实验二 并发Web服务器的实现

【实验目的】

1. 熟悉Linux网络编程。
2. 了解Web服务器原理。
3. 掌握嵌入式Linux多进程、多线程、I/O多路复用三种方式并发服务器的实现。

【实验内容】

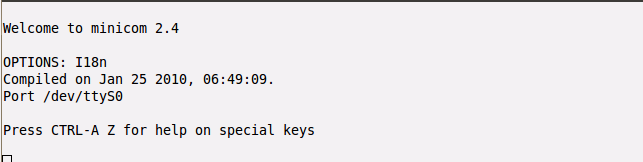
1. 用多进程实现Web服务器。
2. 用多线程实现Web服务器。
3. 用I/O多路复用方式实现Web服务器。

【实验步骤】

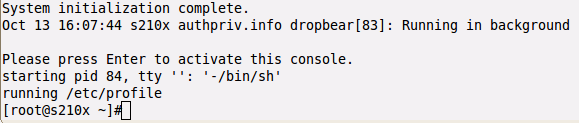
环境配置

1)打开实验开发板，用网线和串口线连接宿主机，并连接电源。

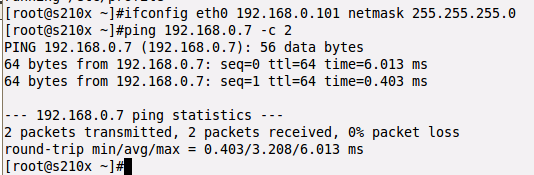
2)启动虚拟机中的Linux系统，在终端中运行minicom -s命令，启动minicom软件



3)拨动实验开发板的电源开关，启动目标机，进入嵌入式Linux系统。启动完了按回车



4）用ifconfig 查看目标机的ip地址，可以设为192.168.0.101,然后用ping命令来测试目标机和宿主机到连通性。

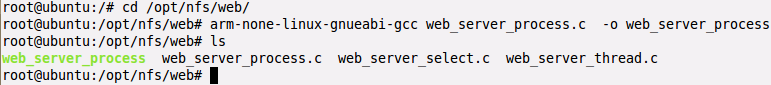


5)把程序源代码文件拷贝到宿主机的/opt/nfs/web目录下或者直接在此目录下创建并编辑源代码文件，结果如下。



实现多进程Web服务器

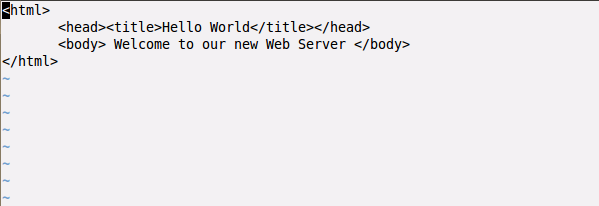
1)用arm-none-linux-gnueabi-gcc命令编译源程序，得到可执行程序web\_server\_process。



2)用vi文本编译器创建文件index.html，用于测试Web服务器。

vi index.html

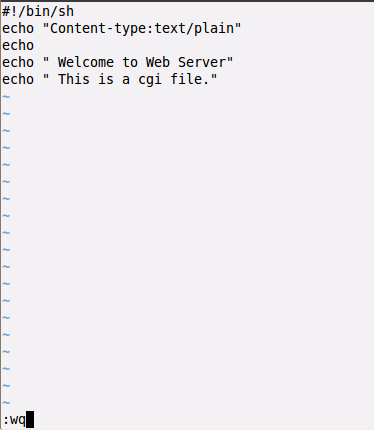
3)在本文件写一下内容。



4）在 /opt/nfs/web 目录创建 cgi-bin 目录，并在新创建的目录下用vi编辑器创建文件hello.cgi

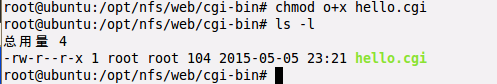
A description...

