电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2014220202021

姓 名 李家兴

（实验） 课程名称 网络计算编程

理论教师 张翔

实验教师 张翔

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：李家兴 学号：2014220202021**

**指导教师：张翔**

**实验地点：信软楼西303 实验时间：2016.10.21、2016.10.28**

1. **实验名称：面向Timer服务的TCP多进程服务器与单进程客户端设计**
2. **实验学时：4**
3. **实验目的：**
4. 理解并掌握多进程服务器编写的基本流程
5. 理解fork函数的使用方法，返回值
6. 理解僵尸进程的意义
7. 理解信号与wait/waitpid
8. **实验原理：**
9. 迭代服务器的特点
10. 多用户的情况下，连接建立后，若没有accept，连接存放在listen函数队列
11. accept从已完成队列的对头摘取新套接字
12. 对客户端的服务是一个串行的过程
13. 多进程并发服务模型
14. 采用多进程的方式引入并行
15. 目的是：使得每个客户端都能够得到一个进程的服务
16. 进程与fork函数
17. 僵尸进程

在fork()/execve()过程中，假设子进程结束时父进程仍然存在，而父进程fork()之前既没安装SIGCHILD信号处理函数调用wait()或waitpid()等待子进程结束，又没有显式忽略该信号，则子进程成为僵尸进程，无法正常结束。

1. **实验内容：**
2. 自行编码设计，在UNP环境下编码实现多进程服务器与单进程客户端
3. 自行编码设计，在原生Linux环境下编码实现多进程服务器与单进程客户端
4. **实验器材（设备、元器件）：**

PC机，VMWARE Workstation虚拟机Ubuntu系统

1. **实验步骤：**
2. 使用fork函数创建多进程
3. 子进程

-关闭监听连接

-调用处理函数，处理客户端的连接

-退出子进程

1. 父进程

-关闭接收客户端的连接

1. 主函数最开始注册信号处理函数

-signal(SIGCHID,sig\_child);

1. 增加信号处理函数

-调用wait或waitpid等待子进程退出，防止出现僵尸进程。 while ((pid = waitpid(-1,&stat,WNOHANG)) > 0) {return;}

1. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**

多进程并发服务器的特点是：采用多进程的方式引入并行，通过fork函数使得每个客户端都能够得到一个进程的服务。

但是在fork()/execve()过程中，假设子进程结束时父进程仍然存在，而父进程fork()之前既没安装SIGCHILD信号处理函数调用wait()或waitpid()等待子进程结束，又没有显式忽略该信号，则子进程成为僵尸进程，无法正常结束。

因此未注册信号处理函数的多进程服务器无法检测到子进程是否关闭，从而容易产生僵尸进程；反之注册了信号处理函数的多进程服务器则显示了子进程结束的信息，并且调用了信号处理函数，能够很容易地管理子进程的关闭，不至于使子进程挂起而产生僵尸进程。

使用fork函数创建多进程的过程为：服务器端（父进程）等待客户端的服务请求到达时，父进程调用fork，使子进程处理本次请求，服务器此时可继续等待下一个子进程请求。

1. **UNP环境**
   1. **未注册信号处理函数**

未调用注册信号函数，仅通过Fork()函数实现多进程服务，当多个子进程请求到达时能够调用str\_echo函数打印子进程传输过来的字符串，但当子进程终止时无响应信息（服务器打开，当ONE子进程请求到达时（图1-2），显示子进程进程号（2735）及内容（ONe!）（图1-1）; 当TWO子进程请求到达时（图1-3），显示子进程进程号（2735）及内容（TWO）（图1-1）。但是当ONE和TWO终止退出时，多进程服务器无响应（图1-4））

* + 1. **Server端**（图1-1、图1-4）

#include "unp.h"

#define BUFFER\_SIZE 1024

char buffer[BUFFER\_SIZE];

void str\_echo2(int sockfd, char\* buf)

{

ssize\_t n;

while (1) {

n = read(sockfd, buf, BUFFER\_SIZE);

if (n > 0){

printf("%s",buf);

Write(sockfd, buf, n);}

else

return;

