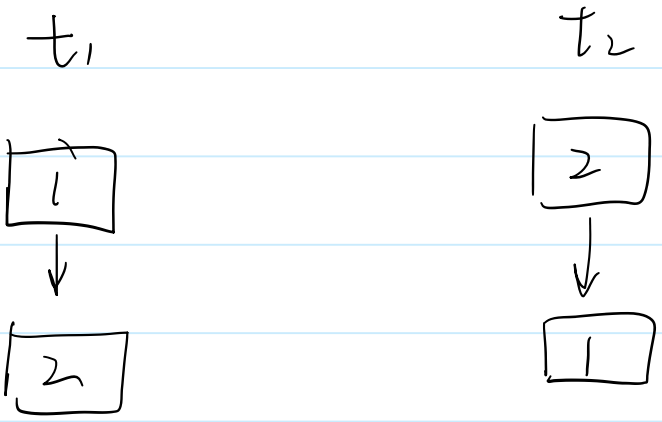
2023年5月3日 9:27

1_sync.c

```c
#include <49func.h>
typedef struct shareRes_s {
    int flag; //描述当前应该执行A还是B 0-->A 1-->B
    pthread_mutex_t mutex;//保护对flag的访问
} shareRes_t;
void *threadFunc(void *arg){
    shareRes_t * pShareRes = (shareRes_t *)arg;
    //判断B能否执行
    while(1){
        pthread_mutex_lock(&pShareRes->mutex);
        if(pShareRes->flag == 1){
            pthread_mutex_unlock(&pShareRes->mutex);
            break;
        }
        pthread_mutex_unlock(&pShareRes->mutex);
    }
    printf("B begin!\n");
    sleep(3);
    printf("B end!\n");
}
```

互斥锁 死锁

2023年5月3日 11:06



加锁的顺序

第二种死锁

2023年5月3日 11:14

持有锁的线程在常锁的状态下终止了

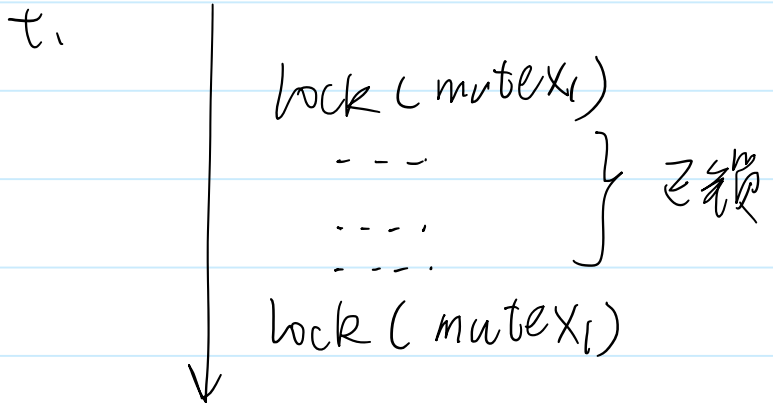
```
pthread_mutex_t mutex;
void *threadFunc(void *arg){
    pthread_mutex_lock(&mutex);
    printf("I am child!\n");
    pthread_exit(NULL);
}
int main()
{
    pthread_t tid;
    pthread_mutex_init(&mutex,NULL);
    pthread_create(&tid,NULL,threadFunc,NULL);
    sleep(1);
    pthread_mutex_lock(&mutex);
    printf("I am parent!\n");
    pthread_mutex_unlock(&mutex);
    pthread_join(tid,NULL);
    return 0;
}
```

