网络程序设计报告

实验一 linux 平台上的 TCP 并发服务

1 实验内容

掌握基本套接字函数使用方法、TCP 协议工作原理、并发服务原理和编程方法。实验内容：

在 linux 平台上实现 1 个 TCP 并发服务器，至少可以为 10 个客户端同时提供服务。

（1）基于 TCP 套接字编写服务器端程序代码，然后编译和调试；

（2）服务器程序要达到：可以绑定从终端输入的 IP 地址和端口；可以显示每一个进程的进

程号；可以显示当前并发执行的进程数量；可以根据客户机要求的服务时间确定进程的生存

时间。

（3）基于 TCP 套接字编写客户端程序代码，然后编译和调试；

（4）客户端程序要达到：可以从终端输入服务器的 IP 地址和端口；可以从终端输入对服务

器的服务时间要求。

（5）联调服务器和客户端，服务器每收到一个连接就新建一个子进程，在子进程中接收客

户端的服务时间请求，根据所请求的时间进行延时，然后终止子进程。如：客户端请求服务

10s，则服务器的子进程运行 10s，然后结束。

（6）服务器要清除因并发服务而产生的僵尸进程。

2 设计分析

（1）要求处理 10 个以上客户端，服务端采用 TCP 并发服务器，每进行一次连接创建一个

子进程，子进程总数加 1。

（2）需要实现端口的绑定和运行时间的设置，中国在客户端实现，argv[]中输入需要的参数，

参数不正确的时候出错处理。

（3）延时功能我使用的 sleep(),直接延时从客户端发送过来的要求的时间，最后结束的时候

子进程需要减 1。

（4）处理僵尸进程，自己实现一个 sigchld\_handler 实现僵尸进程的清理。

3 实现代码

server.c

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <signal.h>

#include <unistd.h>

#include <time.h>

#define LISTENQ 1024

#define BUFSIZE 1024

int open\_listenfd(int port);

void service(int connfd);

void sigchld\_handler(int sig);

static int pcnt;

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

{

int listenfd, connfd, port;

unsigned int clientlen;

struct sockaddr\_in clientaddr;

pid\_t pid;

if (argc != 2) {

fprintf(stderr, "%s <port>\n", argv[0]);

exit(-1);

}

port = atoi(argv[1]);

signal(SIGCHLD, sigchld\_handler);

if ((listenfd = open\_listenfd(port)) == -1) {

fprintf(stderr, "%s\n", strerror(errno));

exit(-1);

}

for (;;) {

clientlen = sizeof(clientaddr);

do {

connfd = accept(listenfd,

(struct sockaddr \*)&clientaddr.sin\_addr.s\_addr,

&clientlen);

} while (connfd == -1 && errno == EINTR);

if (connfd == -1) {

fprintf(stderr, "%s\n", strerror(errno));

exit(-1);

}

if ((pid = fork()) == -1) {

fprintf(stderr, "%s\n", strerror(errno));

} else {

pcnt++;

if (pid == 0) {

close(listenfd);

service(connfd);

close(connfd);

exit(EXIT\_SUCCESS);

}

}

close(connfd);

}

exit(EXIT\_SUCCESS);

}

void service(int connfd)

{

char buf[BUFSIZE];

int seconds;

time\_t tmval;

const struct tm \*tmptr;

read(connfd, buf, BUFSIZE);

sscanf(buf, "%d", &seconds);

if (seconds > 0) {

tmval = time(0);

tmptr = localtime(&tmval);

printf("process %d: %d seconds. total child\_pid: %d\n",getpid(), seconds, pcnt);

sleep(seconds);

}

}

void sigchld\_handler(int sig)

{

pid\_t pid;

time\_t tmval;

const struct tm \*tmptr;

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

tmval = time(0);

tmptr = localtime(&tmval);

printf("process %d exited. total child\_pid: %d\n",pid, --pcnt);

}

signal(SIGCHLD, sigchld\_handler);