}

}

int main(int argc, char \*argv[])

{

int listenfd, connfd;

size\_t clilen;

int backlog=5;

struct sockaddr\_in cliaddr, servaddr;

listenfd = Socket(AF\_INET, SOCK\_STREAM, 0);

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl (INADDR\_ANY);

servaddr.sin\_port = htons(SERV\_PORT);

Bind(listenfd, (struct sockaddr \*) &servaddr,

sizeof(servaddr));

Listen(listenfd, backlog);

* + 1. **Client端**（图1-2、图1-3）

#include "unp.h"

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

int sockfd;

struct sockaddr\_in servaddr;

if (argc != 2)

err\_quit("usage: tcpcli<IPaddress>");

sockfd = Socket(AF\_INET, SOCK\_STREAM, 0);

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(SERV\_PORT);

Inet\_pton(AF\_INET, argv[1], &servaddr.sin\_addr);

Connect(sockfd, (SA \*) &servaddr, sizeof(servaddr));

str\_cli(stdin, sockfd);

exit(0);

}

void str\_cli(FILE \*fp, int sockfd) {

char sendline[MAXLINE], recvline[MAXLINE];

while (Fgets(sendline, MAXLINE, fp) != NULL) {

Write(sockfd, sendline, strlen(sendline));

if (Readline(sockfd, recvline, MAXLINE) == 0)

err\_quit("str\_cli: server terminated prematurely");

}

}

while (1) {

clilen = sizeof(cliaddr);

connfd = Accept(listenfd, (SA \*) &cliaddr, &clilen);

if (Fork() == 0) {

printf("this is child %d\n",getpid());

Close(listenfd);

str\_echo2(connfd, buffer);

Close(connfd);

return 0;

}

Close(connfd);

}

return 0;

}



图1-1 UNP-normal服务端接收子进程请求



图1-2 ONE客户端进程请求成功

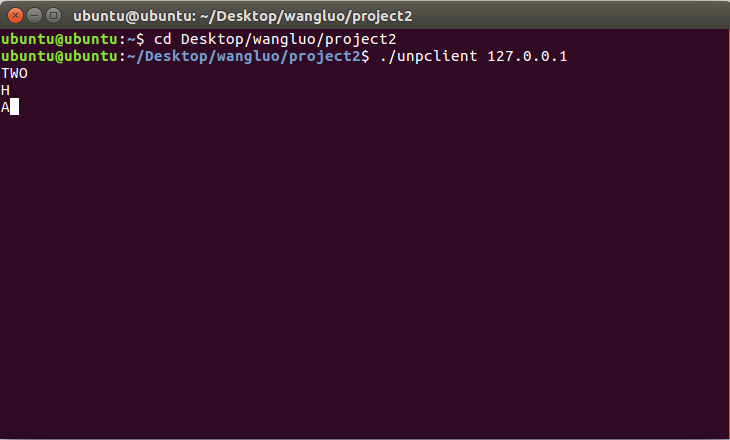


图1-3 TWO客户端进程请求成功

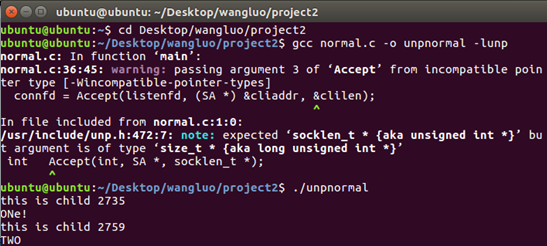


图1-4 当ONE和TWO两个客户端进程结束后服务器端反映

1.2 **注册信号处理函数**

使用signal函数注册信号处理函数，若子进程结束了，waitpid()函数返回一个大于0的数，打印该子进程的id；若子进程未结束，waitpid()函数返回0，不予等待。

当一个进程终止（正常终止或异常终止）时，内核就像其父进程发送SIGCHILD信号。

当注册了信号处理函数过后，当promote服务器端接收到ONE、TWO客户端的退出信号之后，便显示出当前退出的进程ID（ONE(2841)、TWO(2865)）（图2-4）。

