1 条件变量

条件变量给多个线程提供了一个汇合的场所。

依赖的头文件

#include<pthread.h>

函数声明

定义分配条件变量

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

int pthread\_cond\_init(pthread\_cond\_t \*restrict cond, const pthread\_condattr\_t \*restrict attr);

|  |  |
| --- | --- |
| 名称： | pthread\_cond\_init |
| 功能： | initialize condition variables 初始化条件变量 |
| 头文件： | #include <pthread.h> |
| 函数原形： | int pthread\_cond\_init(pthread\_cond\_t \*restrict cond,  const pthread\_condattr\_t \*restrict attr); |
| 参数： |  |
| 返回值： | the pthread\_cond\_destroy() and pthread\_cond\_init() functions shall return zero; otherwise, an error number shall be returned to indicate the error |

int pthread\_cond\_destroy(pthread\_cond\_t \*cond);

|  |  |
| --- | --- |
| 名称： | pthread\_cond\_destroy |
| 功能： | Destroy condition variables |
| 头文件： | #include <pthread.h> |
| 函数原形： | int pthread\_cond\_destroy(pthread\_cond\_t \*cond); |
| 参数： |  |
| 函数说明： | 在释放或废弃条件变量之前，需要毁坏它，使用此函数 |
| 返回值： | the pthread\_cond\_destroy() and pthread\_cond\_init() functions shall return zero; otherwise, an error number shall be returned to indicate the error |

int pthread\_cond\_wait(pthread\_cond\_t \*restrict cond,pthread\_mutex\_t \*restrict mutex);

|  |  |
| --- | --- |
| 名称： | pthread\_cond\_wait |
| 功能： | Wait on a condition，等待某个条件是否成立。对于timewait()函数除了等待以外，可以设置一个时长。 |
| 头文件： | #include <pthread.h> |
| 函数原形： | int pthread\_cond\_wait(pthread\_cond\_t \*restrict cond,pthread\_mutex\_t \*restrict mutex); |
| 参数： |  |
| 函数说明： | 一旦初始化了互斥对象和条件变量，就可以等待这个条件，一个特定条件只能有一个互斥对象，而且条件变量应该表示互斥数据“内部”的一种特殊的条件更改。一个互斥条件可以用许多条件变量（例如：cond\_empty,cond\_full,cond\_cleanup）,但每个条件变量只能有一个互斥对象。 |
| 返回值： |  |

int pthread\_cond\_signal(pthread\_cond\_t \*cond);

|  |  |
| --- | --- |
| 名称： | pthread\_cond\_signal |
| 功能： | signal a condition，这种情况是只有一个线程收到后执行动作。 |
| 头文件： | #include <pthread.h> |
| 函数原形： | int pthread\_cond\_signal(pthread\_cond\_t \*cond); |
| 参数： |  |
| 函数说明： | 活动线程只需要唤醒第一个正在睡眠的线程。假设您只对队列添加了一个工作作业。那么只需要唤醒一个工作程序线程（再唤醒其它线程是不礼貌的！） |
| 返回值： |  |

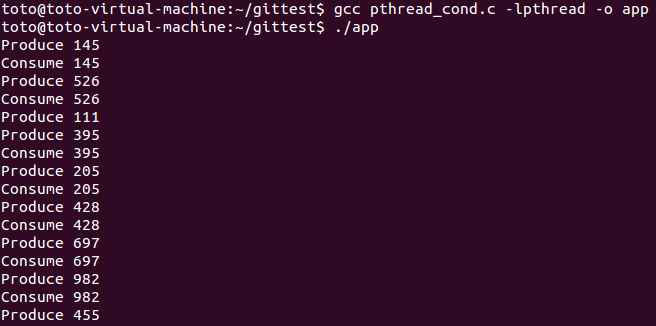
int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

|  |  |
| --- | --- |
| 名称： | pthread\_cond\_broadcast |
| 功能： | broadcast a condition，通过广播的形式发给子线程消息，子线程竞争执行。 |
| 头文件： | #include <pthread.h> |
| 函数原形： | int pthread\_cond\_broadcast(pthread\_cond\_t \*cond); |
| 参数： |  |
| 函数说明： | 如果线程更改某些共享数据，而且它想要唤醒所有正在等待的线程，则应使用 pthread\_cond\_broadcast 调用 |
| 返回值： |  |

案例说明：

|  |
| --- |
| #include <stdlib.h>  #include <pthread.h>  #include <stdio.h>  #include <unistd.h>  struct msg {  struct msg \*next;  int num;  };  struct msg \*head;  /\* 条件变量 \*/  pthread\_cond\_t has\_product = PTHREAD\_COND\_INITIALIZER;  pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER;  void \*consumer(void \*p)  {  struct msg \*mp;  for (;;) {  pthread\_mutex\_lock(&lock);  /\* pthread\_cond\_wait(&has\_product, &lock);  \* 1.阻塞等待has\_product被唤醒，  \* 2.释放互斥锁， pthread\_mutex\_unlock(&lock)  \* 3.当被唤醒时，解除阻塞，并且重新去申请获得互斥锁 pthread\_mutex\_lock(&lock)  \*/  while (head == NULL)  pthread\_cond\_wait(&has\_product, &lock);  mp = head;  head = mp->next;  pthread\_mutex\_unlock(&lock);  printf("Consume %d\n", mp->num);  free(mp);  sleep(rand() % 5);  }  }  void \*producer(void \*p)  {  struct msg \*mp;  for (;;) {  mp =(struct msg \*)malloc(sizeof(struct msg));  mp->num = rand() % 1000 + 1;  printf("Produce %d\n", mp->num);  pthread\_mutex\_lock(&lock);  mp->next = head;  head = mp;  pthread\_mutex\_unlock(&lock);  /\* pthread\_cond\_broadcast(&has\_product) 唤醒等待队列上的所有线程\*/  //发送信号，告诉消费者有产品了  pthread\_cond\_signal(&has\_product);  sleep(rand() % 5);  }  }  int main(int argc, char \*argv[])  {  pthread\_t pid, cid;  srand(time(NULL));  pthread\_create(&pid, NULL, producer, NULL);  pthread\_create(&cid, NULL, consumer, NULL);  pthread\_join(pid, NULL);  pthread\_join(cid, NULL);  return 0;  } |

运行结果：



总结：从上面可以看出，消费者总是消费最先生产出来的一个。