解决方案：在线程终止之前记得解锁

第三种死锁

2023年5月3日 11:28

线程1，做2次lock操作，在加锁的情况下，对同一把锁再lock。



重复加锁时，不得失解锁

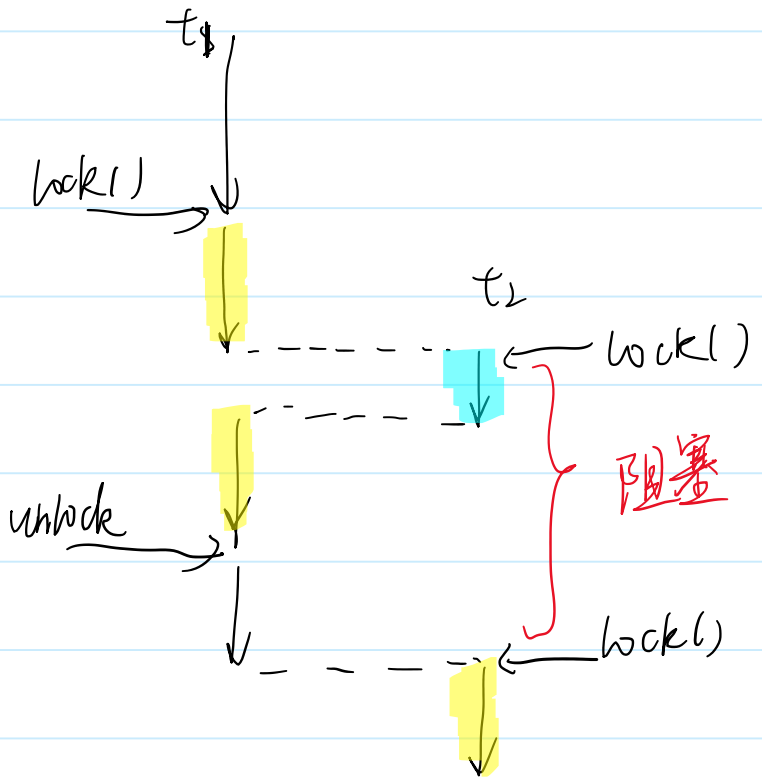
```c
int main()
{
    pthread_mutex_t mutex;
    pthread_mutex_init(&mutex,NULL);
    pthread_mutex_lock(&mutex);
    printf("lock once!\n");
    pthread_mutex_lock(&mutex);
    printf("lock twice!\n");
    pthread_mutex_unlock(&mutex);
    pthread_mutex_unlock(&mutex);
    return 0;
}
```

```
while(1){
    lock();
    ...
    ...
}
```

trylock

2023年5月3日 11:36

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```



非阻塞式加锁

① 若未锁，则加锁继续运行

② 若已锁，则立刻返回报错。

```
while(1){
    trylock
}
```

2023年5月3日 11:53

```c
void *threadFunc(void *arg){
    sleep(1);
    while(1){
        int ret = pthread_mutex_trylock(&mutex);
        THREAD_ERROR_CHECK(ret,"trylock 2");
        if(ret == 0){
            break;
        }
    }
    sleep(2);
    printf("child thread, sleep over!\n");
    pthread_mutex_unlock(&mutex);
    pthread_exit(NULL);
}
int main()
{
    pthread_t tid;
    pthread_mutex_init(&mutex,NULL);
    pthread_create(&tid,NULL,threadFunc,NULL);
    int ret = pthread_mutex_trylock(&mutex);
    THREAD_ERROR_CHECK(ret,"trylock 1");
    sleep(3);
    printf("main thread, sleep over!\n");
    pthread_mutex_unlock(&mutex);
    pthread_join(tid,NULL);
    return 0;
}
```

自旋锁

2023年5月3日 11:51

不满足条件 做死循环

mutex_lock → 阻塞锁

```
[liao@ubuntu Linuxday_17]$ man -k pthread_spin_
pthread_spin_destroy (3) - initialize or destroy a spin lock
pthread_spin_destroy (3posix) - destroy or initialize a spin lock object
pthread_spin_init (3) - initialize or destroy a spin lock
pthread_spin_lock (3) - lock and unlock a spin lock
pthread_spin_lock (3posix) - lock a spin lock object
pthread_spin_trylock (3) - lock and unlock a spin lock
pthread_spin_unlock (3) - lock and unlock a spin lock
pthread_spin_unlock (3posix) - unlock a spin lock object
```