1.2.1 **Server端**（图2-1、图2-4）

#include "unp.h"

#define BUFFER\_SIZE 1024

char buffer[BUFFER\_SIZE];

void sig\_child(int sign)

{

pid\_t pid;

int stat;

while ((pid = waitpid(-1,&stat,WNOHANG)) > 0) {

printf("child %d terminated.\n", pid);

return;}

}

void str\_echo2(int sockfd, char\* buf)

{

ssize\_t n;

while (1) {

n = read(sockfd, buf, BUFFER\_SIZE);

if (n > 0){

printf("%s\n",buf);

Write(sockfd, buf, n);}

else

return;

}

}

int main(int argc, char \*argv[])

{

int listenfd, connfd;

size\_t clilen;

int backlog=5;

struct sockaddr\_in cliaddr, servaddr;

signal(SIGCHLD, sig\_child);

listenfd = Socket(AF\_INET, SOCK\_STREAM, 0);

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl (INADDR\_ANY);

servaddr.sin\_port = htons(SERV\_PORT);

Bind(listenfd, (struct sockaddr \*) &servaddr,

sizeof(servaddr));

Listen(listenfd, backlog);

while (1) {

clilen = sizeof(cliaddr);

connfd = Accept(listenfd, (struct sockaddr \*)&cliaddr, &clilen);

if (connfd < 0 && connfd == EINTR) continue;

if (Fork() == 0) {

printf("this is child %d\n",getpid());

Close(listenfd);

str\_echo2(connfd, buffer);

Close(connfd);

return 0;}

Close(connfd);

}

return 0;