5)hello.cgi 的内容如下图。

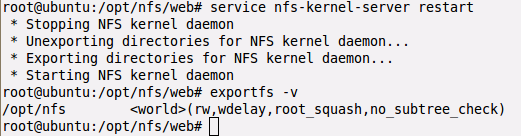


6）用ls命令可以看到其他用户没有执行权限，通过chmod命令来修改hello.cgi的权

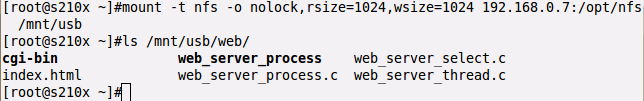
限，使它称为可执行文件。



7）使用service 命令来重启宿主机上的nfs 服务器（nfs-kernel-server），用exportfs 命令查看它的工作目录。如图所示



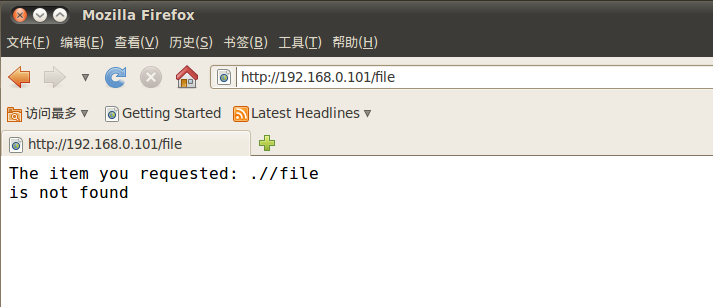
8）在目标机上挂载nfs.如下图。



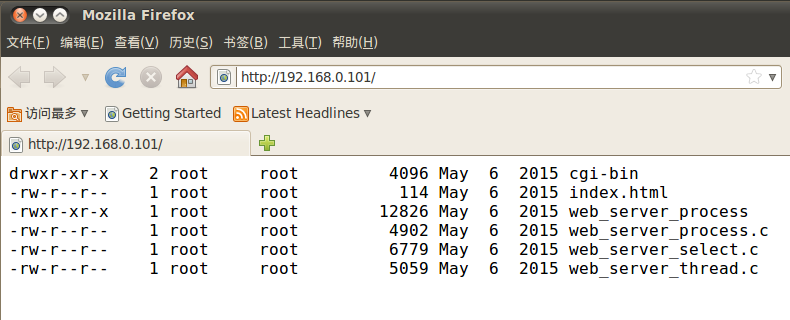
9）在目标机上运行web\_server\_process.

A description...

10)在宿主机打开浏览器，输入 ，http://192.168.0.101/file 查看结果。



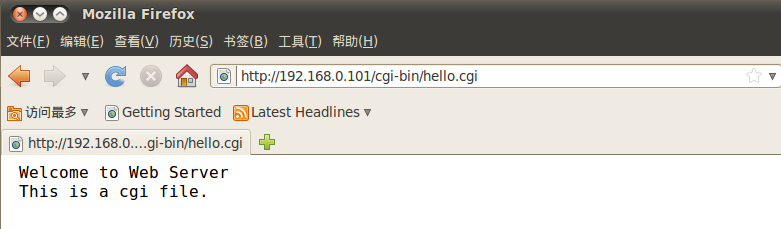
11）在宿主主机的浏览器输入，[http://192.168.0.101](http://192.168.0.101/) 查看结果



12）在宿主主机的浏览器输入，[http://192.168.0.101](http://192.168.0.101/)/index.html 查看结果



13)在宿主主机的浏览器输入，[http://192.168.0.101](http://192.168.0.101/)/cgi-bin/hello.cgi 查看结果

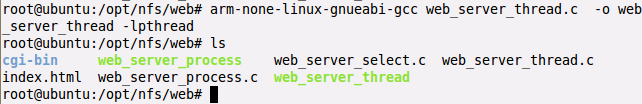


14)在宿主主机的浏览器输入，[http://192.168.0.101](http://192.168.0.101/)/web\_server\_process.c 查看结果



实现多线程Web服务器

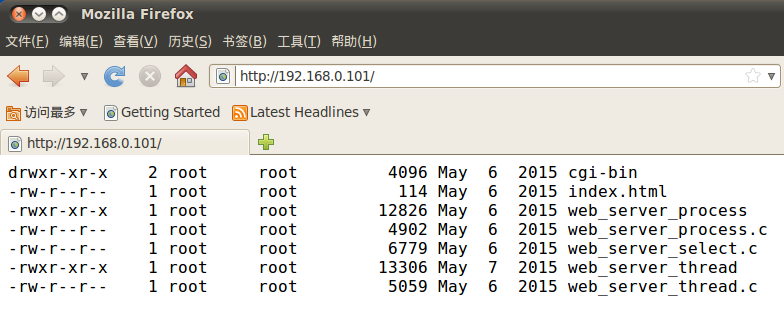
1)在宿主机终端用arm-none-linux-gnueabi-gcc编译源程序，得到可执行程序web\_server\_thread



