آزمایشگاه سیستم عامل

دانشکده برق و کامپیوتر –
دانشگاه صنعتی اصفهان

پاییز ۱۳۹۴

دستور کار جلسه پنجم

دستور کار جلسه پنجم ۱

فراخوانی های سیستمی برای مدیریت Socket، Pipe و Named-Pipe ۲

مروری کلی بر روش های ارتباط میان فرآیندها (Pipe-Socket-Signal) ۳

Pipe ۳

Named-Pipe ۳

Socket ۳

Signal ۴

مثال ها ۵

Pipe ۵

Named-Pipe ۶

TCP Client ۸

TCP Server ۱۰

دستورکار جلسه پنجم ۱۲

فراخوانی های سیستمی برای مدیریت Socket، Pipe و Named-Pipe

در این دستورکار توضیحاتی درباره ارتباط بین فرآیندی ارائه شده است. همچنین فراخوانی های سیستمی برای ایجاد و مدیریت فراخوانی های سیستمی برای مدیریت Socket، Pipe و Named-Pipe بررسی می شوند.

تمامی توابع و فراخوانی های مطرح شده در این دستور کار از دو آدرس زیر آورده شده اند، برای مطالعه جزییات به آنها به صفحات راهنما و یا آدرس های آورده شده مراجعه کنید :

http://www.minix3.org/manpages

http://linux.die.net/man

مروری کلی بر روش های ارتباط میان فرآیندها (Pipe-Socket-Signal)

Pipe

- Pipe عملکردی شبیه به File دارد با این تفاوت که Pipe در حافظه ی RAM قرار می گیرد ولی File بر روی حافظه ی دائمی (I/O) قرار می گیرد، از این رو Pipe از File سریعتر عمل می کند.
- برقراری ارتباط بین فرآیند والد و فرآیند فرزند و همچنین برقراری ارتباط بین فرآیندهای فرزند با یکدیگر .
- در هر دو حالت ذکر شده Pipe ها، توسط والد ایجاد می شوند .
- Pipe در مقایسه با socket از سرعتی بیشتر برخوردارست، در عین حال قابلیت های کنترلی کمتری دارد .
- مدیریت pipe ها به دلیل آنکه همواره بایستی توسط یک فرآیند والد ایجاد شود پیچیدگی بیشتری دارد .

Named-Pipe

- خط لوله نامگذاری شده عملکردی مشابه خط لوله دارد با این تفاوت که فرآیندها با داشتن نام آن می توانند به آن دسترسی داشته باشند و حتما رابطه ی والد–فرزند بین آنها برقرار نیست.
- این خط لوله برای ارتباط بین فرآیندهایی که لزوما رابطه ی والد–فرزند ندارند مناسب است.

Socket

- ایجاد ارتباط بین دو فرآیند با استفاده از IP Address و Port Number .
- اغلب هنگامی از این روش استفاده می شود که دو فرآیند ارتباط والد/فرزند نداشته باشند و یا دو فرآیند قصد ارتباط روی شبکه را داشته باشند.
- سوکت های تعریف شده در POSIX انواع متعددی دارند که از جمله پرکاربردترین آنها سوکت های TCP/UDP/UNIX می باشند.
- نوع سوکت استفاده شده، سرعت و کارایی(performance) آن را تعیین می کند، برای مثال استفاده از سوکت های نوع UDP یا TCP باعث کمتر شدن سرعت ارتباط می شود ولی استفاده از سوکت های نوع UNIX برای دو فرآیند که در یک CPU قرار دارند سرعت و کارایی بالایی خواهد داشت .(سرعتی قابل قیاس با سرعت PIPE)

Signal

- ارتباط بین فرآیندها از راه کنترل رخدادهای خاص و تعریف رفتار فرآیند در قبال رخدادهای تعریف شده .
- برخلاف socket و pipe نمی توان مقداری را از طریق signal منتقل کرد، این روش تنها برای آگاهی از اتفاق افتادن یک رخداد خاص به کار می رود .
- Signal ها نسبت به pipe و socket سرعت کمتر و پیچیدگی های بیشتری دارند و به همین دلیل بایستی با احتیاط به کار برده شوند.

مثال ها

Pipe

```
/*
making a pipe between parent and child, sending messages from parent to child
*/

#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main(){
        int fd[2];
        char buffer[256];
        int x=pipe(fd);
        printf("fd=%d\n",fd[0]);
        pid_t pid;
        pid=fork();
        if(pid==0) //in child
        {
                while(1)
                {
                printf("in child\n");
                read(fd[0],buffer,255);
                printf("%s\n",buffer);
                }
        }
        else// in parent
        {
                while(1)
                {
                        printf("in parent\n");
                        sprintf(buffer,"message to child");
                        write(fd[1],buffer,255);
                        sleep(2);
                }
        }
        return 0;
}
```