}

1.2.2 **Client客户端**（图2-2、图2-3）

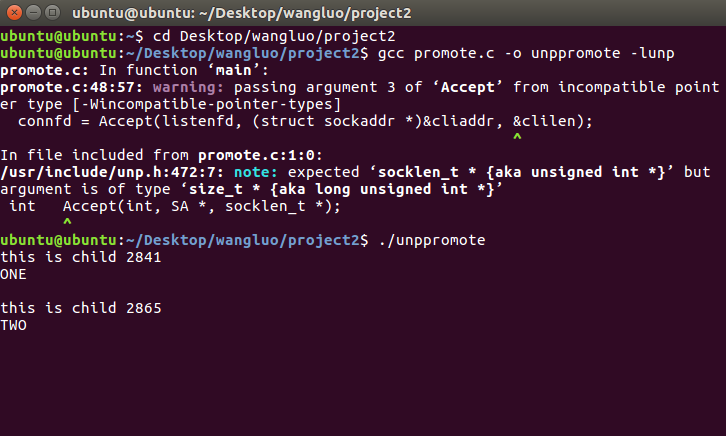


图2-1 UNP-promote服务器端接收客户端请求

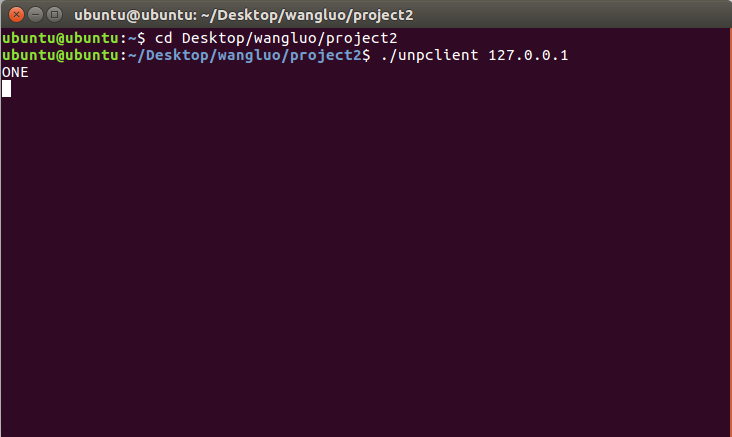


图2-2 ONE单进程客户端向服务器端请求

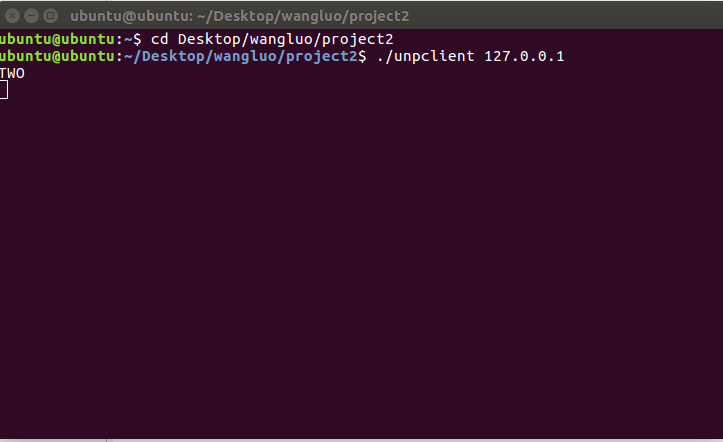


图2-3 TWO单进程客户端向服务器端请求

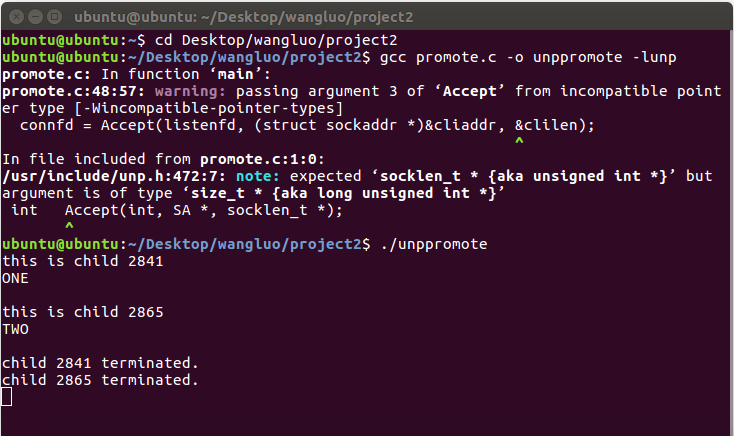


图2-4 UNP-promote处理单进程客户端退出信息

1. **Linux原生环境**
   1. **未注册信号处理函数**
      1. **Server端**（图3-1、图3-4）

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <linux/in.h>

#include <signal.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

char buffer[BUFFER\_SIZE];

void str\_echo2(int sockfd, char\* buf)

{

ssize\_t n;

again:

while((n = read(sockfd, buf, BUFFER\_SIZE)) > 0){

printf("%s",buf);

write(sockfd, buf, n);

}

if(n < 0)

goto again;

else if(n < 0)

printf("str\_echo: read error");

}

int main(int argc, char \*argv[])

{

int listenfd, connfd;

size\_t clilen;

int backlog=5;

pid\_t pid;

int err;

struct sockaddr\_in cliaddr, servaddr;

listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

if(listenfd < 0){

printf("socket error\n");

return -1;

}

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl (INADDR\_ANY);

servaddr.sin\_port = htons(PORT);

err = bind(listenfd, (struct sockaddr\*)&servaddr, sizeof(servaddr));

if(err < 0){

printf("bind error\n");

return -1;

}

err = listen(listenfd, backlog);

if(err < 0){

printf("listen error\n");

return -1;

}

while (1) {

int clilen = sizeof(struct sockaddr);

connfd = accept(listenfd,(struct sockaddr\*)&cliaddr, &clilen);

if(connfd < 0){

continue; }

pid = fork();

if (pid == 0) {

printf("child %d\n", getpid());

close(listenfd);

str\_echo2(connfd, buffer);

close(connfd);

return 0;}

close(connfd);

}

return 0;

}

* + 1. Client端（图3-2、图3-3）

#include <stdio.h>

#include <stdlib.h>

#include <strings.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <linux/in.h>

#include <signal.h>

static int s;

#define PORT 8888

void process\_conn\_client(int s)

{

ssize\_t size = 0;

char buffer[1024];

for(;;){

size = read(0, buffer, 1024);

if(size > 0){

write(s, buffer, size);

size = read(s, buffer, 1024);

if(size < 0)

printf("read error!\n");

