تمرین دوم || عماد آقاجانی || ۸۸۵۲۱۳۴۴

استاد تو آزمایشی که من انجام دادم و کدش در زیر همین فایل قرار گرفته،نشان میدهد که نیازی به وجود حلقه برای pthread_cond_wait نمی باشد . (محل تست : Ubuntu 11)

همچنین میتونم به توضیح تابع pthread_cond_signal/broadcast() هم که در این لینک قرار گرفته، استناد کنم که:

If more than one thread is blocked on a condition variable, the scheduling policy determines the order in which threads are unblocked. When each thread unblocked as a result of a pthread_cond_signal() or pthread_cond_broadcast() returns from its call to pthread_cond_wait() or pthread_cond_timedwait(), the thread owns the mutex with which it called pthread_cond_wait() or pthread_cond_timedwait().

قسمت قرمز رنگ اشاره به متوقف شدن برنامه بعد از فراخوانی تابع pthread_cond_wait دارد .

نمونه کدی که نشان میدهد نیازی به وجود حلقه نسیت: (کدش به پیوست، با نام "*wait_loop_test_EMAD.c*" قرار گرفته)

در این کد هدف چاپ اعداد ۱ تا ۵ و ۱۱ تا ۱۵ توسط functionCount1 و اعداد مابین توسط functionCount2 میباشد که از pthread_cond_t بعنوان **شرط انجام** و **ارتباط دو thread** به شیوه signal/slot استفاده شده است.

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

pthread_mutex_t count_mutex     = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  condition_var   = PTHREAD_COND_INITIALIZER;

void *functionCount1();
void *functionCount2();
int  count = 1;

main()
{
   pthread_t thread1, thread2;

   pthread_create( &thread1, NULL, &functionCount1, NULL);
   pthread_create( &thread2, NULL, &functionCount2, NULL);

   pthread_join( thread1, NULL);
   pthread_join( thread2, NULL);

   exit(0);
}

void *functionCount1() // print 1-5 then 11-15
{
```

```
	pthread_mutex_lock( &count_mutex);
	while ( count <= 5 )
		printf("# %d \n",count++);
	//count is 6
	pthread_cond_signal( &condition_var ); // func2 start printing

	pthread_cond_wait( &condition_var, &count_mutex ); // wait for func2
end signal

	while ( count <= 15 )
		printf("# %d \n",count++);

	pthread_mutex_unlock( &count_mutex );

	return NULL;
}
void *functionCount2() // print 6-10
{
	pthread_mutex_lock( &count_mutex);
	if ( count < 6 )
		pthread_cond_wait( &condition_var, &count_mutex ); // unlock and
wait

	while ( count <= 10 )
		printf("## %d \n",count++);

	pthread_cond_signal( &condition_var ); // func1 start second part

	pthread_mutex_unlock( &count_mutex );

	return NULL;
}
```

Compile: `cc -lpthread cond1.c`
Run: `./a.out`
Results:

```
	# 1
	# 2
	# 3
	# 4
	# 5
	## 6
	## 7
	## 8
	## 9
	## 10
	# 11
	# 12
	# 13
	# 14
	# 15
```

(Expected Result B-))