}

int open\_listenfd(int port)

{

int listenfd, optval = 1;

struct sockaddr\_in serveraddr;

/\* creat socket \*/

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

return -1;

/\* setsocket \*/

if (setsockopt(listenfd, SOL\_SOCKET, SO\_REUSEADDR,

(const void \*)&optval, sizeof(int)) < 0)

return -1;

memset(&serveraddr, 0, sizeof(serveraddr));

serveraddr.sin\_family = AF\_INET;

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

serveraddr.sin\_port = htons((unsigned short) port);

/\*bind\*/

if (bind(listenfd, (struct sockaddr \*)&serveraddr,

sizeof(serveraddr)) < 0)

return -1;

if (listen(listenfd, LISTENQ) < 0)

return -1;

return listenfd;

}

client.c

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <unistd.h>

#include <sys/types.h>

int open\_clientfd(char \*hostname, int port);

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

{

int clientfd, port;

char \*host, \*seconds;

if (argc != 4) {

fprintf(stderr, "usage: %s <host> <port> <seconds>\n", argv[0]);

exit(-1);

}

host = argv[1];

port = atoi(argv[2]);

seconds = argv[3];

if ((clientfd = open\_clientfd(host, port)) == -1) {

fprintf(stderr, "%s\n", strerror(errno));

exit(-1);

}

write(clientfd, seconds, strlen(seconds));

close(clientfd);

exit(EXIT\_SUCCESS);

}

int open\_clientfd(char \*hostname, int port)

{

int clientfd;

struct hostent \*hp;

struct sockaddr\_in serveraddr;

if ((clientfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

return -1;

if ((hp = gethostbyname(hostname)) == NULL)

return -2;

memset(&serveraddr, 0, sizeof(serveraddr));

serveraddr.sin\_family = AF\_INET;

memcpy((char \*)&serveraddr.sin\_addr.s\_addr,

(char \*)hp->h\_addr\_list[0],

hp->h\_length);

serveraddr.sin\_port = htons(port);

/\*connect \*/

if (connect(clientfd, (struct sockaddr \*)&serveraddr,

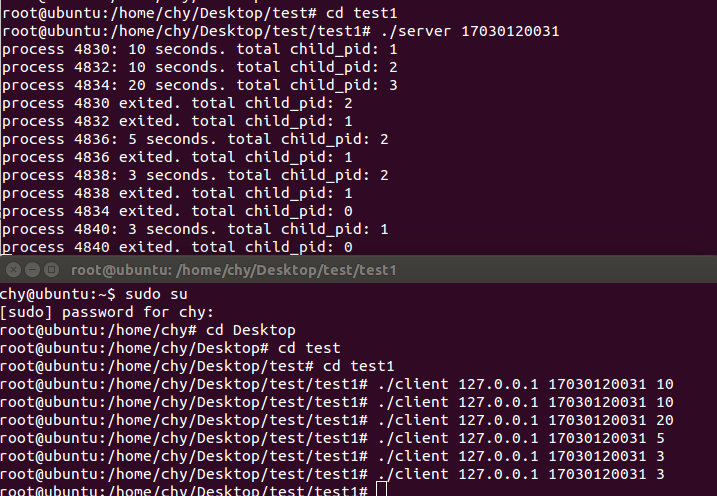
sizeof(serveraddr)) < 0)

return -1;

return clientfd;