//write(1, buffer, size);

}

}

}

int main(int argc, char \*argv[])

{

struct sockaddr\_in server\_addr;

int err;

s = socket(AF\_INET, SOCK\_STREAM, 0);

if(s < 0){

printf("socket error\n");

return -1;

}

bzero(&server\_addr, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

server\_addr.sin\_port = htons(PORT);

inet\_pton(AF\_INET, argv[1], &server\_addr.sin\_addr);

connect(s, (struct sockaddr\*)&server\_addr, sizeof(struct sockaddr));

process\_conn\_client(s);

close(s);

}

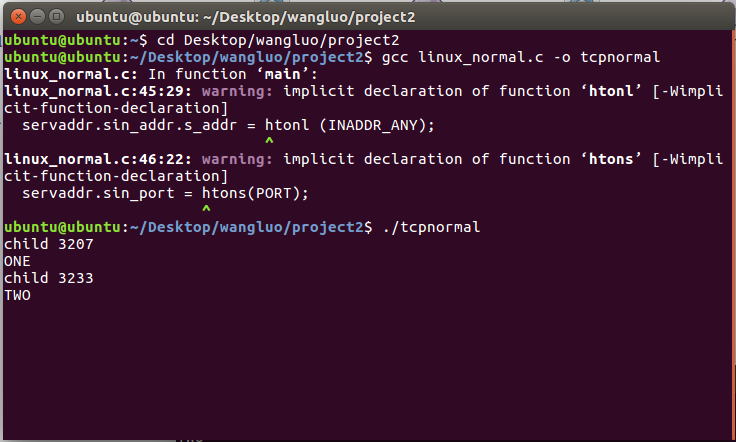


图3-1 未注册信号处理函数的服务器接收客户端请求

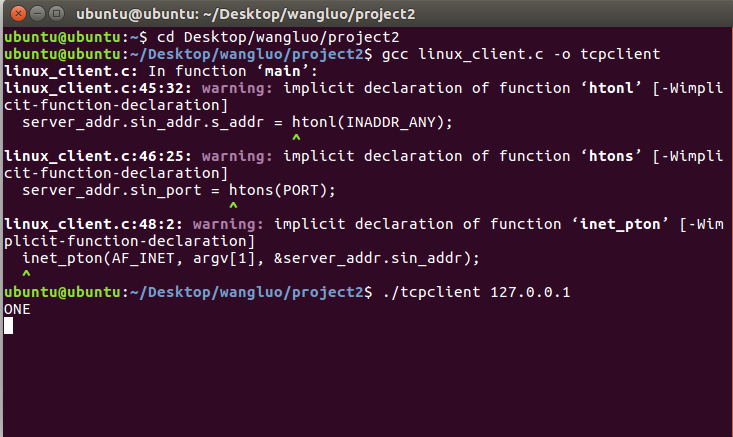


图3-2 ONE客户端向服务器端发出请求

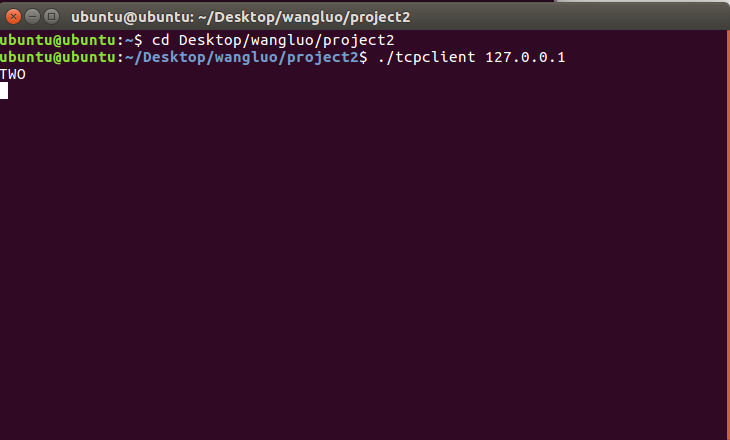


图3-3 TWO客户端向服务器端发出请求

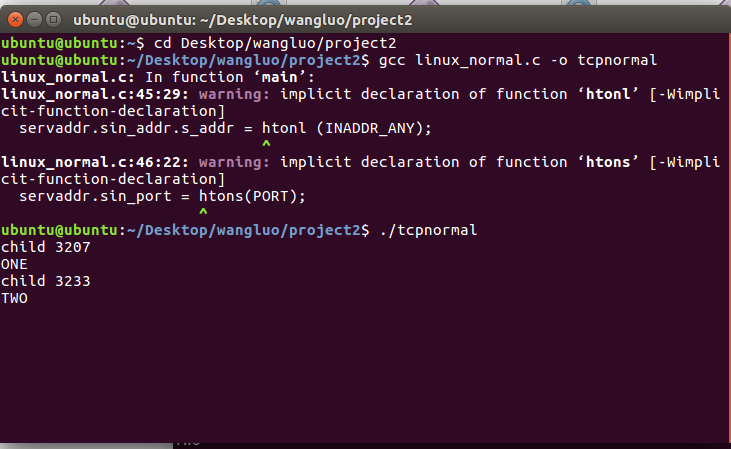


图3-4 ONE、TWO两个客户端退出后的服务器端响应

* 1. 注册信号处理函数
     1. Server端（图4-1、图4-4）

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <linux/in.h>

#include <signal.h>

#define PORT 8888

#define BUFFER\_SIZE 1024

char buffer[BUFFER\_SIZE];

void sig\_child(int sign)

{

pid\_t pid;

int stat;

while ((pid = waitpid(-1,&stat,WNOHANG)) > 0) {

printf("child %d terminated.\n", pid);

return;

}

}

void str\_echo2(int sockfd, char\* buf)