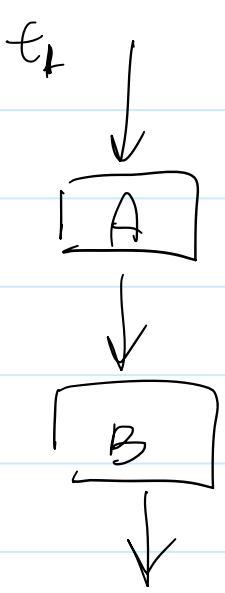
读写锁

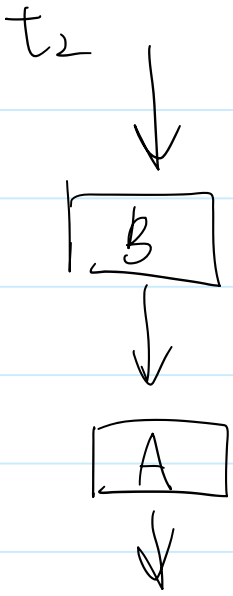
2023年5月3日 11:54

```
[liao@ubuntu Linuxday_17]$ man -k pthread_rwlock
pthread_rwlock_destroy (3posix) - destroy and initialize a read-write lock object
pthread_rwlock_rdlock (3posix) - lock a read-write lock object for reading
pthread_rwlock_timedrdlock (3posix) - lock a read-write lock for reading
pthread_rwlock_timedwrlock (3posix) - lock a read-write lock for writing
pthread_rwlock_tryrdlock (3posix) - lock a read-write lock object for reading
pthread_rwlock_trywrlock (3posix) - lock a read-write lock object for writing
pthread_rwlock_unlock (3posix) - unlock a read-write lock object
pthread_rwlock_wrlock (3posix) - lock a read-write lock object for writing
pthread_rwlockattr_destroy (3posix) - destroy and initialize the read-write lock attributes object
pthread_rwlockattr_getkind_np (3) - set/get the read-write lock kind of the thread read-write lock attribute object
pthread_rwlockattr_getpshared (3posix) - get and set the process-shared attribute of the read-write lock attributes object
pthread_rwlockattr_init (3posix) - initialize the read-write lock attributes object
pthread_rwlockattr_setkind_np (3) - set/get the read-write lock kind of the thread read-write lock attribute object
pthread_rwlockattr_setpshared (3posix) - set the process-shared attribute of the read-write lock attributes object
```

使用trylock解决第一种死锁

2023年5月3日 11:57





t1

trylock(A)(循环)

trylock(B) 成功

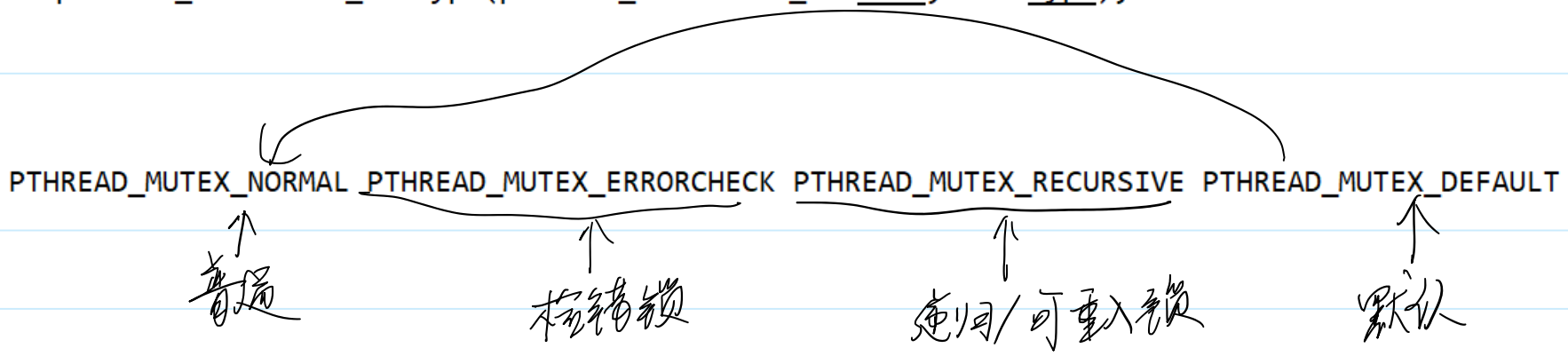
处理A

活锁 引入随机时间 sleep

锁的类型

2023年5月3日 14:35

```
int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
    int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```



设置锁的属性

2023年5月3日 14:39

① pthread_mutexattr_t

② pthread_mutexattr_init

③ pthread_mutexattr_settype

⇒ pthread_mutexattr_t

↓ &

④ pthread_mutex_init

检错锁

2023年5月3日 14:47