}

4.实验结果



可以看到，实现了计算总的进程数，显示进程号和指定端口，子进程延时时间设置，僵尸进程清理等功能。

实验二 进程间的协调通信

1.实验内容

掌握进程的概念、进程间通信的基本原理、集成间通信的主要类型和各自的特点。实验内容：

在 linux 平台上实现 1 个父进程、2 个子进程，利用管道和共享内存实现两个子进程之间数

据快速传送。

(1） 创建一个进程，再创建一对管道、一块共享内存（大于 64kB）；

(2) 通过 fork（）函数生成 2 个子进程；

(3) 调试程序，确定父、子进程之间可以双向通信；

(4) 调试程序，确定两个子进程之间可以通过父进程中转实现双向通信；

(5) 调试程序，确定两个子进程都可访问共享内存；

(6) 实现两个子进程之间无冲突地访问共享内存。传送的数据块不小于 32kB，为了能够看

到演示效果，读/写每个字节后都延时 0.5ms。

2.设计分析

对于管道：

无名管道简单方便．但局限于单向通信的工作方式．并且只能在创建它的进程及其子孙进程

之间实现管道的共享.所以需要建立两个管道，实现子进程 1 写管道 1，父进程读管道 1 ；

父进程再把内容写进管道 2 里面，子进程 2 在管道 2 里面读，实现通信。



对于共享内存：

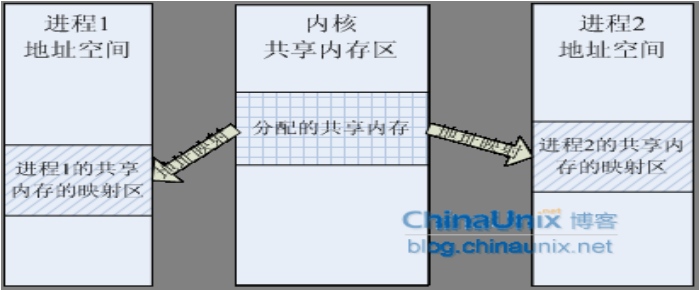
进程直接读写内存，不需要任何数据的拷贝

•为了在多个进程间交换信息，内核专门留出了一块内存区

•由需要访问的进程将其映射到自己私有地址空间

•进程直接读写这一内存区而不需要进行数据的拷贝，提高了效率

多个进程共享一段内存，需要依靠某种同步机制，如互斥锁和信号量等



3.代码实现

Pipe

#include<unistd.h>

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<wait.h>

#include<string.h>

#include<sys/types.h>

using namespace std;

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

{

int pipe1[2],pipe2[2];

//pid\_t pid1,pid2;

int n;

char cstr[]="child1 data";

char buf[128];

//char buf2[128];

//memset(buf1,'\0',128); //初始化

//memset(buf2,'\0',128);

if(pipe(pipe1)<0||pipe(pipe2)<0)

cout<<"pipe error"<<endl;

pid\_t pid1=fork();

pid\_t pid2=fork();

if(pid1==-1)

{

printf("fork pid1 failed!");

}

else if(pid1 > 0)

{

//父进程,用管道1读数据,管道2写数据

cout<<"father PID:"<<getpid()<<endl;

if(read(pipe1[0],buf,100)>0)

cout<<"father received:"<<buf<<endl;

write(pipe2[1],buf,strlen(buf));

}

else if(pid1==0)

{

cout<<"\npid1:"<<getpid()<<endl;

//子进程1用管道1写数据

close(pipe1[0]);//关闭pipe1读端口

close(pipe2[0]);//关闭pipe2读端口

close(pipe2[1]);//关闭pipe2写端口

write(pipe1[1],cstr,strlen(cstr));

}

if(pid2==-1)

{

printf("fork pid2 failed!");

}

else if(pid2==0)

{

//子进程2用管道2读数据

close(pipe1[0]);//关闭pipe1读端口

close(pipe1[1]);//关闭pipe1写端口

close(pipe2[1]);//关闭pipe2写端口

if(read(pipe2[0],buf,128)>0)

{

cout<<"\npid2:"<<getpid()<<endl;

//printf("\npid2 : pid=%d\n",getpid());

//printf("pid2 : buf=%s\n",buf);

cout<<"pid2 receive:"<<buf<<endl;

exit(0);

}

}

return 0;

}

Share\_mem

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <iostream>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <wait.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <sys/time.h>

#include <signal.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define MEM\_SIZE 1024

using namespace std;

int shmid;

int status;