{

ssize\_t n;

again:

while((n = read(sockfd, buf, BUFFER\_SIZE)) > 0){

printf("%s",buf);

write(sockfd, buf, n);

}

if(n < 0)

goto again;

else if(n < 0)

printf("str\_echo: read error");

}

int main(int argc, char \*argv[])

{

int listenfd, connfd;

size\_t clilen;

int backlog=5;

pid\_t pid;

int err;

struct sockaddr\_in cliaddr, servaddr;

signal(SIGCHLD, sig\_child);

listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

if(listenfd < 0){

printf("socket error\n");

return -1;

}

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl (INADDR\_ANY);

servaddr.sin\_port = htons(PORT);

err = bind(listenfd, (struct sockaddr\*)&servaddr, sizeof(servaddr));

if(err < 0){

printf("bind error\n");

return -1;

}

err = listen(listenfd, backlog);

if(err < 0){

printf("listen error\n");

return -1;

}

while (1) {

int clilen = sizeof(struct sockaddr);

connfd = accept(listenfd,(struct sockaddr\*)&cliaddr, &clilen);

if(connfd < 0){

continue;

}

pid = fork();

if (pid == 0) {

printf("this is child %d\n",getpid());

close(listenfd);

str\_echo2(connfd, buffer);

close(connfd);

return 0;

}

}

return 0;

