RWLOCK

```
typedef struct RWlock {
    pthread_mutex_t lock;
    pthread_cond_t cond;
    int AR, AW, WR, WW;
}RWLock;
```

```
void RWLock_init(RWLock* l) {
    pthread_mutex_init(&l->lock, NULL);
    pthread_cond_init(&l->cond, NULL);
    l->AR = l->AW = l->WR = l->WW = 0;
}
```

```
void RWLock_destory(RWLock* l){
    pthread_mutex_destroy(&l->lock);
}
```

RWLOCK

```
void RWLock_startwrite(RWLock* l) {
    pthread_mutex_lock(&l->lock);
    l->WW ++;
    while(l->AW > 0) {
        pthread_cond_wait(&l->cond, &l->lock);
    }
    l->WW--;
    l->AW++;
    pthread_mutex_unlock(&l->lock);
}

void RWLock_donewrite(RWLock* l) {
    pthread_mutex_lock(&l->lock);
    l->AW--;
    if(l->WW > 0) {
        pthread_cond_signal(&l->cond);
    } else {
        pthread_cond_broadcast(&l->cond);
    }
    pthread_mutex_unlock(&l->lock);
}
```

```
void RWLock_startread(RWLock* l) {
    pthread_mutex_lock(&l->lock);
    l->WR ++;
    while(l->WW > 0 || l->AW > 0) {
        pthread_cond_wait(&l->cond, &l->lock);
    }
    l->WR--;
    l->AR++;
    pthread_mutex_unlock(&l->lock);
}

void RWLock_doneread(RWLock* l) {
    pthread_mutex_lock(&l->lock);
    l->AR--;
    if(l->AR==0 || l->WW > 0)
        pthread_cond_signal(&l->cond);
    pthread_mutex_unlock(&l->lock);
}
```

RWLOCK Test

```
void* rfn(void* arg) {
    int id = *(int*) arg;
    RWLock_startread(&rwlock);
    usleep(100000);
    read_count ++;
    RWLock_doneread(&rwlock);
    return NULL;
}

void* wfn(void* arg) {
    int id = *(int*) arg;
    RWLock_startwrite(&rwlock);
    write_count ++;
    shared_resource++;
    usleep(100000);
    RWLock_donewrite(&rwlock);
    return NULL;
}
```

```
RWLock rwlock;
int shared_resource = 0;
atomic_int read_count = 0;
atomic_int write_count = 0;

int main() {
    RWLock_init(&rwlock);
    pthread_t reader[5], writer[3];
    int reader_ids[5] = {0, 1, 2, 3, 4};
    int writer_ids[3] = {0, 1, 2};

    for (int i = 0; i < 5; i ++){
        pthread_create(&reader[i], NULL, rfn, &reader_ids[i]);
    }

    for (int i = 0; i < 3; i ++){
        pthread_create(&writer[i], NULL, wfn, &writer_ids[i]);
    }

    for(int i = 0;i<5;i++){
        pthread_join(reader[i], NULL);
    }

    for(int i = 0;i<3;i++){
        pthread_join(writer[i], NULL);
    }

    RWLock_destory(&rwlock);

    // 输出最终结果
    printf("Final shared resource value: %d\n", shared_resource);
    printf("Total read operations: %d\n", read_count);
    printf("Total write operations: %d\n", write_count);

    return 0;
}
```

CAS RWLOCK

```c
#define WRITE_LOCK 0x80000000  // 写锁标志位（高位）
#define READ_MASK  0x7FFFFFFF  // 读锁计数掩码（低位）
// 初始化 RWLock
void RWLock_init(RWLock *rwlock) {
    atomic_store(&rwlock->state, 0);
}
```

```c
// 获取读锁
void RWLock_startRead(RWLock *rwlock) {
    while (1) {
        int current = atomic_load(&rwlock->state);
        if (current & WRITE_LOCK) {  // 如果写锁被占用，则自旋
            sched_yield();  // 让出 CPU
            continue;
        }
        // 尝试增加读锁计数
        if (atomic_compare_exchange_weak(&rwlock->state, &current,
current + 1)) {
            break;
        }
    }
}

// 释放读锁
void RWLock_doneRead(RWLock *rwlock) {
    atomic_fetch_sub(&rwlock->state, 1);  // 减少读锁计数
}
```

```c
// 获取写锁
void RWLock_startWrite(RWLock *rwlock) {
    while (1) {
        int current = atomic_load(&rwlock->state);
        if (current != 0) {  // 如果有读锁或写锁被占用，则自旋
            sched_yield();
            continue;
        }
        // 尝试设置写锁（高位）
        if (atomic_compare_exchange_weak(&rwlock->state, &current,
WRITE_LOCK)) {
            break;
        }
    }
}

// 释放写锁
void RWLock_doneWrite(RWLock *rwlock) {
    atomic_store(&rwlock->state, 0);  // 释放写锁
}
```

