مقدمه ای بر Socket programming

گاهی اوقات پیش میاد که یکی از دوستان فایل بزرگی رو از من بخواد که به خاطر حجم زیاد نتونم به ایمیل attach کنمش. این جور مواقع معمولا فایل رو کپی میکنم تو root دایکتوریه apache و لینکش رو میدم به دوستم تا از رو سیستم من دانلودش کنه. حالا ممکنه همون لحظه فرصت دانلود کردن نداشته باشه و نگهداره بعدا دانلود کنه و من هم که طبیعتا فراموش میکنم که فایل رو از root دایکتوریه apache پاک کنم. بعد از مدتی این ها جمع میشن روی هم و من نیاز پیدا میکنم که چند ماه یکبار یه سری به root دایکتوری apache بزنم و فایل های اضافه رو پاک کنم.
فکر کردم اگه یه برنامه با C بنویسم که فقط یک فایل رو، تنها یک بار دانلود بده و تموم بشه هم از دردسره بزرگ شدن root دایرکتوری apache خلاص میشم، هم یه مثال خیلی ساده خواهم داشت برای آموزش شروع برنامه نوسی socket ها در C .
البته اینکار رو میشه با netcat و perl و python و خیلی ابزار های دیگه، بسیار راحت تر از این انجام داد. منتها چون زبان C همیشه بعنوان یک hobby برای من مطرح بوده، دنبال بهانه میگردم که برنامه C بنویسم.
سعی میکنم source رو خیلی خلاصه ننویسم که توضیح دادنش سخت بشه و امکانات خاصی هم اضافه نمیکنم که بیخود source طولانی نشه.
اسمش رو میگذارم sendtoweb ، و می خوام تنها پارامتری که میگیره، آدرس فایلی باشه که قراره دانلود بشه. از پورت 8181 هم به شکل built-in استفاده میکنم که ضمن اجرای برنامه نمیشه تغییرش داد.
قراره من اینطوری اجراش کنم :

```
./sendtoweb /home/pejman/test.pdf
```

و با فرض اینکه آی پی valid من باشه 83.170.42.33 ، کسی که فایل رو قراره دانلود کنه، از این لینک استفاده میکنه :

http://83.170.42.33:8181/test.pdf

بریم سراغ source . اول باید چک کنم که حتما اسم فایلی که قراره دانلود بشه به شکل پارامتر به برنامه پاس داده شده باشه:

```
char *me = basename(argv[0]);
if (argc < 2) {
	fprintf(stderr, "Usage:\n %s <file address>\n", me);
	exit(1);
}
```

توجه کنید که برای نمایش خطا از fprintf استفاده کردم و خروجی رو فرستادم به stderr . با این روش، این امکان به کابر داده میشه که خروجی اصلی برنامه رو از خروجی خطا ها جدا کنه. مثلا اگه نخواست پیغامهای خطا رو ببینه برنامه رو به این شکل اجرا کنه:

```
# ./sendtoweb 2> /dev/null
```

فایلی رو که اسمش به عنوان پارامتر پاس داده شده، برای خوندن باز میکنم :

```
int file_fd;
char *file_addr = argv[1];
file_fd = open(file_addr, O_RDONLY);
```

```
if(file_fd == -1) {
	fprintf(stderr, "Can not open file: %s", file_addr);
	exit(1);
}
```

با تابع getaddrinfo باید یک struct رو پر کنم که توابع بعدی ازش به عنوان آدرس استفاده خواهند کرد:

```
#define PORT "8181"
int status;
struct addrinfo hints, *servinfo;
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
status = getaddrinfo(NULL, PORT, &hints, &servinfo);
if(status != 0) {
	fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
	exit(1);
}
```

تابع getaddrinfo از hints به عنوان ورودی استفاده میکنه و نتیجه رو در servinfo ذخیره میکنه. با memset تمام hints رو با 0 پر میکنم. AF_INET یعنی Ipv4 می خوام استفاده کنم.
SOCK_STREAM یعنی ارتباط با کاربر براساس TCP باشه. AI_PASSIVE یعنی روی همه ی IP های سیستم listening خواهم خواست. در اجرای getaddrinfo اگه اولین پارامتر NULL نباشه، AI_PASSIVE در نظر گرفته نمیشه و فقط روی IP ی که این پارامتر مشخص میکنه listening درخواست میشه. توجه داشته باشید که getaddrinfo تنها servinfo رو برای استفاده در توابع بعدی آماده میکنه و هیچ کاری در زمینه ارتباط با شبکه انجام نمیده.