}

2.2.2 Client客户端（图4-2、图4-3）

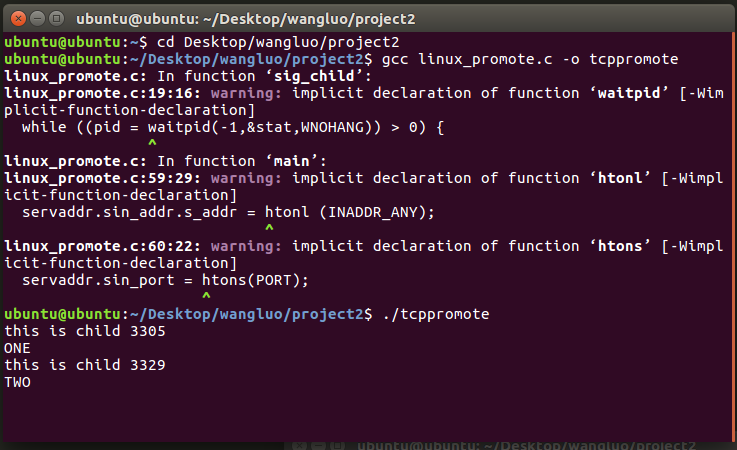


图4-1 注册信号处理函数的服务器接收客户端请求

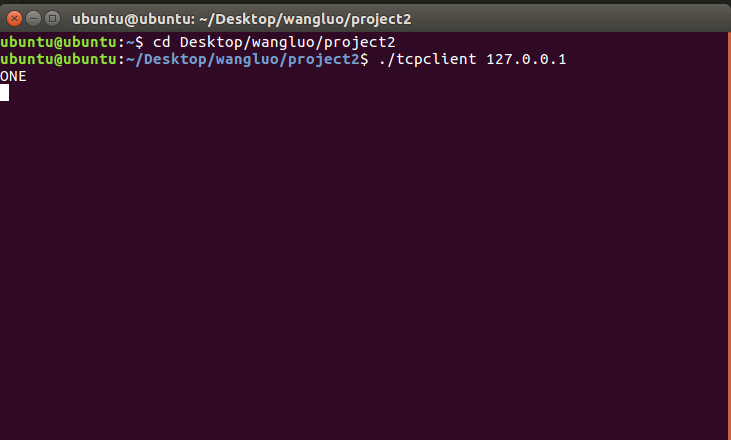


图4-2 ONE客户端请求服务器

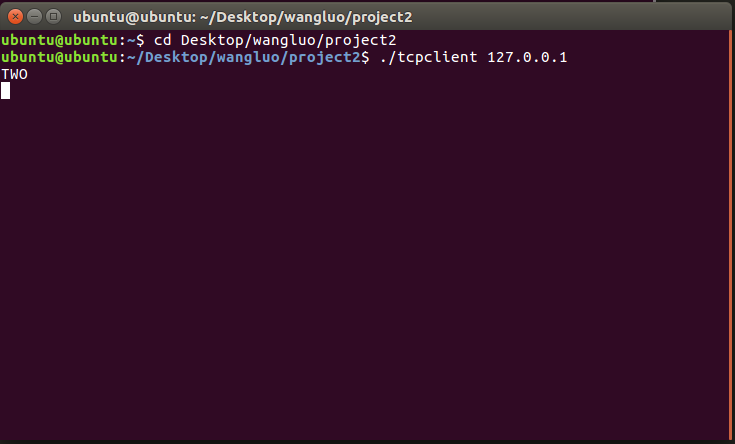


图4-3 TWO客户端请求服务器

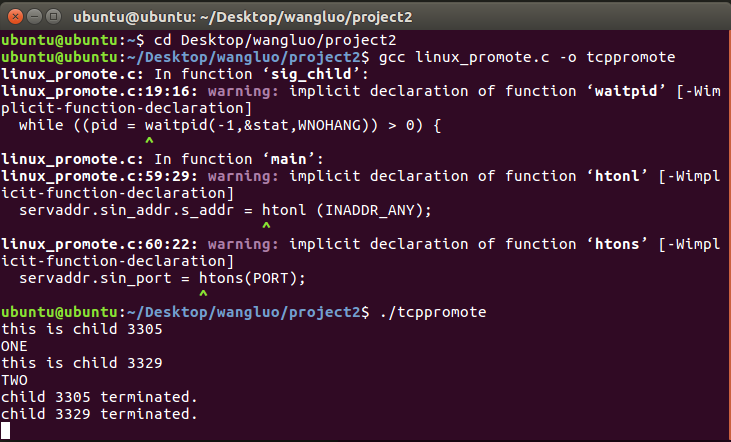


图4-4 ONE、TWO两个客户端退出后的服务器端响应

1. **总结及心得体会：**

通过本次实验，掌握了使用了fork函数创建多进程的方法，并且理解了注册信号处理函数对于处理僵尸进程的意义。

1. **对本实验过程及方法、手段的改进建议：**

在一开始使用fork()函数创建子进程时，由于没有对fork()函数的返回值掌握清楚，导致出错很多，当加了返回值判断机制后，整个程序的问题所在一目了然。

**报告评分：**

**指导教师签字：**