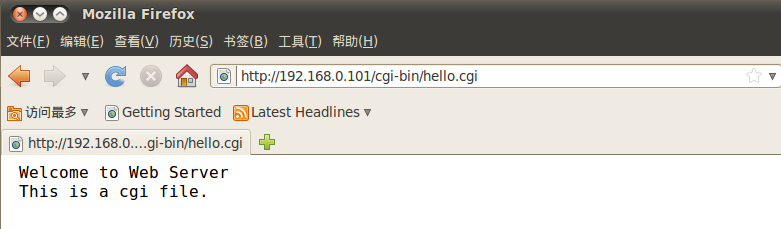
2）在目标机上运行web\_server\_thread



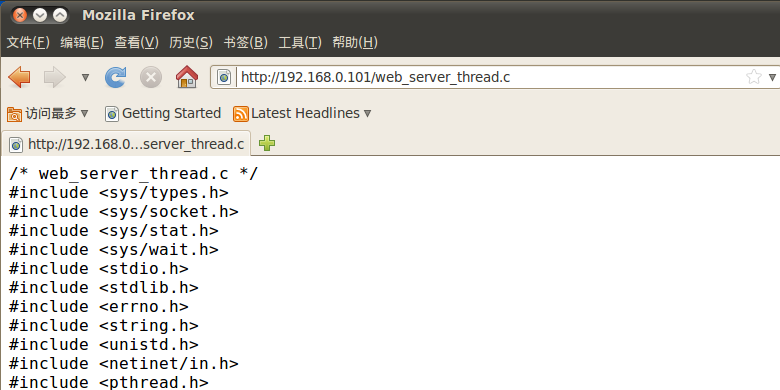
3)在宿主机上打开浏览器，输入 [http://192.168.0.101](http://192.168.0.101/) 查看结果。



4）宿主机上的浏览器去中输入 <http://192.168.0.101/cgi-bin/hello.cgi>

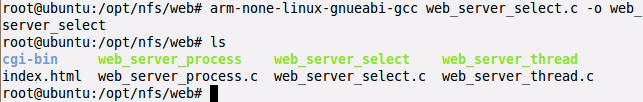


5)在一样的浏览器上输入 <http://192.168.0.101/web_server_thread.c>



实现I/O多路复用方式的Web服务器

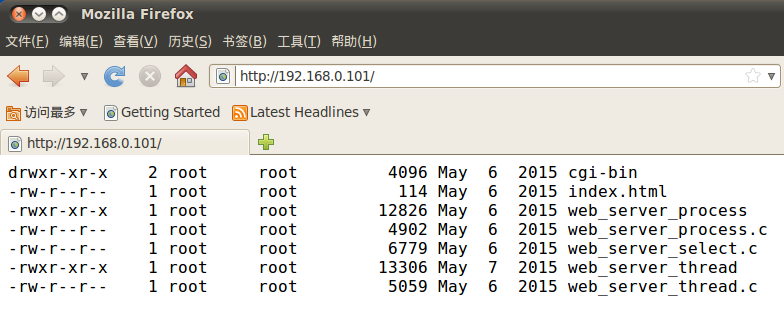
1)在宿主机上用arm-none-linux-gnueabi-gcc 命令来编译 web\_server\_select.c



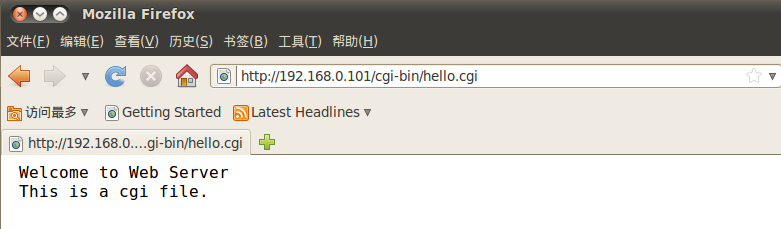
2)在目标机上运行刚编译的程序web\_server\_select，端口一样为 80

A description...

