<https://blog.csdn.net/hudashi/article/details/7709421>

原文：

<http://apps.hi.baidu.com/share/detail/19786281>

<http://hi.baidu.com/boobleoo0/blog/item/5f935039a37c58f8b311c77f.html>

<http://topic.csdn.net/u/20110105/16/12717238-9816-4571-a03d-e8b603724946.html>

   pthread\_cond\_wait() 用于阻塞当前线程，等待别的线程使用pthread\_cond\_signal()或pthread\_cond\_broadcast来唤醒它。 pthread\_cond\_wait() 必须与pthread\_mutex 配套使用。pthread\_cond\_wait()函数一进入wait状态就会自动release mutex。当其他线程通过pthread\_cond\_signal()或pthread\_cond\_broadcast，把该线程唤醒，使pthread\_cond\_wait()通过（返回）时，该线程又自动获得该mutex。

  pthread\_cond\_signal函数的作用是发送一个信号给另外一个正在处于阻塞等待状态的线程,使其脱离阻塞状态,继续执行.如果没有线程处在阻塞等待状态,pthread\_cond\_signal也会成功返回。

  使用pthread\_cond\_signal一般不会有“惊群现象”产生，他最多只给一个线程发信号。假如有多个线程正在阻塞等待着这个条件变量的话，那么是根据各等待线程优先级的高低确定哪个线程接收到信号开始继续执行。如果各线程优先级相同，则根据等待时间的长短来确定哪个线程获得信号。但无论如何一个pthread\_cond\_signal调用最多发信一次。

  但是pthread\_cond\_signal在多处理器上可能同时唤醒多个线程，当你只能让一个线程处理某个任务时，其它被唤醒的线程就需要继续 wait，而且规范要求pthread\_cond\_signal至少唤醒一个pthread\_cond\_wait上的线程，其实有些实现为了简单在单处理器上也会唤醒多个线程.

   另外，某些应用，如线程池，pthread\_cond\_broadcast唤醒全部线程，但我们通常只需要一部分线程去做执行任务，所以其它的线程需要继续wait.所以强烈推荐对pthread\_cond\_wait() 使用while循环来做条件判断.

以下就是一个来自MAN的示例

  Consider two shared variables x and y, protected by the mutex mut, and a condition vari-

       able cond that is to be signaled whenever x becomes greater than y.

              int x,y;

              pthread\_mutex\_t mut = PTHREAD\_MUTEX\_INITIALIZER;

              pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;

       Waiting until x is greater than y is performed as follows:

              pthread\_mutex\_lock(&mut);

              while (x <= y) {

                      pthread\_cond\_wait(&cond, &mut);

              }

              /\* operate on x and y \*/

              pthread\_mutex\_unlock(&mut);

       Modifications on x and y that may cause x to become greater than y should signal the con-

       dition if needed:

              pthread\_mutex\_lock(&mut);

              /\* modify x and y \*/

              if (x > y) pthread\_cond\_broadcast(&cond);

              pthread\_mutex\_unlock(&mut);

pthread\_cond\_signal函数与条件变量的典型应用就是用来实现producer/consumer模型。

**示例1**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#define BUFFER\_SIZE 8

struct Products

{

int buffer[BUFFER\_SIZE];

/\*保证存取操作的原子性 互斥性\*/

pthread\_mutex\_t locker;

/\*是否可读\*/

pthread\_cond\_t notEmpty;

/\*是否可写\*/

pthread\_cond\_t notFull;

int posReadFrom;

int posWriteTo;

};

int BufferIsFull(struct Products\* products)

{

if ((products->posWriteTo + 1) % BUFFER\_SIZE == products->posReadFrom)

{

return (1);

}

return (0);

}

int BufferIsEmpty(struct Products\* products)

{

if (products->posWriteTo == products->posReadFrom)

{

return (1);

}

return (0);

}

/\*制造产品\*/。

void Produce(struct Products\* products, int item)

{

/\*原子操作\*/

pthread\_mutex\_lock(&products->locker);

/\*无空间可写入\*/

while (BufferIsFull(products))

{

pthread\_cond\_wait(&products->notFull, &products->locker);

}

/\*写入数据\*/

products->buffer[products->posWriteTo] = item;

products->posWriteTo++;

if (products->posWriteTo >= BUFFER\_SIZE)

products->posWriteTo = 0;