از اون جایی که همه چیز در linux یک فایله، برای ارتباط با شبکه هم یک descriptor لازم خواهیم داشت که مثل file descriptor کار میکنه و برای ورودی و خروجی قراره ازش استفاده کنیم. descriptor رو با کمک تابع socket به connection اختصاص میدیم:

```
int listen_fd;
listen_fd = socket(servinfo->ai_family, servinfo->ai_socktype,
                                                servinfo->ai_protocol);
if(listen_fd == -1) {
	perror("socket error: ");
	exit(1);
}
```

توجه کنید که servinfo قبلا با تابع getaddrinfo پر شده بود و و تابع socket از همون مقادیر استفاده میکنه.

بعد از پایان یک ارتباط TCP یه مدتی پورته اون ارتباط در وضعیت TIME_WAIT باقی میمونه که اگه از packet های connection چیزی جا مونده باشه، همه رو دریافت کنه. در این حالت اگه سرور رو دوباره اجرا کنیم پیغام "Address already in use " رو میبینیم و اجرا متوقف میشه. برای اینکه این

مشکل پیش نیاد با تابع setsockopt یه آپشنه SO_REUSEADDR روی socket ست میکنم تا پورت حتی در وضعیت TIME_WAIT هم قابل استفاده باشه:

```
int status, yes = 1;
status = setsockopt(listen_fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof yes);
if(status == -1) {
	perror("setsockopt error: ");
	exit(1);
}
```

نوبت می رسه به اختصاص پورت به connection ، اگه پورت با یه پروسس دیگه درگیر باشه تو این مرحله error میبینیم :

```
int status;
status = bind(listen_fd, servinfo->ai_addr, servinfo->ai_addrlen);
if(status == -1) {
	perror("bind error: ");
	exit(1);
}
```

توجه داشته باشید که شماره پورت رو قبلا توسط تابع getaddrinfo نوشته بودیم تو servinfo و اینجا کافیه فقط از servinfo استفاده کنیم.

حالا وقتشه که listening رو شروع کنیم و منتظر بمونیم که client درخواست بفرسته :

```
#define BACKLOG 1
int status;
status = listen(listen_fd, BACKLOG);
if(status  == -1) {
	perror("listen error: ");
	exit(1);
}
```

BACKLOG تعداد connection هایی رو که قراره تو صف بمونن تا نوبتشون برسه رو مشخص میکنه.

بلافاصله بعد از رسیدن request ، تابع accept اجرا میشه و یه socket جدید به این ارتباط اختصاص میده. رسم بر اینه که سرور ها در این لحظه یه پروسس child رو fork میکنن و با پاس دادن این socket جدید به پروسس child بقیه ی ارتباط با client رو به عهده ی اون می گذارن و خودشون مجددا به listening ادامه میدن تا در خواستهای بعدی رو دریافت کنن. منتها در این مورد خاص که ما قراره فقط به یک client سرویس بدیم، نیازی به child پروسس نخواهیم داشت:

```
socklen_t addr_size;
static struct sockaddr_storage client_addr;
addr_size = sizeof(client_addr);
accept_fd = accept(listen_fd, (struct sockaddr *)&client_addr, &addr_size);
if(accept_fd == -1) {
	perror("accept error: ");
	exit(1);
```

```
}
```