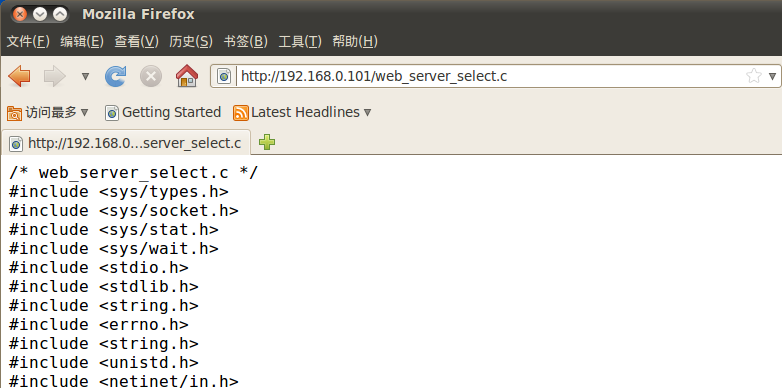
3)在宿主机上打开浏览器输入 [http://192.168.0.101](http://192.168.0.101/) 查看结果



4）宿主机上的浏览器去中输入 <http://192.168.0.101/cgi-bin/hello.cgi>



5)在一样的浏览器上输入 <http://192.168.0.101/web_server_select.c>



程序清单：

1. 多进程Web服务器

/\* web\_server\_process.c \*/

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <netinet/in.h>

#define BUFSIZE 1024

#define MAX\_QUE\_CONN\_NM 10

int start\_server(int portnum);

void sigchld\_handler(int sig);

int process\_request(char \*rq, int fd);

int resp\_cannot\_do(int fd);

int not\_exist(char \*f);

int resp\_do\_404(char \*item, int fd);

int is\_a\_dir(char \*f);

int resp\_do\_ls(char \*dir, int fd);

int is\_a\_cgi\_file(char \*f);

int resp\_do\_exec(char \*prog, int fd);

int resp\_do\_cat(char \*f, int fd);

void pack\_header(FILE \*fp, char \*extension);

char \*get\_filename\_extension(char \*f);

void ignore\_others(FILE \*fp);

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

{

int sock;

int portnum;

int fd;

pid\_t pid;

FILE \*fpin;

char request[BUFSIZE];

if (argc != 2) {

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

exit(-1);

}

portnum = atoi(argv[1]);

sock = start\_server(portnum);

signal(SIGCHLD, sigchld\_handler);

while(1) {

fd = accept(sock, NULL, NULL);

while ((pid = fork()) == -1);

if (pid == 0) {

fpin = fdopen(fd, "r");

memset(request, 0, sizeof(request));

fgets(request, BUFSIZE, fpin);

ignore\_others(fpin);

process\_request(request, fd);

fclose(fpin);

exit(0);

}

close(fd);

}

close(sock);

exit(0);

}

int start\_server(int portnum)

{

struct sockaddr\_in server\_sockaddr;

int sockfd;

int i = 1;

/\* create socket link \*/

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(-1);

}

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

server\_sockaddr.sin\_port = htons(portnum);

server\_sockaddr.sin\_addr.s\_addr = INADDR\_ANY;

bzero(server\_sockaddr.sin\_zero, 8);

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &i, sizeof(i));

if (bind(sockfd, (struct sockaddr \*)&server\_sockaddr, sizeof(struct sockaddr)) == -1) {

perror("bind");

exit(-1);

}

if (listen(sockfd, MAX\_QUE\_CONN\_NM) == -1)

{

perror("listen");

exit(-1);

}

return sockfd;

}

void sigchld\_handler(int sig)

{

while (waitpid(-1, 0, WNOHANG) > 0);

return;

}

int process\_request(char \*rq, int fd)

{

char cmd[BUFSIZE];

char arg[BUFSIZE];

strcpy(arg, "./");

if (sscanf(rq, "%s %s", cmd, arg + 2) != 2) {

return 1;

}

if (strcmp(cmd, "GET") != 0)

{

resp\_cannot\_do(fd);

} else if (not\_exist(arg)) {

resp\_do\_404(arg, fd);

} else if (is\_a\_dir(arg)) {

resp\_do\_ls(arg, fd);

} else if (is\_a\_cgi\_file(arg)) {

resp\_do\_exec(arg, fd);

} else {

resp\_do\_cat(arg, fd);

}

return 0;

}

int resp\_cannot\_do(int fd)

{

FILE \*fp = fdopen(fd, "w");

fprintf(fp, "HTTP/1.1 501 Not Implemented\r\n");

fprintf(fp, "Content-type: text/plain\r\n");

fprintf(fp, "\r\n");

fprintf(fp, "That command is not yet implemented\r\n");

fclose(fp);

return 0;

}

int not\_exist(char \*f)

{

struct stat info;

return (stat(f, &info) == -1);

}

int resp\_do\_404(char \*item, int fd)

{

FILE \*fp = fdopen(fd, "w");

fprintf(fp, "HTTP/1.1 404 Not Found\r\n");

fprintf(fp, "Content-type: text/plain\r\n");

fprintf(fp, "\r\n");

fprintf(fp, "The item you requested: %s\r\nis not found\r\n", item);

fclose(fp);

return 0;