```c
int main()
{
    pthread_mutexattr_t mutexattr;
    pthread_mutexattr_init(&mutexattr);//初始化锁的属性
    pthread_mutexattr_settype(&mutexattr,PTHREAD_MUTEX_ERRORCHECK);//设置锁的属性
    pthread_mutex_t mutex;
    pthread_mutex_init(&mutex,&mutexattr);//用锁的属性初始化锁

    // 主线程先后对同一个锁加锁两次
    int ret = pthread_mutex_lock(&mutex);
    THREAD_ERROR_CHECK(ret,"mutex_lock 1");
    printf("lock once!\n");
    ret = pthread_mutex_lock(&mutex);
    THREAD_ERROR_CHECK(ret,"mutex_lock 2");
    printf("lock twice!\n");
    return 0;
}
```

```
[liao@ubuntu Linuxday_17]$ ./5_errorcheck
lock once!
mutex_lock 2:Resource deadlock avoided
lock twice!
```

可重入锁

2023年5月3日 14:52

- 线程对同一把锁可以多次重复加锁
  每一次加锁，引用计数+1

```
pthread_mutexattr_t mutexattr;
pthread_mutexattr_init(&mutexattr);//初始化锁的属性
pthread_mutexattr_settype(&mutexattr,PTHREAD_MUTEX_RECURSIVE);//设置锁的属性
pthread_mutex_t mutex;
pthread_mutex_init(&mutex,&mutexattr);//用锁的属性初始化锁

//
pthread_t tid;
pthread_create(&tid,NULL,threadFunc,&mutex);

// 主线程先后对同一个锁加锁两次
int ret = pthread_mutex_lock(&mutex);
THREAD_ERROR_CHECK(ret,"mutex_lock 1");
printf("lock once!\n");
ret = pthread_mutex_lock(&mutex);
THREAD_ERROR_CHECK(ret,"mutex_lock 2");
printf("lock twice!\n");
sleep(1);
printf("sleep over 1\n");
pthread_mutex_unlock(&mutex);
sleep(1);
printf("sleep over 2\n");
pthread_mutex_unlock(&mutex);

pthread_join(tid,NULL);
return 0;
```

```
void *threadFunc(void *arg){
    pthread_mutex_t * pmutex = (pthread_mutex_t *)arg;
    sleep(1);
    pthread_mutex_lock(pmutex);
    printf("child thread!\n");
    pthread_mutex_unlock(pmutex);
}
```

两个卖票窗口并发卖票

2023年5月3日 15:00

对于共享资源的访问 没有放临界区

```
void *sellWindow1(void *arg){
    shareRes_t * pShareRes = (shareRes_t *)arg;
    while(pShareRes->ticket > 0){
        pthread_mutex_lock(&pShareRes->mutex);
        printf("Before window1 sells ticket, ticket = %d\n", pShareRes->ticket);
        --pShareRes->ticket;
        printf("After window1 sells ticket, ticket = %d\n", pShareRes->ticket);
        pthread_mutex_unlock(&pShareRes->mutex);
        //sleep(1);
    }
    pthread_exit(NULL);
}
```

把对共享资源的访问放入临界区进行保护

2023年5月3日 15:17