/\*发信\*/

pthread\_cond\_signal(&products->notEmpty);

/\*解锁\*/

pthread\_mutex\_unlock(&products->locker);

}

int Consume(struct Products\* products)

{

int item;

pthread\_mutex\_lock(&products->locker);

/\*为空时持续等待,无数据可读\*/

while (BufferIsEmpty(products))

{

pthread\_cond\_wait(&products->notEmpty, &products->locker);

}

/\*提取数据\*/

item = products->buffer[products->posReadFrom];

products->posReadFrom++;

/\*如果到末尾,从头读取\*/

if (products->posReadFrom >= BUFFER\_SIZE)

products->posReadFrom = 0;

pthread\_cond\_signal(&products->notFull);

pthread\_mutex\_unlock(&products->locker);

return item;

}

#define END\_FLAG (-1)

struct Products products;

void\* ProducerThread(void\* data)

{

int i;

for (i = 0; i < 16; ++i)

{

printf("producer: %d\n", i);

Produce(&products, i);

}

Produce(&products, END\_FLAG);

return NULL;

}

void\* ConsumerThread(void\* data)

{

int item;

while (1)

{

item = Consume(&products);

if (END\_FLAG == item)

       break;

printf("consumer: %d\n", item);

}

return (NULL);

}

int main(int argc, char\* argv[])

{

pthread\_t producer;

pthread\_t consumer;

int result;

pthread\_create(&producer, NULL, &ProducerThread, NULL);

pthread\_create(&consumer, NULL, &ConsumerThread, NULL);

pthread\_join(producer, (void \*)&result);

pthread\_join(consumer, (void \*)&result);

exit(EXIT\_SUCCESS);

}

示例2

pthread\_cond\_broadcast的是使用

#include <pthread.h> #include <stdlib.h> #include <stdio.h> #include <unistd.h> #include <string.h> pthread\_mutex\_t mymutex1 = PTHREAD\_MUTEX\_INITIALIZER; pthread\_mutex\_t mymutex2 = PTHREAD\_MUTEX\_INITIALIZER; pthread\_cond\_t mycond = PTHREAD\_COND\_INITIALIZER; void \*mythread1(void \*param) {  printf("begin mythread1.\n");  pthread\_mutex\_lock(&mymutex1);  printf("wait in mythread1.\n");  pthread\_cond\_wait(&mycond,&mymutex1);  pthread\_mutex\_unlock(&mymutex1);  printf("end mythread1.\n");  return NULL; } void \*mythread2(void \*param) {  printf("begin mythread2.\n");  pthread\_mutex\_lock(&mymutex2);  printf("wait in mythread2.\n");  pthread\_cond\_wait(&mycond,&mymutex2);  pthread\_mutex\_unlock(&mymutex2);  printf("end mythread2.\n");  return NULL; } int main(void) {  printf("begin main thread.\n");  int i; pthread\_t tid1,tid2;  pthread\_create(&tid1,NULL,mythread1,NULL);  pthread\_create(&tid2,NULL,mythread2,NULL);  sleep(5);  printf("try to wake up mythread1 and mythread2 in main thread.\n");  if(pthread\_cond\_broadcast(&mycond)){ printf("error\n");  return 1;  }  void \*res;  pthread\_join(tid1, &res);  pthread\_join(tid2, &res);  printf("end main thread.\n");  return 0; }

运行结果：

**实例2**

pthread\_cond\_broadcast的使用

**[cpp]** [view plain](https://blog.csdn.net/hudashi/article/details/7709421) [copy](https://blog.csdn.net/hudashi/article/details/7709421)