}

int is\_a\_dir(char \*f)

{

struct stat info;

return (stat(f, &info) != -1 && S\_ISDIR(info.st\_mode));

}

int resp\_do\_ls(char \*dir, int fd)

{

FILE \*fp = fdopen(fd, "w");

pack\_header(fp, "text/plain");

fprintf(fp, "\r\n");

fflush(fp);

dup2(fd, 1);

dup2(fd, 2);

close(fd);

execlp("ls", "ls", "-l", dir, NULL);

perror(dir);

return 1;

}

int is\_a\_cgi\_file(char \*f)

{

return(strcmp(get\_filename\_extension(f), "cgi") == 0);

}

int resp\_do\_exec(char \*prog, int fd)

{

FILE \*fp = fdopen(fd, "w");

pack\_header(fp, NULL);

fflush(fp);

dup2(fd, 1);

dup2(fd, 2);

close(fd);

execl(prog, prog, NULL);

perror(prog);

return 1;

}

int resp\_do\_cat(char \*f, int fd)

{

int c;

char \*extension = get\_filename\_extension(f);

char \*content = "text/plain";

FILE \*fpsock, \*fpfile;

if (strcmp(extension, "html") == 0) {

content = "text/html";

} else if (strcmp(extension, "gif") == 0) {

content = "text/gif";

} else if (strcmp(extension, "jpg") == 0) {

content = "text/jpg";

} else if (strcmp(extension, "jpeg") == 0) {

content = "text/jpeg";

}

fpsock = fdopen(fd, "w");

fpfile = fopen(f, "r");

if (fpsock != NULL && fpfile != NULL) {

pack\_header(fpsock, content);

fprintf(fpsock, "\r\n");

while ((c = getc(fpfile)) != EOF) {

putc(c, fpsock);

}

fclose(fpfile);

fclose(fpsock);

return 0;

}

return 1;

}

void pack\_header(FILE \*fp, char \*extension)

{

fprintf(fp, "HTTP/1.1 200 OK\r\n");

if (extension)

fprintf(fp, "Content-type: %s\r\n", extension);

}

char \*get\_filename\_extension(char \*f)

{

return strrchr(f, '.') + 1;

}

void ignore\_others(FILE \*fp)

{

char buf[BUFSIZE];

fgets(buf, BUFSIZE, fp);

while(strcmp(buf, "\r\n"))

fgets(buf, BUFSIZE, fp);

return;

}

2. 多线程Web服务器

/\* web\_server\_thread.c \*/

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <netinet/in.h>

#include <pthread.h>

#define BUFSIZE 1024

#define MAX\_QUE\_CONN\_NM 10

void \*thread(void \*vargp);

int start\_server(int portnum);

int process\_request(int fd);

int resp\_cannot\_do(int fd);

int not\_exist(char \*f);

int resp\_do\_404(char \*item, int fd);

int is\_a\_dir(char \*f);

int resp\_do\_ls(char \*dir, int fd);

int is\_a\_cgi\_file(char \*f);

int resp\_do\_exec(char \*prog, int fd);

int resp\_do\_cat(char \*f, int fd);

void pack\_header(FILE \*fp, char \*extension);

char \*get\_filename\_extension(char \*f);

void ignore\_others(FILE \*fp);

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

{

int sock;

int portnum;

int \*fd;

pthread\_t tid;

if (argc != 2) {

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

exit(-1);

}

portnum = atoi(argv[1]);

sock = start\_server(portnum);

while(1) {

fd = malloc(sizeof(int));

\*fd = accept(sock, NULL, NULL);

pthread\_create(&tid, NULL, thread, fd);

}

close(sock);

exit(0);

}

int start\_server(int portnum)

{

struct sockaddr\_in server\_sockaddr;

int sockfd;

int i = 1;

/\* create socket link \*/

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(-1);

}

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

server\_sockaddr.sin\_port = htons(portnum);

server\_sockaddr.sin\_addr.s\_addr = INADDR\_ANY;

bzero(server\_sockaddr.sin\_zero, 8);

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &i, sizeof(i));

if (bind(sockfd, (struct sockaddr \*)&server\_sockaddr, sizeof(struct sockaddr)) == -1) {

perror("bind");

exit(-1);

}

if (listen(sockfd, MAX\_QUE\_CONN\_NM) == -1)

{

perror("listen");

exit(-1);

