دستور کار جلسه چهارم

فراخوانی های سیستمی برای مدیریت فرآیندها و مدیریت زمان ۳

فراخوانی های سیستمی برای مدیریت زمان ۴

فراخوانی های سیستمی برای ایجاد و مدیریت فرآیندها ۴

مثالها ۶

دستور کار جلسه چهارم ۹

آشنایی با مفهوم forkدر سیستم عامل

هنگامی که یک کلاینت یک صفحه را از یک سرور وب درخواست میکند اگر سرور پاسخ آن کلاینت را بدهد و سپس به درخواست کلاینت دیگری گوش کند در آن فاصله زمانی بسیاری از درخواستهابه سرور بی پاسخ مانده و شکست میخورد بنابراین اگر سرور بتواندهر درخواستی که دریافت میکندرا به شکل موازی با درخواستهای دیگر پاسخ دهدمیتواند بسیار کاراتر عمل کندو دریک زمان به تعداد بیشتری کلاینت سرویس دهد...اینجاست که مفهوم fork معنا پیدا کرده و ملموس تر میشود..

تابع fork: هنگامی که برنامه شما این تابع را صدا میکندسیستم عامل برنامه شما را که در قالب یک پروسه اجرا میشود را به دو پروسه دقیقا همانندتبدیل میکند که بصورت موازی اجرا میشود ..خروجی این تابع یا صفر است یا یک عدد..برنامه شما با بررسی خروجی این تابع متوجه میشود که الان در کدام پروسه هست و با توجه به این موضوع به کاری متفاوت بپردازد..اگر خروجی این تابع صفر بود پروسه فرزند واگر عدد بود پروسه پدر اجرا میشود...بعد از اینکه پروسه فرزند ایجاد شد هر دو پروسه دستورات بعد از fork را اجرا میکنند....

فراخوانی های سیستمی برای مدیریت فرآیندها و مدیریت زمان

در این دستورکار فراخوانی های سیستمی برای ایجاد و مدیریت فرآیندها و مدیریت زمان 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

فراخوانی های سیستمی برای مدیریت زمان

این تابع زمان را به ما برمیگرداند..

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

The function gettimeofday() can get the time as well as a timezone

تابع time زمان را در متغیر second برمیگرداند اگر ورودی null باشد زمان جاری محاسبه شده و در متغیر second برمیگردد

```
seconds = time (&seconds)
```

Get the elapsed time since Jan. 1, 1970

فراخوانی های سیستمی برای ایجاد و مدیریت فرآیندها

```
pid_t fork(void);
```

Create a child process identical to the parent

این تابع برای بررسی وضعیت فرزند مشخص شده با pid به کار میرود..اگر خروجی این تابع صفر بود یعنی فرزند آزاد نشده و اگر مخالف صفر بود یعنی فرزند آزاد شده است.

```
pid_t waitpid(pid_t pid, int *status, int options);
```

Wait for a child to terminate

```
exit (status)
```

Terminate process execution and return status

این تابع برای دریافت شماره پروسسه فرآیند مورد استفاده قرارمیگیرد.

```
pid_t getpid(void);
```

returns the process ID of the calling process.

این تابع برای دریافت شماره پروسه پدر فرآیند مورد استفاده قرار میگیرد.

```
pid_t getppid(void);
```

returns the process ID of the parent of the calling process.

```
unsigned int sleep(unsigned int seconds);
```

makes the calling prcess sleep until seconds seconds have elapsed or a signal arrives which is not ignored.

مثالها

fork

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
        pid_t pid;
        pid=fork();
        int inChild=0;
        if(pid==0)
        {
                inChild=1;
        }

        while(inChild==0)
        {
                printf("this is Parent\n");
                sleep(1);
        }

        while(inChild==1)
        {
                printf("this is Child\n");
                sleep(1);

        }

        return 0;
}
```

waitpid

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <time.h>

#define MAXCHILD 5

int main()
{
        pid_t child [MAXCHILD];
        int inChild=0;
        int status=0;
        int alive;
        srand(time(NULL));

        for (int i=0;i<MAXCHILD;i++)
        {
                child[i]=fork();
                if(child[i]==0)
                {
                        inChild=1;
                        break;
                }
        }
        while (inChild==1)
        {
                int r = rand()%10;
                printf("message from child %d \n",getpid());
                sleep(r);
                inChild=-1;
        }
        while(inChild==0)
        {
                sleep(1);
                for(int i=0;i<MAXCHILD;i++)
                {
                        alive=waitpid(child[i],&status,0);

                        if(alive==0)
                                printf("child[%d] is still alive\n",child[i]);
                        else
                                printf("child[%d] is dead now \n",child[i]);
                }
        }
        return 0;
}
```

gettimeofday

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <time.h>

int main(int argc,char ** argv)
{
        struct timeval start,stop;
        gettimeofday(&start,NULL);

        sleep(3);

        gettimeofday(&stop,NULL);

        long sec=stop.tv_sec-start.tv_sec;
        float m1=start.tv_usec;
        float m2=stop.tv_usec;

        long elapsed = sec*1000+(m2-m1)/1000;
        printf("%ld\n",elapsed);

        return 0;
}
```