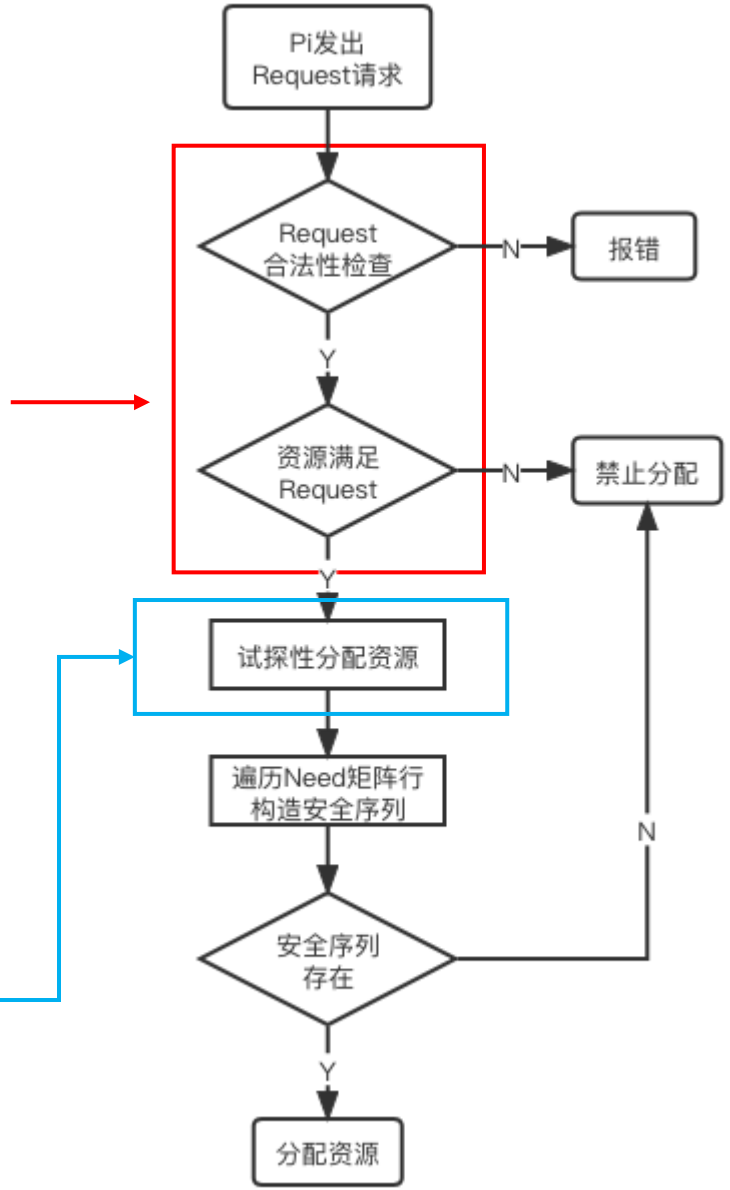
- CAS: Compare And Swap
- Test参考一般的RWLOCK，使用方式基本是一样的
- Note: 这个实现相比一般的RWLOCK少了一些特性，比如要等读者都完成才能写，这会导致饥饿

Banker

```python
def valid_check(P, Request, Need):
    if np.sum(Request > Need[P, :]):
        print("\033[0;31mERROR!P{} REQUEST EXCEEDS ITS NEED!\033[0m".format(P))
        return False
    print("\033[0;32mVALID CHECK PASSED!\033[0m")
    return True


def resource_check(Request, Available):
    if np.sum(Request > Available):
        print("\033[0;31mREQUEST EXCEEDS AVAILABLE!\033[0m")
        return False
    else:
        print("\033[0;32mRESOURCE CHECK PASSED!\033[0m")
        return True
```

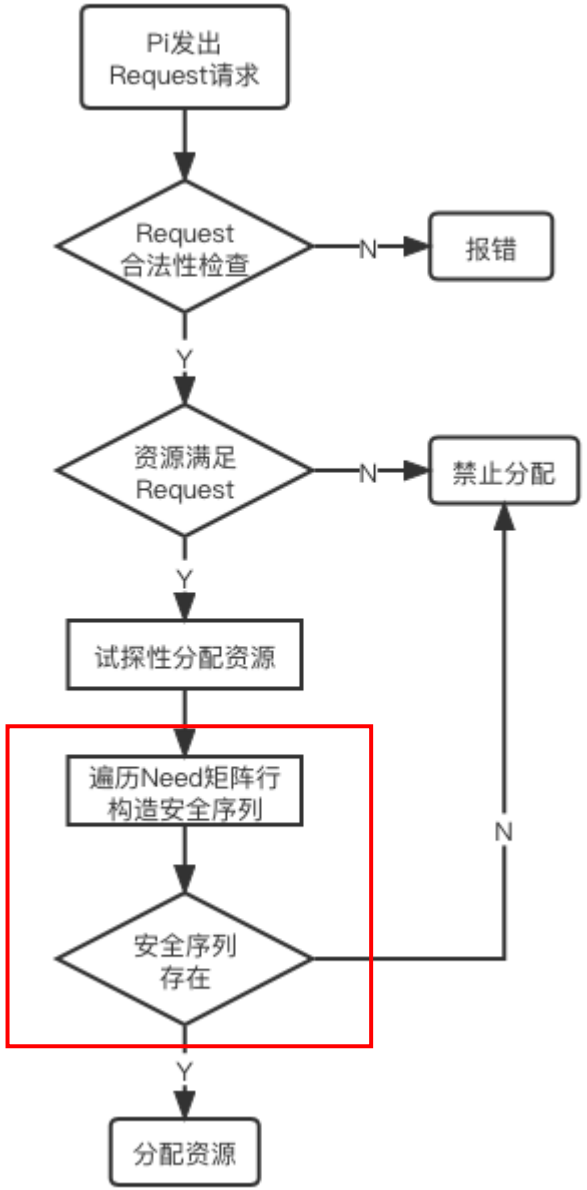
```python
def try_allocate(Available, Allocation, Need, Request, P):
    Available = Available - Request
    Allocation[P, :] = Allocation[P, :] + Request
    Need[P, :] = Need[P, :] - Request
    return Available, Need, Allocation
```