Named-Pipe

```
/*
making a pipe between parent and child and sending messages from parent to child
*/

#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/stat.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>

int main(){
        int pipe, inChild;
        int tmp;
        char path[20];
        sprintf(path,"1.pipe");
        printf("%s\n",path);

        //making the named-pipe
        mkfifo(path,0777);

        char buffer[256];
        bzero(buffer,256);
        pid_t pid;
        pid=fork();
        inChild=0;
        if (pid==0)
                inChild=1;


        while(inChild==1)
        {
                /*
                   operations on named-pipe are similar to a file
                   we open the named-pipe
                */
                pipe=open(path,O_RDONLY|O_NONBLOCK);
                read(pipe,buffer,255);
                printf("child <- %s\n",buffer);
                bzero(buffer,256);
                sleep(1);
        }
```

```
        tmp=0;
        pipe=open(path,O_WRONLY);
        while(inChild==0)
        {
                sprintf(buffer,"%d",tmp);
                tmp++;
                printf("parent -> %s", buffer);
                write(pipe,buffer,strlen(buffer));
                sleep(1);
        }
        return 0;
}
```

TCP Client

```
/*
example of TCP socket client
client sends message "message from client" to server every 1 second
*/

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <netdb.h>

int main()
{
        char buffer[256];
        int socket1, portNo;
        struct hostent * server;
        char ipv4[32];
        sprintf(ipv4,"127.0.0.1");
        server=gethostbyname(ipv4);

        struct sockaddr_in server_address;
        portNo=6000;
        socket1=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

        server_address.sin_family=AF_INET;
        server_address.sin_port=htons(portNo);
        bcopy(   (char *)server->h_addr,
                 (char *)&server_address.sin_addr.s_addr,
                 server->h_length
             );

        connect(socket1,(struct sockaddr *)&server_address,sizeof(server_address));
        sprintf(buffer,"message from client");
        while(1)
        {
                write(socket1,buffer,strlen(buffer));
                sleep(1);
        }
        return 0;
}
```

مقایسه برنامه نویسی سوکت سمت سرور و کلاینت:

توابع موجود در سمت سرور اندکی با توابع موجود در سمت کلاینت متفاوت است..

در سمت کلاینت تنها سوکت ساخته شده و به سوکت سرور متصل میشود اما در سمت سرور پس از ساخت سوکت از تابع **bind** استفاده میشود ..این تابع به این منظور استفاده میشود که به سوکتی که باز کرده ایم یک شماره پورت نسبت میدهیم در واقع به سیستم عامل اعلام میکنیم که سوکتهایی که آدرس مقصدشان با آدرس مورد نظر مطابقت دارد به سمت سرور بیاید.

تابع دیگر تابع **listen** است که با استفاده از این تابع به سیستم عامل اعلام میکنیم کارش را برای پذیرش تقاضای ارتباط TCP شروع کند.همچنین فضای لازم جهت ارسال و دریافت داده ها ایجاد میکند...با استفاده از این تابع اعلام میکنیم که سرور چه تعداد درخواست میتواند از سمت کلاینت قبول کند.

همچنین در سمت سرور سوکت جدیدی تحت عنوان **acceptedsocket** ایجاد میشودکه جهت ارسال و دریافت اطلاعات از کلاینت از این سوکت استفاده میشود.در واقع به این ترتیب به سیستم عامل اعلام میکنیم یکی از ارتباطات معلق را به برنامه ما معرفی کند.

TCP Server

```
/*
example of TCP socket server
server listens to incoming clients and acceptes maximum 5 sockets per second
server checks accepted socket every 1 second and receives the message
*/

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <netdb.h>

int main(){
        char buffer[256];
        int socket1, portNo, clientLength;
        int acceptedSocket;

        // defining a host entity to store information of server
        struct hostent * server;
        char ipv4[32];
        sprintf(ipv4,"127.0.0.1");

        // initializing address for server entity
        server=gethostbyname(ipv4);

        // server_address = explicit address of server
        //client_address = client information
        struct sockaddr_in server_address, client_address ;
        clientLength=sizeof(client_address);

        portNo=6000;     // server listens to this port number

        // making socket family = AF_INET, type = SOCK_STREAM , protocol = TCP
        socket1=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

        //initializing server addres
        server_address.sin_family=AF_INET;
        server_address.sin_port=htons(portNo);
        server_address.sin_addr.s_addr=INADDR_ANY;

        //binding socket to server address
        bind ( socket1, (struct sockaddr*) &server_address,sizeof(server_address));

        //listening to incoming requests from clients
        //backlog(maximum number of connections per second) = 5
        listen(socket1,5);


        acceptedSocket = accept(  socket1,
                          (struct sockaddr * )&client_address,
```

```
                        &clientLength
                                );
        bzero(buffer,256);
        int n;
        while(1)
        {
                read(acceptedSocket, buffer, 256);
                printf("[%s@%d]# %s\n", ipv4, portNo, buffer);
                bzero(buffer,256);
                sleep(1);
        }
        return 0;
}
```