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

{

char\* p\_addr, \*c1\_addr, \*c2\_addr;

key\_t key;

char\* name = (char\* )"/dev";

char\* text = (char\* )"hello";

int pid1,pid2;

key=ftok(name,'a');

if(key==-1)

{

perror("ftok error");

exit(1);

}

shmid=shmget(key,MEM\_SIZE,0777|IPC\_CREAT);

if(shmid==-1)

{

perror("shmget error");

exit(1);

}

if((pid1=fork())==-1)

{

printf("childprocess1 failed!");

}

else if(pid1>0)

{

cout<<"father PID:"<<getpid()<<endl;

wait(&status);

p\_addr = (char\* )shmat(shmid,0,0);

cout<<"father received:"<<p\_addr<<endl;

}

else

{

cout<<"child1 PID:"<<getpid()<<endl;

c1\_addr = (char\* )shmat(shmid,0,0);

memset(c1\_addr, '\0',1024);

strncpy(c1\_addr,text,1024);

exit(0);

}

if((pid2=fork())==-1)

{

printf("childprocess2 failed!");

}

else if(pid2==0)

{

c2\_addr = (char\* )shmat(shmid,0,0);

cout<<"\nchild2 PID:"<<getpid()<<endl;

cout<<"child2 received:"<<c2\_addr<<endl;

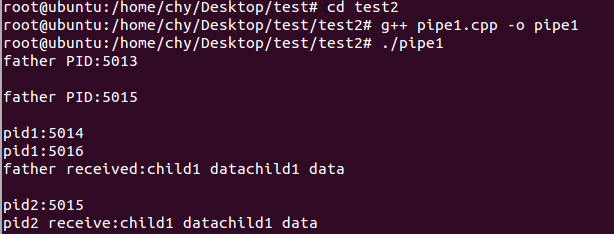
exit(0);

}

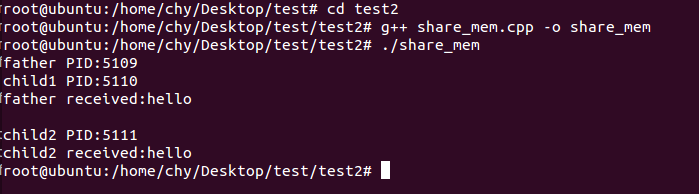
return 0;

}

4.实验结果



管道：子进程1写，子进程2读，成功。



通过共享内存两个子进程成功通信。

实验三 Windows 平台上的 TCP 并发服务

1.实验内容

编程内容与实验 1 相同，操作系统为 windows。了解 Windows 与 Linux 平台编程环境

的差异，掌握 Winsock 编程接口及编程方法。

2.实验试剂分析

服务控制管理器(SCM)是操作系统的组成部分，它的作用是与各种各样的服务进行通信。这种通信的流程为：

SCM ----启动服务进程----> 服务

SCM <----注册service-main函数---- 服务

SCM ----service-main----> 服务

SCM <----注册处理程序---- 服务

在系统启动时，将启动每个服务进程，进而调用该进程的主函数，它是该服务的入口。一个服务进程可能包含了多个服务，每项都会被该服务注册一个service-main函数入口点，service-main函数包含服务的实际功能。

 service-main函数的一个重要任务是用SCM注册一个处理程序。处理程序用来响应来自服务控制程序的事件，如SCM：停止，暂停或重新开始。

使用 winsocket 编程实现即可，需要查看许多函数，与 linux 稍微不同

我没有实现并发，只是单纯实现了这个模型，还有延时功能。

3.代码实现

Server

#include <stdio.h>

#include <winsock2.h>

#include <unistd.h>

#include <time.h>

#pragma comment (lib, "ws2\_32.lib")

int main()

{

WSADATA wsaData;

WSAStartup( MAKEWORD(2, 2), &wsaData);

fputs("port: ", stdout);

int port;

scanf("%d", &port);

