(stdout, "receive: %s", recvbuf);
                    //sleep(4);
                    writen(peerfd, recvbuf, strlen(recvbuf));
                    //write(peerfd, "test\n", strlen("test\n"));

                    if (--nready <= 0) {
                        break;
                    }

                }

            }

        }

    //  close(peerfd);
        close(listenfd);

        return 0;
    }
```

poll 编写服务器

```
    int main(int argc, char **argv) {
```

```
        signal(SIGPIPE, SIG_IGN);
        int listenfd = socket(AF_INET, SOCK_STREAM, 0);
        if (listenfd < 0) {
            ERR_EXIT("socket");
        }

        int on = 1;
        if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on))
< 0)
            ERR_EXIT("setsockopt");

        struct sockaddr_in servaddr;
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(8989);
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        socklen_t len = sizeof servaddr;
        int ret = bind(listenfd, (struct sockaddr*) &servaddr, len);
        if (ret < 0) {
            ERR_EXIT("bind");
        }

        ret = listen(listenfd, SOMAXCONN);
        if (ret < 0) {
            ERR_EXIT("listen");
        }


        struct pollfd client[2048];
        int i;
        for (i = 0; i < 2048; ++i) {
            client[i].fd = -1;
        }
        client[0].fd = listenfd;
        client[0].events = POLLIN;
        int maxi = 0;
        int nready;

        while (1) {
            /*rset = allset;
              nready = select(maxfd + 1, &rset, NULL, NULL, NULL); */
            nready = poll(client, maxi + 1, -1);
            if (nready == -1) {
                if (errno == EINTR) {
```

```
            continue;
        } else {
            ERR_EXIT("poll");
        }
    }
    if (nready == 0) {
        continue;
    }
    //if (FD_ISSET(listenfd, &rset)) {
    if (client[0].revents & POLLIN) {
        struct sockaddr_in peeraddr;
        bzero(&peeraddr, sizeof peeraddr);
        len = sizeof peeraddr;

        //accept
        int peerfd = accept(listenfd, (struct sockaddr*) &peeraddr, &len);
        if (peerfd == -1) {
            ERR_EXIT("accept");
        }


        int i;
        for (i = 0; i < 2048; ++i) {
            if (client[i].fd == -1) {
                client[i].fd = peerfd;
                client[i].events = POLLIN;    //容易遗漏
                if (i > maxi) {
                    maxi = i;
                }
                break;
            }
        }

        if (i == 2048) {
            fprintf(stderr, "too many clients\n");
            exit(EXIT_FAILURE);
        }

        fprintf(stdout, "IP = %s, port = %d\n",
                inet_ntoa(peeraddr.sin_addr), ntohs(peeraddr.sin_port));

        //如果等于零，说明其他 fd 不需要操作
        if (--nready <= 0) {
            continue;
```

```
            }

        }

        int i;
        //for (i = 0; i <= maxi; ++i) {
        for (i = 1; i <= maxi; ++i) {
            int peerfd = client[i].fd;
            if (peerfd == -1) {
                continue;
            }
            //if (FD_ISSET(peerfd, &rset)) {
            if (client[i].revents & POLLIN) {
                char recvbuf[MAXLINE + 1] = { 0 };
                int ret = readline(peerfd, recvbuf, MAXLINE);
                if (ret == -1) {
                    ERR_EXIT("readline");
                }
                if (ret == 0) {
                    fputs("client close\n", stdout);
                    //FD_CLR(peerfd, &allset);
                    close(peerfd);
                    client[i].fd = -1;
                    continue;
                }
                fprintf(stdout, "receive: %s", recvbuf);
                //sleep(4);
                writen(peerfd, recvbuf, strlen(recvbuf));
                //write(peerfd, "test\n", strlen("test\n"));

                if (--nready <= 0) {
                    break;
                }

            }

        }

    }

//  close(peerfd);
    close(listenfd);

    return 0;
```

```
}
```

epoll 编写服务器

```
int main(int argc, char **argv) {

    signal(SIGPIPE, SIG_IGN);
    int listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if (listenfd < 0) {
        ERR_EXIT("socket");
    }

    int on = 1;
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0)
        ERR_EXIT("setsockopt");

    struct sockaddr_in servaddr;
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8989);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    socklen_t len = sizeof servaddr;
    int ret = bind(listenfd, (struct sockaddr*) &servaddr, len);
    if (ret < 0) {
        ERR_EXIT("bind");
    }

    ret = listen(listenfd, SOMAXCONN);
    if (ret < 0) {
        ERR_EXIT("listen");
    }


    int epollfd = epoll_create(EVENT_MAX);
    if (epollfd == -1) {
        ERR_EXIT("epoll_create");
    }
    struct epoll_event events[EVENT_MAX];
    int nready;
    struct epoll_event ev;
```

```
ev.data.fd = listenfd;
ev.events = EPOLLIN;
ret = epoll_ctl(epollfd, EPOLL_CTL_ADD, listenfd, &ev);
if (ret == -1) {
    ERR_EXIT("epoll_ctl");
}

while (1) {

    //nready = poll(client, maxi + 1, -1);
    nready = epoll_wait(epollfd, events, EVENT_MAX, -1);
    if (nready == -1) {
        if (errno == EINTR) {
            continue;
        } else {
            ERR_EXIT("epoll");
        }
    }
    if (nready == 0) {
        continue;
    }

    int i;
    for (i = 0; i < nready; ++i) {

        //if (client[0].revents & POLLIN) {
        if (events[i].data.fd == listenfd) {
            struct sockaddr_in peeraddr;
            bzero(&peeraddr, sizeof peeraddr);
            len = sizeof peeraddr;

            //accept
            int connfd = accept(listenfd, (struct sockaddr*) &peeraddr,
                    &len);
            if (connfd == -1) {
                ERR_EXIT("accept");
            }

            struct epoll_event ev;
            ev.data.fd = connfd;
            ev.events = EPOLLIN;
            int ret = epoll_ctl(epollfd, EPOLL_CTL_ADD, connfd, &ev);
            if(ret == -1){
                ERR_EXIT("epoll add");
```

```
                }


                fprintf(stdout, "IP = %s, port = %d\n",
                        inet_ntoa(peeraddr.sin_addr), ntohs(peeraddr.sin_port));

            } else {
                int peerfd = events[i].data.fd;
                if (peerfd == -1) {
                    continue;
                }
                //if (FD_ISSET(peerfd, &rset)) {
                if (events[i].events & POLLIN) {
                    char recvbuf[MAXLINE + 1] = { 0 };
                    int ret = readline(peerfd, recvbuf, MAXLINE);
                    if (ret == -1) {
                        ERR_EXIT("readline");
                    }
                    if (ret == 0) {
                        fputs("client close\n", stdout);
                        close(peerfd);
                        //client[i].fd = -1;
                        struct epoll_event ev;
                        ev.data.fd = peerfd;

                        ret = epoll_ctl(epollfd, EPOLL_CTL_DEL, peerfd, &ev);
                        if (ret == -1) {
                            ERR_EXIT("epoll_ctl");
                        }
                        continue;
                    }
                    fprintf(stdout, "receive: %s", recvbuf);
                    writen(peerfd, recvbuf, strlen(recvbuf));

                }
            }

        }

    }

//  close(peerfd);
    close(listenfd);
```

```
    return 0;
}
```