



دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

# آزمایشگاه سیستم عامل

جلسه هشتم

برنامه‌نویسی thread

علی فانیان

زینب زالی

تابستان ۱۳۹۸



## POSIX threads

با مفهوم **thread** و مزایای برنامه‌نویسی **multithread** در درس سیستم عامل آشنا شدید. POSIX، کتابخانه **pthread** را برای برنامه‌نویسی چندینگی ارائه کرده است. در این کتابخانه علاوه بر ساخت **thread**، امکانات زیادی شامل تنظیم زمان‌بندی‌های مختلف از طریق **pthread\_attr** و همزمانی برنامه چندینگی (**synchronization**) وجود دارد.

## فراخوانی های سیستمی مدیریت thread

### Header

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

### pthread\_create

```
int pthread_create ( pthread_t *thread,  
                    const pthread_attr_t *attr,  
                    void *(*start_routine) (void *),  
                    void *arg  
                    );
```

با فراخوانی این تابع، روتینی که در آرگومان سوم مشخص شده‌است در یک نخ جدید، شروع به اجرا می‌کند (نخ جدید را نخ فرزند و نخی که **pthread\_create** در آن فراخوانی شده نخ والد می‌گوییم). به این روتین اصطلاحاً **runner** می‌گویند و الگوی آن در قسمت بعد، مشخص شده است. آرگومان اول، هندلر نخ است که پس از ساخته شدن نخ، در صورت موفقیت مقدار مخالف **NULL** خواهد داشت. آرگومان دوم مشخصاتی را برای نخ جدید، تعریف می‌کند. در صورتی که بخواهیم الگوریتم زمان‌بندی یا الویت زمان‌بندی یا پارامترهای دیگر مربوط به نخ را تغییر دهیم، موارد مربوطه را باید در رکوردی از نوع **pthread\_attr\_t** تنظیم کرده (با استفاده از توابع مربوطه) و سپس در آرگومان دوم استفاده کنیم. اما اگر بخواهیم نخ با تنظیمات پیش‌فرض سیستم ساخته شود، مقدار آرگومان دوم را می‌توان **NULL** داد و یا از تابع **pthread\_attr\_init** برای مقداردهی رکورد آن استفاده کرد. آرگومان چهارم این تابع هم پارامترهای ورودی **runner** را مشخص می‌کند که باید حتماً همه پارامترها در قالب **void\*** به تابع ارسال شود.

### thread execution routine (runner)

```
void * (void * arg)  
{  
    //execution routine  
    pthread_exit(void * return_value);  
}
```

نمونه‌ای از یک روتین **runner** را می‌بینید که آرگومان ورودی آن **void\*** است. بدین ترتیب هر نوع آرگومان ورودی که برای این روتین نیاز باشد باید در قالب **void\*** به آن ارسال شود و سپس در بدنه **runner**، به نوع دلخواه **cast** شود. اگر تعداد آرگومان مورد نیاز بیش از یک باشد، موارد مربوطه به صورت یک رکورد یا **structure** تعریف شده، **cast** به **void\*** می‌شود و در آرگومان چهارم **pthread\_create** ارسال می‌شود سپس در بدنه روتین **runner**، دوباره به نوع رکورد موردنظر **cast** می‌شود. تابع **pthread\_exit** هم جهت خاتمه دادن به اجرای نخ در بدنه **runner** استفاده می‌شود و قابلیت ارسال یک مقدار خروجی را دارد. این مقدار خروجی توسط نخ‌ی که **pthread\_join** را برای این نخ فراخوانی کرده دریافت می‌شود.

### pthread\_join

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

با استفاده از این فراخوانی، می‌توان منتظر اتمام نخ‌ی با هندلر **thread** شد. همچنین مقدار خروجی نخ موردنظر در آرگومان سوم دریافت می‌شود (این مقدار همان‌طور که در بخش قبل بیان شد، باید توسط **pthread\_exit** در تابع **runner** ارسال شود).

### compile using gcc

```
gcc code.c -o appName -lpthread
```

جهت کامپایل، برنامه‌ای که از توابع هدر **pthread** استفاده کرده است لازم است کتابخانه **pthread** در کامپایل اضافه شود.



## مثال

### 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" that is a random number generated in the thread's routine
*/

#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <unistd.h>

#define THREADS 4
#define TIMEOUT 10

void *routine1(void * x)
{
    int *t = (int*)malloc(sizeof(int));
    *t = rand()%TIMEOUT;
    //int t = rand()%TIMEOUT; //replace two above lines with this line
    printf("threadIdx = %d, execution time = %d\n",*(int*)x, *t);
    for(int i=0; i<*t; i++)
        printf("threadIdx = %d; run = %d\n", *(int*)x, i);
    pthread_exit((void*)t);
}

int main ()
{
    pthread_t threads[THREADS];
    int thread_id[THREADS];
    for ( int i=0;i<THREADS;i++){
        thread_id[i] = i;
        pthread_create(&threads[i], NULL, routine1, (void *)&thread_id[i] );
        //replace two above lines with the below line
        //pthread_create(&threads[i], NULL, routine1, (void *)&i );
    }
    int *retval = (int*)malloc(sizeof(int));
    for (int i=0; i<THREADS; i++)
    {
        pthread_join(threads[i],(void**)&retval);
        printf("threadIdx %d finished, return_value = %d \n",i,*retval);
    }
    return 0;
}
```

به نحوه ارسال آرجومان‌های روتین thread و همچنین برگرداندن خروجی نخ دقت کنید. توجه داشته باشید که اگر آدرس‌ها در آرجومان‌های ورودی یا خروجی نخ‌های متفاوت مشترک باشند، ممکن است در اثر همزمانی اجراها یا خارج شدن از scope، مقادیر مربوطه صحیح ارسال نشوند. برای بررسی این موضوع کد را با توجه به کامنت‌های بین کدها تغییر داده و دوباره اجرا کنید.