```
void *sellWindow1(void *arg){
    shareRes_t * pShareRes = (shareRes_t *)arg;
    while(1){
        pthread_mutex_lock(&pShareRes->mutex);
        if(pShareRes->ticket <= 0){
            pthread_mutex_unlock(&pShareRes->mutex);
            break;
        }
        printf("Before window1 sells ticket, ticket = %d\n", pShareRes->ticket);
        --pShareRes->ticket;
        printf("After window1 sells ticket, ticket = %d\n", pShareRes->ticket);
        pthread_mutex_unlock(&pShareRes->mutex);
        //sleep(1);
    }
    pthread_exit(NULL);
}
```

引入一个加票窗口

2023年5月3日 15:23

先卖票到小于等于10张，再加票

同步.

条件.

① flag. 决定 加票能否运行

③ 在加票线程中，额外检查 flag.

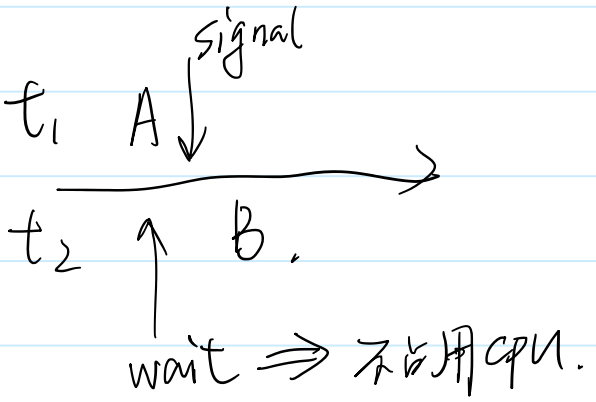
```
}
void * addTicket(void *arg){
    shareRes_t * pShareRes = (shareRes_t *)arg;
    while(1){
        pthread_mutex_lock(&pShareRes->mutex);
        if(pShareRes->flag != 0){
            // 加票
            pShareRes->ticket += 10;
            printf("ticket is added!\n");
            pthread_mutex_unlock(&pShareRes->mutex);
            break;
        }
        pthread_mutex_unlock(&pShareRes->mutex);
    }
    pthread_exit(NULL);
}
```

条件变量 → 资源：等待 wait
通知 signal

2023年5月3日 16:04

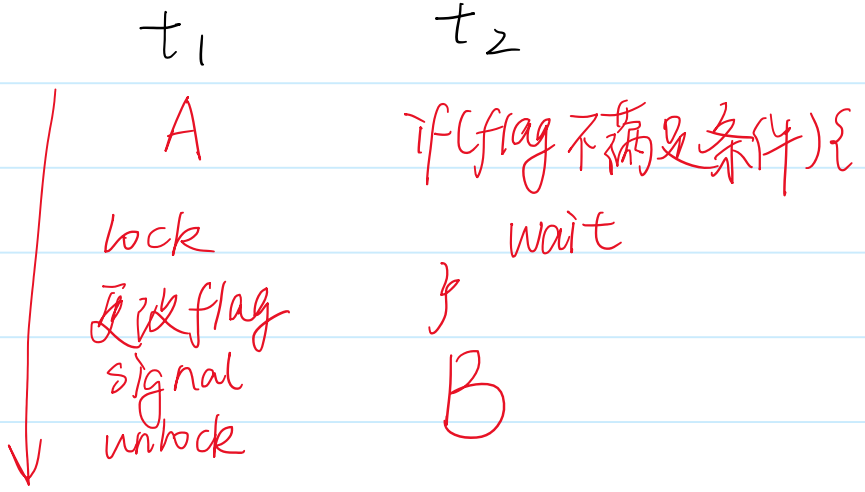
无竞争地线程间同步

满足条件不要死循环



① 设计一个flag条件，决定B能否执行

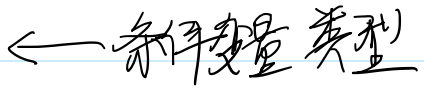
② flag是共享资源，flag的访问必须加锁 mutex



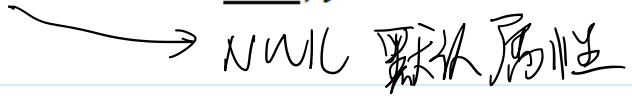
条件变量相关的代码

2023年5月3日 16:20

```
pthread_cond_t
```




```
int pthread_cond_init(pthread_cond_t *restrict cond,
    const pthread_condattr_t *restrict attr);
```




```
int pthread_cond_signal(pthread_cond_t *cond);

int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

pthread_cond_wait的原理

2023年5月3日 16:25

阻塞等待 直到signal

pthread_mutex_lock.
if(flag不满足){
pthread_cond_wait
}
.....// 已锁状态

上半部 等待之前
① 检查是否加锁.
② 把本线程加入到阻塞队列
③ 解锁并阻塞等待 ←—— 原子操作

下半部 醒来以后
④ 醒来之后 先加锁
⑤ 以加锁状态 执行后续代码.