1. #include <pthread.h>
2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <unistd.h>
5. #include <string.h>
6. pthread\_mutex\_t mymutex1 = PTHREAD\_MUTEX\_INITIALIZER;
7. pthread\_mutex\_t mymutex2 = PTHREAD\_MUTEX\_INITIALIZER;
8. pthread\_cond\_t mycond = PTHREAD\_COND\_INITIALIZER;
9. **void** \*mythread1(**void** \*param)
10. {
11. printf("begin mythread1.\n");
12. pthread\_mutex\_lock(&mymutex1);
13. printf("wait in mythread1.\n");
14. pthread\_cond\_wait(&mycond,&mymutex1);
15. pthread\_mutex\_unlock(&mymutex1);
16. printf("end mythread1.\n");
17. **return** NULL;
18. }
19. **void** \*mythread2(**void** \*param)
20. {
21. printf("begin mythread2.\n");
22. pthread\_mutex\_lock(&mymutex2);
23. printf("wait in mythread2.\n");
24. pthread\_cond\_wait(&mycond,&mymutex2);
25. pthread\_mutex\_unlock(&mymutex2);
26. printf("end mythread2.\n");
27. **return** NULL;
28. }
29. **int** main(**void**)
30. {
31. printf("begin main thread.\n");
32. **int** i;
33. pthread\_t tid1,tid2;
34. pthread\_create(&tid1,NULL,mythread1,NULL);
35. pthread\_create(&tid2,NULL,mythread2,NULL);
36. sleep(2);
37. printf("try to wake up mythread1 and mythread2 in main thread.\n");
38. **if**(pthread\_cond\_broadcast(&mycond)){
39. printf("error\n");
40. **return** 1;
41. }
42. **void** \*res;
43. pthread\_join(tid1, &res);
44. pthread\_join(tid2, &res);
45. printf("end main thread.\n");
46. **return** 0;
47. }

运行结果：

begin main thread.  
begin mythread1.  
wait in mythread1.  
begin mythread2.  
wait in mythread2.  
end mythread2.  
try to wake up mythread1 and mythread2 in main thread.  
end mythread1.  
end main thread.

注意mythread2并真正的等待，它和我们的期望有所差别，似乎一个pthread\_cond\_t只能对应一个pthread\_mutex\_t

我们把以上代码稍作修正就可以了，具体见下面的**实例3**

**实例3**

**[cpp]** [view plain](https://blog.csdn.net/hudashi/article/details/7709421) [copy](https://blog.csdn.net/hudashi/article/details/7709421)

1. #include <pthread.h>
2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <unistd.h>
5. #include <string.h>
6. pthread\_mutex\_t mymutex1 = PTHREAD\_MUTEX\_INITIALIZER;
7. //pthread\_mutex\_t mymutex2 = PTHREAD\_MUTEX\_INITIALIZER;
8. pthread\_cond\_t mycond = PTHREAD\_COND\_INITIALIZER;
9. **void** \*mythread1(**void** \*param)
10. {
11. printf("begin mythread1.\n");
12. pthread\_mutex\_lock(&mymutex1);
13. printf("wait in mythread1.\n");
14. pthread\_cond\_wait(&mycond,&mymutex1);
15. pthread\_mutex\_unlock(&mymutex1);
16. printf("end mythread1.\n");
17. **return** NULL;
18. }
19. **void** \*mythread2(**void** \*param)
20. {
21. printf("begin mythread2.\n");
22. pthread\_mutex\_lock(&mymutex1);
23. printf("wait in mythread2.\n");
24. pthread\_cond\_wait(&mycond,&mymutex1);
25. pthread\_mutex\_unlock(&mymutex1);
26. printf("end mythread2.\n");
27. **return** NULL;
28. }
29. **int** main(**void**)
30. {
31. printf("begin main thread.\n");
32. **int** i;
33. pthread\_t tid1,tid2;
34. pthread\_create(&tid1,NULL,mythread1,NULL);
35. pthread\_create(&tid2,NULL,mythread2,NULL);
36. sleep(2);
37. printf("try to wake up mythread1 and mythread2 in main thread.\n");
38. **if**(pthread\_cond\_broadcast(&mycond)){
39. printf("error\n");
40. **return** 1;
41. }
42. **void** \*res;
43. pthread\_join(tid1, &res);
44. pthread\_join(tid2, &res);
45. printf("end main thread.\n");
46. **return** 0;
47. }

运行结果

123@xyy ~/gcc-test  
$ gcc threadTest.c -o test.exe  
123@xyy ~/gcc-test  
$ ./test.exe  
begin main thread.  
begin mythread1.  
wait in mythread1.  
begin mythread2.  
wait in mythread2.  
try to wake up mythread1 and mythread2 in main thread.  
end mythread1.  
end mythread1.  
end main thread.

该结果才是我们真正需要的。

结束！