}

return sockfd;

}

void \*thread(void \*vargp)

{

int fd = \*((int \*)vargp);

pthread\_detach(pthread\_self());

free(vargp);

process\_request(fd);

close(fd);

return;

}

int process\_request(int fd)

{

char request[BUFSIZE], cmd[BUFSIZE], arg[BUFSIZE];

FILE \*fpin;

fpin = fdopen(fd, "r");

fgets(request, BUFSIZE, fpin);

ignore\_others(fpin);

strcpy(arg, "./");

if (sscanf(request, "%s %s", cmd, arg + 2) != 2) {

return 1;

}

if (strcmp(cmd, "GET") != 0)

{

resp\_cannot\_do(fd);

} else if (not\_exist(arg)) {

resp\_do\_404(arg, fd);

} else if (is\_a\_dir(arg)) {

resp\_do\_ls(arg, fd);

} else if (is\_a\_cgi\_file(arg)) {

resp\_do\_exec(arg, fd);

} else {

resp\_do\_cat(arg, fd);

}

fclose(fpin);

return 0;

}

int resp\_cannot\_do(int fd)

{

FILE \*fp = fdopen(fd, "w");

fprintf(fp, "HTTP/1.1 501 Not Implemented\r\n");

fprintf(fp, "Content-type: text/plain\r\n");

fprintf(fp, "\r\n");

fprintf(fp, "That command is not yet implemented\r\n");

fclose(fp);

return 0;

}

int not\_exist(char \*f)

{

struct stat info;

return (stat(f, &info) == -1);

}

int resp\_do\_404(char \*item, int fd)

{

FILE \*fp = fdopen(fd, "w");

fprintf(fp, "HTTP/1.1 404 Not Found\r\n");

fprintf(fp, "Content-type: text/plain\r\n");

fprintf(fp, "\r\n");

fprintf(fp, "The item you requested: %s\r\nis not found\r\n", item);

fclose(fp);

return 0;

}

int is\_a\_dir(char \*f)

{

struct stat info;

return (stat(f, &info) != -1 && S\_ISDIR(info.st\_mode));

}

int resp\_do\_ls(char \*dir, int fd)

{

pid\_t pid;

FILE \*fp = fdopen(fd, "w");

pack\_header(fp, "text/plain");

fprintf(fp, "\r\n");

fflush(fp);

while ((pid = fork()) == -1);

if (pid == 0) {

dup2(fd, 1);

dup2(fd, 2);

close(fd);

execlp("ls", "ls", "-l", dir, NULL);

perror(dir);

exit(-1);

}

waitpid(pid, NULL, 0);

return 0;

}

int is\_a\_cgi\_file(char \*f)

{

return(strcmp(get\_filename\_extension(f), "cgi") == 0);

}

int resp\_do\_exec(char \*prog, int fd)

{

pid\_t pid;

FILE \*fp = fdopen(fd, "w");

pack\_header(fp, NULL);

fflush(fp);

while ((pid = fork()) == -1);

if (pid == 0) {

dup2(fd, 1);

dup2(fd, 2);

close(fd);

execl(prog, prog, NULL);

perror(prog);

exit(-1);

}

waitpid(pid, NULL, 0);

return 0;

}

int resp\_do\_cat(char \*f, int fd)

{

int c;

char \*extension = get\_filename\_extension(f);

char \*content = "text/plain";

FILE \*fpsock, \*fpfile;

if (strcmp(extension, "html") == 0) {

content = "text/html";

} else if (strcmp(extension, "gif") == 0) {

content = "text/gif";

} else if (strcmp(extension, "jpg") == 0) {

content = "text/jpg";

} else if (strcmp(extension, "jpeg") == 0) {

content = "text/jpeg";

}

fpsock = fdopen(fd, "w");

fpfile = fopen(f, "r");

if (fpsock != NULL && fpfile != NULL) {

pack\_header(fpsock, content);

fprintf(fpsock, "\r\n");

while ((c = getc(fpfile)) != EOF) {

putc(c, fpsock);

}

fclose(fpfile);

fclose(fpsock);

return 0;

}

return 1;

}

void pack\_header(FILE \*fp, char \*extension)

{

fprintf(fp, "HTTP/1.1 200 OK\r\n");

if (extension)

fprintf(fp, "Content-type: %s\r\n", extension);

}

char \*get\_filename\_extension(char \*f)

{

return strrchr(f, '.') + 1;

}

void ignore\_others(FILE \*fp)