بعد از اجرای accept میتونم با تابع recv درخواستی رو که از طرف مرورگر client برام فرستاده شده بخونم. آخرش یه صفر هم اضافه میکنم که بعدا بشه مثل یک string ازش استفاده کرد:

```
int len;
len = recv(accept_fd, buffer, BUFSIZE, 0);
if(len == 0 || len == -1) {
	fprintf(stderr, "Failed to read browser request");
	exit(1);
}
if(len > 0 && len < BUFSIZE)
	buffer[len] = 0;
else
	buffer[0] = 0;
```

اگه تا به حال سعی کرده باشید با telnet به یک web server وصل بشید، حتما می دونید ساده ترین شکل HTTP request اینطورپه:

```
GET / HTTP/1.1
```

بعد از GET حتما یک space هست، بعد url نوشته میشه بعد دوباره یه space و در نهایت HTTP/1.1 . پس اگه browser برای دانلود فایل test.pdf درخواست فرستاده باشه، request به این شکل دست سرور میرسه :

```
 GET /test.pdf HTTP/1.1
```

چک میکنم که حتما GET و space ، ابتدای request باشن:

```
if(strncmp(buffer,"GET ",4) && strncmp(buffer,"get ",4)) {
	printf("GET method supported only.\n");
	exit(1);
}
```

محض تفریح، دوست دارم request رو ببینم، دیدنش برام جالبه:

```
fprintf(stdout, "Request:\n---\n%s\n---\n",buffer);
```

می خوام در request همه چیز رو بعد از url حذف کنم. یعنی HTTP/1.1 یا هر چیزه دیگه ای که هست. لازمش ندارم:

```
for(i = 4; i < BUFSIZE; i++) {
	if(buffer[i] == ' ') {
		buffer[i] = 0;
		break;
	}
}
```

چک میکنم که url با اسم فایلی که من قراره دانلود بدم یکی باشه، اینطوری خیالم راحته فقط کسی که اسم فایل رو می دونه، می تونه دانلودش کنه. اگه url اشتباه بود یه Not Found برای مرورگره client میفرستم :

```
if(strcmp(&buffer[5],filename)) {
	fprintf(stderr, "Filename not matched: %s\n",&buffer[5]);
	sprintf(buffer, "HTTP/1.0 404 Not Found\r\n"
			"Content-Type: text/html\r\n\r\n"
			"<HTML><BODY>"
			"<b>Not Found !!!</b>"
			"</BODY></HTML>\r\n");
	send(accept_fd, buffer, strlen(buffer), 0);
	exit(1);
}
```

حالا که url هم درست بوده اول HTTP header رو میفرستم:

```
sprintf(buffer,"HTTP/1.0 200 OK\r\nContent-Type: application\r\n\r\n");
send(accept_fd, buffer, strlen(buffer), 0);
```

و در نهایت فایل رو به شکل block های 8 کیلوبایتی send میکنم :

```
while((len = read(file_fd, buffer, BUFSIZE)) > 0)
	send(accept_fd, buffer, len, 0);
```

با این که ضمن exit شدن برنامه ، فایل descriptor ها خود به خود close میشن، ولی برای تمیز بودن code همه رو close میکنم :

```
	close(listen_fd);
	close(accept_fd);
	close(file_fd);
```

حالا میتونید همه ی source یکجا و مرتب ببنید :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <libgen.h>

#define BUFSIZE 8096
#define BACKLOG 1      /* pending connections queue length */
#define PORT "8181"

int main(int argc, char **argv)
{
        int i, len, status, listen_fd, accept_fd, file_fd, yes = 1;
        socklen_t addr_size;
        char buffer[BUFSIZE+1];
        char *me = basename(argv[0]);
        char *filename = basename(argv[1]);
        char *file_addr = argv[1];
        struct addrinfo hints, *servinfo;
        /* Static variables will fill with zeros */
        static struct sockaddr_storage client_addr;

        /* Check syntax */
        if (argc < 2) {
                fprintf(stderr, "Usage:\n %s <file address>\n", me);
                exit(1);
        }

        /* Open the file for reading */
        file_fd = open(file_addr, O_RDONLY);
        if(file_fd == -1) {
                fprintf(stderr, "Can not open file: %s", file_addr);
                exit(1);
        }