先掌握基本流程

2023年5月3日 17:13

```c
typedef struct shareRes_s {
    int flag; // 0-->B不能运行 1-->B可以运行
    pthread_mutex_t mutex; // 保护flag的锁
    pthread_cond_t cond; // 条件变量 提供一种无竞争的同步机制
} shareRes_t;
```

```c
printf("A begin!\n");
sleep(3);
printf("A end!\n");
pthread_mutex_lock(&shareRes.mutex);
shareRes.flag = 1;
pthread_cond_signal(&shareRes.cond);
pthread_mutex_unlock(&shareRes.mutex);
```

先事件

调整flag，cond_signal

```c
pthread_mutex_lock(&pShareRes->mutex);
if(pShareRes->flag != 1){//B不能执行的话
    pthread_cond_wait(&pShareRes->cond,&pShareRes->mutex);
}
pthread_mutex_unlock(&pShareRes->mutex);
printf("B begin!\n");
sleep(3);
printf("B end!\n");
```

先检查是否要等待.

后事件.

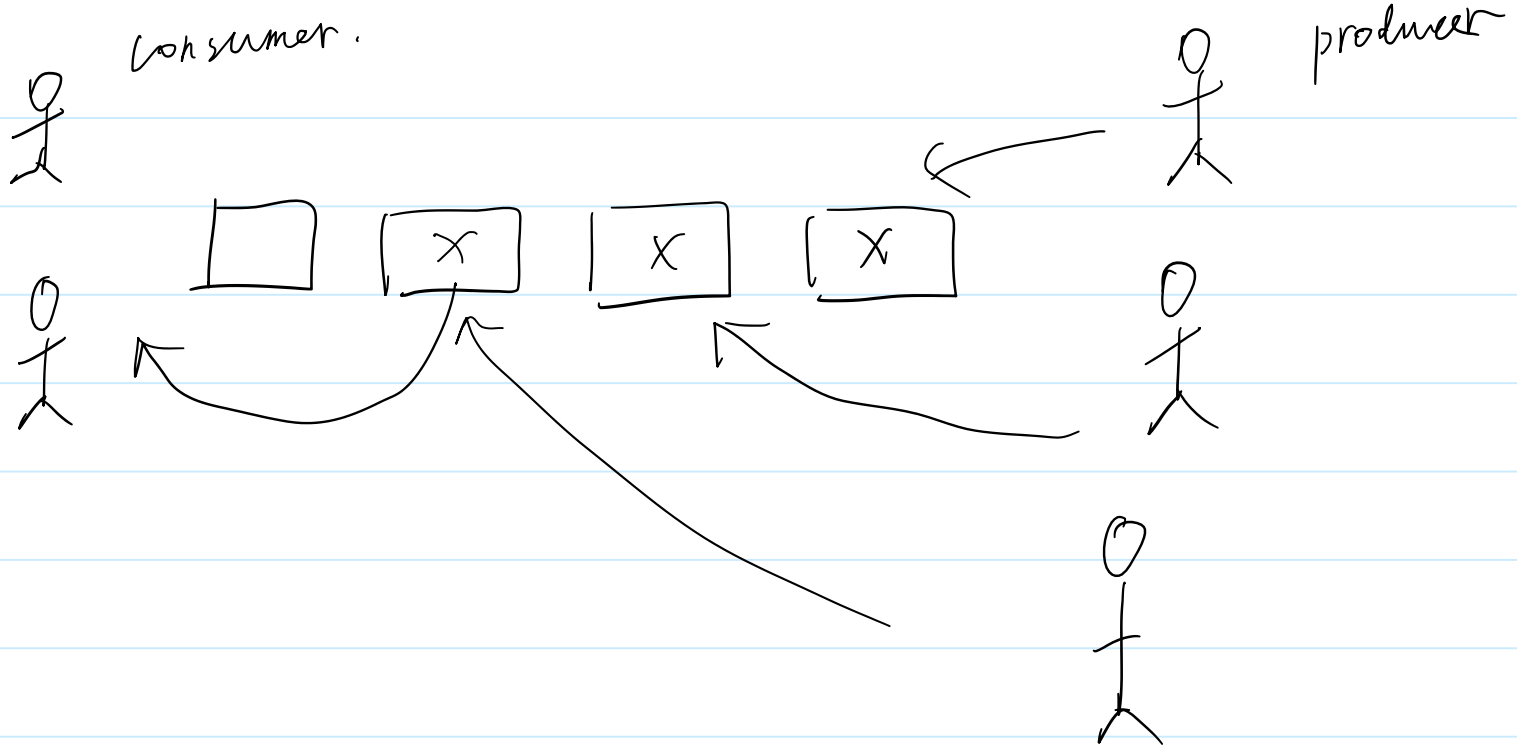
使用条件变量改造加票

2023年5月3日 17:18

```
void * addTicket(void *arg){
    shareRes_t * pShareRes = (shareRes_t *)arg;
    pthread_mutex_lock(&pShareRes->mutex);
    if(pShareRes->flag == 0){
        printf("ticket is enough now!\n");
        pthread_cond_wait(&pShareRes->cond,&pShareRes->mutex);
    }
    printf("ticket is not enough now!\n");
    pShareRes->ticket += 10;
    pthread_mutex_unlock(&pShareRes->mutex);
    pthread_exit(NULL);
}
int main()
```

生产者消费者模型

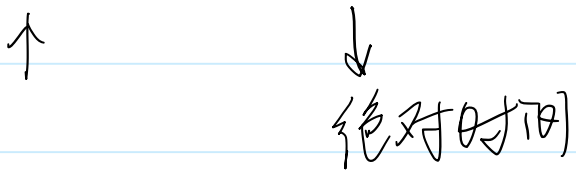
2023年5月3日 17:29



带有超时时限的cond_wait

2023年5月3日 17:37

```
int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
```




```
struct timespec {
    time_t tv_sec;        /* seconds */
    long   tv_nsec;       /* nanoseconds */