{

char buf[BUFSIZE];

fgets(buf, BUFSIZE, fp);

while(strcmp(buf, "\r\n"))

fgets(buf, BUFSIZE, fp);

return;

}

3. I/O多路复用方式的Web服务器

/\* web\_server\_select.c \*/

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <netinet/in.h>

#define BUFSIZE 1024

#define MAX\_QUE\_CONN\_NM 10

typedef struct {

int maxfd;

int maxi;

int nready;

int clientfd[FD\_SETSIZE];

int len[FD\_SETSIZE];

char request[FD\_SETSIZE][BUFSIZE];

fd\_set read\_set;

fd\_set ready\_set;

} pool;

int start\_server(int portnum);

void init\_pool(int sockfd, pool \*p);

void add\_client(int connfd, pool \*p);

void check\_clients(pool \*p);

int process\_request(char \*rq, int fd);

int resp\_cannot\_do(int fd);

int not\_exist(char \*f);

int resp\_do\_404(char \*item, int fd);

int is\_a\_dir(char \*f);

int resp\_do\_ls(char \*dir, int fd);

int is\_a\_cgi\_file(char \*f);

int resp\_do\_exec(char \*prog, int fd);

int resp\_do\_cat(char \*f, int fd);

void pack\_header(FILE \*fp, char \*extension);

char \*get\_filename\_extension(char \*f);

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

{

int sock;

int fd;

int portnum;

static pool pool;

if (argc != 2) {

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

exit(-1);

}

portnum = atoi(argv[1]);

sock = start\_server(portnum);

init\_pool(sock, &pool);

while(1) {

pool.ready\_set = pool.read\_set;

pool.nready = select(pool.maxfd + 1, &pool.ready\_set, NULL, NULL, NULL);

if ( pool.nready== -1) {

perror("select");

exit(-1);

}

if (FD\_ISSET(sock, &pool.ready\_set)) {

if ((fd = accept(sock, NULL, NULL)) == -1) {

perror("accept");

close(sock);

exit(-1);

}

add\_client(fd, &pool);

}

check\_clients(&pool);

}

}

int start\_server(int portnum)

{

struct sockaddr\_in server\_sockaddr;

int sockfd;

int i = 1;

/\* create socket link \*/

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(-1);

}

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

server\_sockaddr.sin\_port = htons(portnum);

server\_sockaddr.sin\_addr.s\_addr = INADDR\_ANY;

bzero(server\_sockaddr.sin\_zero, 8);

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &i, sizeof(i));

if (bind(sockfd, (struct sockaddr \*)&server\_sockaddr, sizeof(struct sockaddr)) == -1) {

perror("bind");

exit(-1);

}

if (listen(sockfd, MAX\_QUE\_CONN\_NM) == -1)

{

perror("listen");

exit(-1);

}

return sockfd;

}

void init\_pool(int sockfd, pool \*p)

{

int i;

p->maxi = -1;

for(i = 0; i < FD\_SETSIZE; i++)

p->clientfd[i] = -1;

p->maxfd = sockfd;

FD\_ZERO(&p->read\_set);

FD\_SET(sockfd, &p->read\_set);

}

void add\_client(int connfd, pool \*p)

{

int i;

p->nready--;

for (i = 0; i < FD\_SETSIZE; i++)

if (p->clientfd[i] < 0) {

p->clientfd[i] = connfd;

memset(p->request[i], 0, BUFSIZE);

p->len[i] = 0;

FD\_SET(connfd, &p->read\_set);

if (connfd > p->maxfd)

p->maxfd = connfd;

if (i > p->maxi)

p->maxi = i;

break;

}

if (i == FD\_SETSIZE)

printf("add\_client error: Too many clients\n");

}

void check\_clients(pool \*p)

{

int i;

int connfd;

int n;

int ret;

char buf[BUFSIZE];

char \*q;

for (i=0; (i<=p->maxi)&&(p->nready>0); i++) {

connfd = p->clientfd[i];

if ((connfd > 0) && (FD\_ISSET(connfd, &p->ready\_set))) {

p->nready--;

ret = read(connfd, p->request[i]+p->len[i], BUFSIZE);

if (ret == -1) {

perror("read error");

close(connfd);

continue;

} else if (ret == 0) {

printf("close\n");

if (close(connfd) < 0) {

perror("close connfd failed");

exit(-1);

}

FD\_CLR(connfd, &p->read\_set);

p->clientfd[i] = -1;

