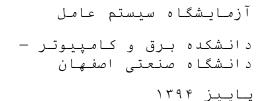




دستور كار جلسه هفتم

۲	فراخوانی های سیستمی برای ایجاد و مدیریت semaphore و thread
٣	فراخوانی های سیستمی مدیریت thread
٣	Header
٣	pthread_create
٣	thread execution routine
٣	pthread_join
۴	فراخوانی های سیستمی مدیریت semaphore
۴	compile using gcc
۵	مثال ها
۵	Threads: Creating , Executing and Joining
9	Accessing variable "total", avoiding multiple writes using semaphores
v	An Example of Busy-waiting
۸	دسته رکار حلسه هفتم





فراخوانی های سیستمی برای ایجاد و مدیریت semaphore و

در این دستورکار توضیحاتی درباره thread ارائه شده، همچنین درباره همگام سازی (Synchronization) اجرای در این دستورکار توضیحاتی درباره semaphore نیز جزئیاتی ذکر شده است.

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

http://linux.die.net/man

thread یا نخ: نخها قطعه کدهایی هستند که بطور همزمان و موازی با هم اجرا میشوند..مثل گرفتن اطلاعات از طریق شبکه و ذخیره آنها روی دیسک ...

در برنامه نویسی نخها ، همواره یک روتین تعریف میشود که در حقیقت عملی است که نخها باید بصورت موازی انجام دهند.



```
آزمایشگاه سیستم عامل
دانشکده برق و کامپیوتر -
دانشگاه صنعتی اصفهان
پاییز ۱۳۹۴
```

فراخوانی های سیستمی مدیریت thread

Header

```
#include <pthread.h>
```

pthread_create

thread execution routine

اگر بخواهیم صبر کنیم که نخ مشخص شده کارش تمام شود از تابع زیر استفاده میکنیم.

pthread join

```
pthread_join(pthread_t thread, void **return_value);
```





دانشکده برق و کامپیوتر – دانشگاه صنعتی اصفهان

یاپیز ۱۳۹۴

آشنایی با مفهوم سمافور:

سمافور نشان دهنده تعداد واحد منابعی است که در حال حاضر در دسترس هستند و پروسه ها میتوانند از آن استفاده کنند..در واقع به منظور مدیریت دسترسی پروسه ها به منابع مشترک از سمافور استفاده میشود..عملکرد سمافور به این صورت است که: هر فرآیندی که بخواهد به منبع دسترسی داشته باشد مقدار سمافور را بررسی میکند اگر مثبت بود فرآیند میتواند از منبع مشترک استفاده کند در اینصورت فرآیند یک واحد از سمافور میکاهد تا نشان دهد از این منبع مشترک استفاده کرده است..این عمل را با استفاده از تابع ()sem_wait انجام میدهد..اگر سمافور صفر یا کوچکتر از صفر بود فرآیند بخواب میرود تا زمانی که سمافور مقداری مثبت بگیرد در این حالت فرآیند از خواب بیدار میشود و از مرحله قبل شروع میکند..وقتی فرآیند کارش با منبع تمام شد یک واحد به سمافور اضافه میکند که این عمل را با استفاده از تابع ()sem_post انجام میدهد..وقتی مقدارش صفر یا بزرگتر از صفر شد یکی از فرآیندهایی که بخواب رفته به صورت تصادفی یا به روش FIFO توسط سیستم عامل بیدار میشود....

فراخوانی های سیستمی مدیریت semaphore

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
int sem_getvalue(sem_t *sem, int *valp);
int sem_destroy(sem_t *sem);
```

compile using gcc

```
gcc code.c -o appName -std=c99 -lpthread
compiled with library "pthread"
```

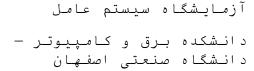


```
آزمایشگاه سیستم عامل
دانشکده برق و کامپیوتر –
دانشگاه صنعتی اصفهان
پاییز ۱۳۹۴
```

مثال ها

Threads: Creating, Executing and Joining

```
- this program creates 4 threads
- execution routine for threads is "routine1", we pass thread index (i) as
 execution routine argument
- each thread executes the routine in an arbitrary order, in this condition we have
no control on order of execution
- at pthread_join(), master thread waits for worker threads to complete their
 execution, then receives their "exit value"
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#define THREADS 4
void *routine1(void * x)
   printf("threadIdx = %d\n",x);
   pthread_exit((void *)x);
int main ()
        pthread_t threads[THREADS];
        for ( int i=0;i<THREADS;i++)</pre>
                 pthread_create(&threads[i], NULL, routinel, (void *)i);
        int retval;
        for (int i=0; i<THREADS; i++)
                pthread_join(threads[i],&retval);
                printf("threadIdx %d finished, return_value = %d \n",i,retval);
        return 0;
```



یاییز ۱۳۹۴



Accessing variable "total", avoiding multiple writes using semaphores

```
- program creates 4 threads, assigns "routine1" as execution routine for each
thread
- defines semaphore "sem1" in global space to be accessible by all threads
- each thread before entering its critical section, evaluates the value of "sem1"
- remark: sem_wait decrements semaphore /sem_post increments semaphore
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define THREADS 4
sem_t sem1;
int total=0;
void *routine1(void * id )
        int idx=(int)id;
         sem_wait(&sem1);
        //beginning of critical section
        total+=1;
        printf("thread=%d and total=%d \n",idx,total);
         sleep(1);
         //end of critical section
         sem_post(&sem1);
        pthread_exit((void *)idx);
int main ()
        sem_init(&sem1,0,1);
        pthread_t threads[THREADS];
         for ( int i=0;i<THREADS;i++)</pre>
                 pthread_create(&threads[i],NULL,routine1,(void *)i);
         for (int i=0; i<THREADS; i++)</pre>
                 pthread_join(threads[i],NULL);
         return 0;
```





دانشکده برق و کامپیوتر – دانشگاه صنعتی اصفهان

پاییز ۱۳۹۴

An Example of Busy-waiting

```
- this program creates 4 threads and assigns "routine1" as execution routine for
each thread
- threads will be synchronized by checking the value of variable "total"
- this method is called "busy-waiting"
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define THREADS 4
#define SIZE 16
int data[SIZE];
sem_t sem1;
int step=0;
int total=0;
void *func1(void * id )
    int idx=(int)id;
    //busy wait on step value
    while (step < 16)
        while ( step%THREADS != idx );
        //beginning of critical section
        total+=1;
        printf("thread=%d and total=%d \n",idx,total);
        step++;
        sleep(1);
        //end of critical section
    pthread_exit((void *)idx);
int main ()
    sem_init(&sem1,0,1);
    pthread_t threads[THREADS];
    for ( int i=0;i<THREADS;i++)</pre>
                 pthread_create(&threads[i], NULL, func1, (void *)i);
    for (int i=0; i<THREADS; i++)
        pthread_join(threads[i],NULL);
   return 0;}
```