**تمرین دوم || عماد آقاجانی || 88521344**

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](http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cond_signal.html) 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()](http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cond_wait.html) or [pthread\_cond\_timedwait()](http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cond_timedwait.html), the thread owns the mutex with which it called [pthread\_cond\_wait()](http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cond_wait.html) or [pthread\_cond\_timedwait()](http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cond_timedwait.html).

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 1 تا 5 و 11 تا 15 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-) )