} else {

p->len[i] += ret;

q = &p->request[i][p->len[i]-4];

if (strcmp(q, "\r\n\r\n") == 0) { //request[i] end

q = strchr(p->request[i], '\r');

\*q = '\0';

process\_request(p->request[i], connfd);

close(connfd);

FD\_CLR(connfd, &p->read\_set);

p->clientfd[i] = -1;

}// if(strcmp(q, "\r\n\r\n") == 0)

}//ret

}//if((connfd > 0) && (FD\_ISSET(connfd, &p->ready\_set)))

}//for

}

int process\_request(char \*rq, int fd)

{

char cmd[BUFSIZE], arg[BUFSIZE];

strcpy(arg, "./");

if (sscanf(rq, "%s %s", cmd, arg + 2) != 2) {

return 1;

}

if (strcmp(cmd, "GET") != 0)

{

resp\_cannot\_do(fd);

} else if (not\_exist(arg)) {

resp\_do\_404(arg, fd);

} else if (is\_a\_dir(arg)) {

resp\_do\_ls(arg, fd);

} else if (is\_a\_cgi\_file(arg)) {

resp\_do\_exec(arg, fd);

} else {

resp\_do\_cat(arg, fd);

}

return 0;

}

int resp\_cannot\_do(int fd)

{

FILE \*fp = fdopen(fd, "w");

fprintf(fp, "HTTP/1.1 501 Not Implemented\r\n");

fprintf(fp, "Content-type: text/plain\r\n");

fprintf(fp, "\r\n");

fprintf(fp, "That command is not yet implemented\r\n");

fclose(fp);

return 0;

}

int not\_exist(char \*f)

{

struct stat info;

return (stat(f, &info) == -1);

}

int resp\_do\_404(char \*item, int fd)

{

FILE \*fp = fdopen(fd, "w");

fprintf(fp, "HTTP/1.1 404 Not Found\r\n");

fprintf(fp, "Content-type: text/plain\r\n");

fprintf(fp, "\r\n");

fprintf(fp, "The item you requested: %s\r\nis not found\r\n", item);

fclose(fp);

return 0;

}

int is\_a\_dir(char \*f)

{

struct stat info;

return (stat(f, &info) != -1 && S\_ISDIR(info.st\_mode));

}

int resp\_do\_ls(char \*dir, int fd)

{

pid\_t pid;

FILE \*fp = fdopen(fd, "w");

pack\_header(fp, "text/plain");

fprintf(fp, "\r\n");

fflush(fp);

while ((pid = fork()) == -1);

if (pid == 0) {

dup2(fd, 1);

dup2(fd, 2);

close(fd);

execlp("ls", "ls", "-l", dir, NULL);

perror(dir);

exit(-1);

}

waitpid(pid, NULL, 0);

return 0;

}

int is\_a\_cgi\_file(char \*f)

{

return(strcmp(get\_filename\_extension(f), "cgi") == 0);

}

int resp\_do\_exec(char \*prog, int fd)

{

pid\_t pid;

FILE \*fp = fdopen(fd, "w");

pack\_header(fp, NULL);

fflush(fp);

while ((pid = fork()) == -1);

if (pid == 0) {

dup2(fd, 1);

dup2(fd, 2);

close(fd);

execl(prog, prog, NULL);

perror(prog);

exit(-1);

}

waitpid(pid, NULL, 0);

return 0;

}

int resp\_do\_cat(char \*f, int fd)

{

int c;

char \*extension = get\_filename\_extension(f);

char \*content = "text/plain";

FILE \*fpsock, \*fpfile;

if (strcmp(extension, "html") == 0) {

content = "text/html";

} else if (strcmp(extension, "gif") == 0) {

content = "text/gif";

} else if (strcmp(extension, "jpg") == 0) {

content = "text/jpg";

} else if (strcmp(extension, "jpeg") == 0) {

content = "text/jpeg";

}

fpsock = fdopen(fd, "w");

fpfile = fopen(f, "r");

if (fpsock != NULL && fpfile != NULL) {

pack\_header(fpsock, content);

fprintf(fpsock, "\r\n");

while ((c = getc(fpfile)) != EOF) {

putc(c, fpsock);

}

fclose(fpfile);

fclose(fpsock);

return 0;

}

return 1;

}

void pack\_header(FILE \*fp, char \*extension)

{

fprintf(fp, "HTTP/1.1 200 OK\r\n");

if (extension)

fprintf(fp, "Content-type: %s\r\n", extension);

}

char \*get\_filename\_extension(char \*f)

{

return strrchr(f, '.') + 1;

}