        /* Load up address structs */
        memset(&hints, 0, sizeof hints); /* make sure the struct is empty */
        hints.ai_family = AF_INET;       /* use IPv4 */
        hints.ai_socktype = SOCK_STREAM; /* TCP stream sockets */
        hints.ai_flags = AI_PASSIVE;     /* fill in my IP */
        status = getaddrinfo(NULL, PORT, &hints, &servinfo);
        if(status != 0) {
                fprintf(stderr, "getaddrinfo error: %s\n",
                                        gai_strerror(status));
                exit(1);
        }

        /* Setup a network socket */
        listen_fd = socket(servinfo->ai_family, servinfo->ai_socktype,
```

```
                                                servinfo->ai_protocol);
if(listen_fd == -1) {
        perror("socket error: ");
        exit(1);
}

/* if port is in the TIME_WAIT state, go ahead and reuse it anyway. */
status = setsockopt(listen_fd, SOL_SOCKET, SO_REUSEADDR,
                                        &yes, sizeof yes);
if(status == -1) {
        perror("setsockopt error: ");
        exit(1);
}

/* Bind it to the port */
status = bind(listen_fd, servinfo->ai_addr, servinfo->ai_addrlen);
if(status == -1) {
        perror("bind error: ");
        exit(1);
}

/* Start listening to the port */
status = listen(listen_fd, BACKLOG);
if(status  == -1) {
        perror("listen error: ");
        exit(1);
}

/* Accept an incoming connection */
addr_size = sizeof(client_addr);
accept_fd = accept(listen_fd, (struct sockaddr *)&client_addr,
                                                        &addr_size);
if(accept_fd == -1) {
        perror("accept error: ");
        exit(1);
}

/* Read browser request */
len = recv(accept_fd, buffer, BUFSIZE, 0);
if(len == 0 || len == -1) {
        fprintf(stderr, "Failed to read browser request");
        exit(1);
}
if(len > 0 && len < BUFSIZE)
        buffer[len] = 0;
else
        buffer[0] = 0;
```

```
	/* Check HTTP method */
	if(strncmp(buffer,"GET ",4) && strncmp(buffer,"get ",4)) {
		printf("GET method supported only.\n");
		exit(1);
	}

	/* Show browser request */
	fprintf(stdout, "Request:\n---\n%s\n---\n",buffer);

	/* buffer shuold look like 'GET URL HTTP/1.0' */
	/* remove everything after URL */
	for(i = 4; i < BUFSIZE; i++) {
		if(buffer[i] == ' ') {
			buffer[i] = 0;
			break;
		}
	}

	/* Check requested url matchs with filename */
	if(strcmp(&buffer[5],filename)) {
		fprintf(stderr, "Filename not matched: %s\n",&buffer[5]);
		sprintf(buffer, "HTTP/1.0 404 Not Found\r\n"
				"Content-Type: text/html\r\n\r\n"
				"<HTML><BODY>"
				"<b>Not Found !!!</b>"
				"</BODY></HTML>\r\n");
		send(accept_fd, buffer, strlen(buffer), 0);
		exit(1);
	}

	/* Write HTTP header to socket */
	sprintf(buffer,"HTTP/1.0 200 OK\r\nContent-Type: application\r\n\r\n");
	send(accept_fd, buffer, strlen(buffer), 0);

	/* Send file in 8KB blocks */
	while((len = read(file_fd, buffer, BUFSIZE)) > 0)
		send(accept_fd, buffer, len, 0);

	/* Cleanup */
	close(listen_fd);
	close(accept_fd);
	close(file_fd);

	printf("Successfully sent: %s\n", file_addr);
}
```

منابع :

http://beej.us/guide/bgnet/
http://www.ibm.com/developerworks/systems/library/es-nweb/

آدرس این نوشته در فرمت های دیگر:

odt: http://pmoghadam.com/homepage/Pages/Drafts/socket-sendtoweb.odt

pdf: http://pmoghadam.com/homepage/Pages/Drafts/socket-sendtoweb.pdf

پژمان مقدم
زنجان - 80/09/28