SOCKET servSock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

struct sockaddr\_in sockAddr;

memset(&sockAddr, 0, sizeof(sockAddr));

sockAddr.sin\_family = PF\_INET;

sockAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

sockAddr.sin\_port = htons(port);

bind(servSock, (SOCKADDR\*)&sockAddr, sizeof(SOCKADDR));

listen(servSock, 20);

SOCKADDR clntAddr;

int nSize = sizeof(SOCKADDR);

for (;;) {

SOCKET clntSock = accept(servSock, (SOCKADDR\*)&clntAddr, &nSize);

char recvBuf[MAXBYTE] = {0};

recv(clntSock, recvBuf, MAXBYTE, 0);

int sec;

time\_t tmval;

struct tm \*tmptr;

sscanf(recvBuf, "%d", &sec);

if (sec > 0) {

tmval = time(0);

tmptr = localtime(&tmval);

printf("[%02d:%02d:%02d] server will sleep %d seconds\n",

tmptr->tm\_hour, tmptr->tm\_min,

tmptr->tm\_sec, sec);

sleep(sec);

tmval = time(0);

tmptr = localtime(&tmval);

printf("[%02d:%02d:%02d] server is wake up\n",

tmptr->tm\_hour, tmptr->tm\_min, tmptr->tm\_sec);

}

closesocket(clntSock);

}

closesocket(servSock);

WSACleanup();

return 0;

}

Client

#include <stdio.h>

#include <stdlib.h>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

int main()

{

WSADATA wsaData;

WSAStartup(MAKEWORD(2, 2), &wsaData);

fputs("ip: ", stdout);

char host[64];

scanf("%s", host);

fputs("port: ", stdout);

int port;

scanf("%d", &port);

SOCKET sock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

struct sockaddr\_in sockAddr;

memset(&sockAddr, 0, sizeof(sockAddr));

sockAddr.sin\_family = PF\_INET;

sockAddr.sin\_addr.s\_addr = inet\_addr(host);

sockAddr.sin\_port = htons(port);

connect(sock, (SOCKADDR\*)&sockAddr, sizeof(SOCKADDR));

char msg[MAXBYTE];

fputs("input serve second: ", stdout);

int time;

scanf("%d", &time);

snprintf(msg, MAXBYTE, "%d\n", time);

send(sock, msg, strlen(msg) + 1, 0);

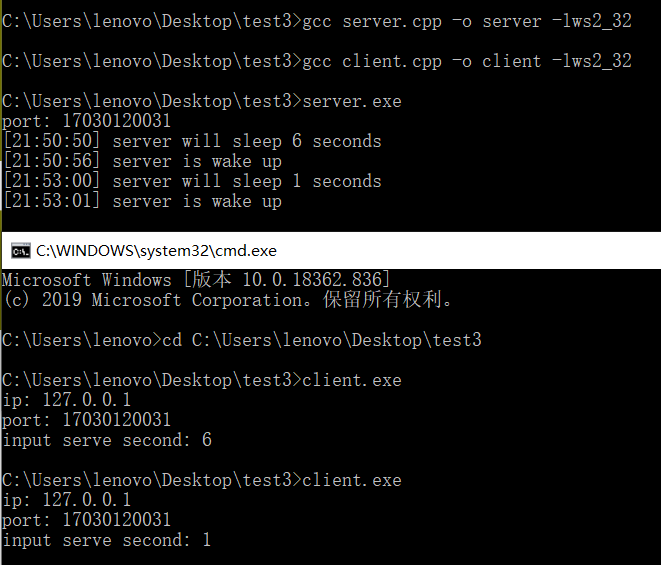
closesocket(sock);

WSACleanup();

return 0;

}

4.实验结果



达到了预期的效果，能够建立连接，实现延时。

出现的问题，[Error] ld returned 1 exit status，一直有这个问题，查了很多资料，是windons sdk的问题，最后只能在gcc \*.cpp -o \* 后面加上-lws2\_32，浪费了很多时间，但也了解了一个问题及其解决方法。