};
```

```c
#include <49func.h>
int main()
{
    pthread_mutex_t mutex;
    pthread_cond_t cond;
    pthread_mutex_init(&mutex,NULL);
    pthread_cond_init(&cond,NULL);
    pthread_mutex_lock(&mutex);
    time_t now = time(NULL);
    printf("before time = %s\n", ctime(&now));
    struct timespec abstime;
    abstime.tv_sec = now+10;
    abstime.tv_nsec = 0;
    pthread_cond_timedwait(&cond,&mutex,&abstime);
    pthread_mutex_unlock(&mutex);
    now = time(NULL);
    printf("after time = %s\n", ctime(&now));
    return 0;
}
```

pthread_cond_broadcast 广播

2023年5月3日 17:47

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

```
typedef struct shareRes_s {
    int num; //描述饭的数量
    pthread_mutex_t mutex;
    pthread_cond_t cond;
} shareRes_t;
void *threadFunc(void *arg){
    shareRes_t * pShareRes = (shareRes_t *)arg;
    //顾客
    pthread_mutex_lock(&pShareRes->mutex);
    //if(pShareRes->num == 0){
    while(pShareRes->num == 0){//把if换成while避免虚假唤醒
        pthread_cond_wait(&pShareRes->cond,&pShareRes->mutex);
    }
    printf("Before I got food! num = %d\n", pShareRes->num);
    --pShareRes->num;
    printf("After I got food! num = %d\n", pShareRes->num);
    pthread_mutex_unlock(&pShareRes->mutex);
    pthread_exit(NULL);
}
```

```
//厨师
sleep(3);
pthread_mutex_lock(&shareRes.mutex);
++shareRes.num;
//pthread_cond_signal(&shareRes.cond);
pthread_cond_broadcast(&shareRes.cond);
pthread_mutex_unlock(&shareRes.mutex);
```