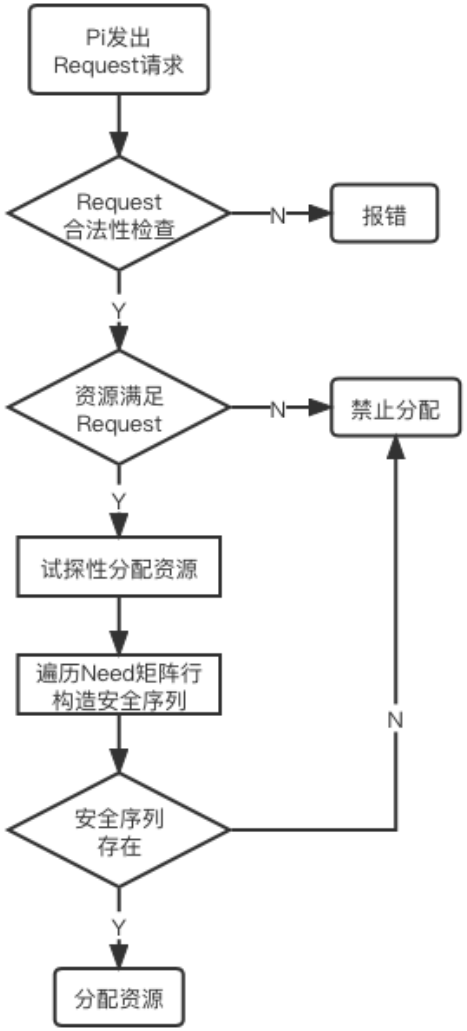
参考 [银行家算法之Python实现[操作系统实验]_银行家算法python-CSDN博客]

Banker

```python
def safe_check(Work, Need, Allocation, n):
    finished = np.sum(np.all(Need == 0, axis=1) & np.all(Allocation == 0, axis=1))
    print("Number of processes with both Need and Allocation as 0: {}".format(finished))
    Q = []
    while True:
        i = 0
        while i < n:
            if i not in Q and not np.sum(Need[i, :] > Work) and not (np.all(Need[i, :] == 0) and np.all(Allocation[i, :] == 0)):
                Q.append(i)
                temp = Work.copy()
                Work = Work + Allocation[i, :]
                print_safe_check(i, temp, Need, Allocation, Work)
                break
            i = i + 1
        if i == n:
            break
    if len(Q) < n - finished:
        print("\033[0;31mSAFE CHECK FAILED!\033[0m")
        return False
    else:
        print("\033[0;32mSAFE CHECK PASSED!\nSecured Sequence is {}.\033[0m".format(Q))
        return True
```



Banker



Max：任务要求的最大设备资源
Available：目前可用的设备资源
Allocation：目前分配的资源
Request：任务请求的资源
Need：任务需要的设备资源

作业中N：任务数量
M：资源类型
R：资源数量
T：任务要求的最大资源数量

T <-> Max
R <-> Available
R : [1, M]
T : [N, M]

```python
def Banker(n, Available, Max, Allocation, P, Request):
    """
    :param n: int
    :param Available: array[n][m]
    :param Max: array[n][m]
    :param Allocation: array[n][m]
    :param P: index of process to request
    :param Request: array[m]
    :return: Available, Need, Allocation
    """
    global finished
    print_request_info(P, Request)
    Available = np.asarray(Available)
    Max = np.asarray(Max)
    Allocation = np.asarray(Allocation)
    Request = np.asarray(Request)
    Need = Max - Allocation
    print_resource(Available, Need, Allocation, n)
    if (valid_check(P, Request, Need) and resource_check(Request, Available) and
            safe_check(*try_allocate(Available.copy(),
                                     Allocation.copy(),
                                     Need.copy(),
                                     Request, P), n)):
        Available, Need, Allocation = try_allocate(Available.copy(),
                                                   Allocation.copy(),
                                                   Need.copy(),
                                                   Request, P)
        # 如果有进程的Need全是0，将该进程的Allocation加到Available里面，并将Allocation置0
        for i in range(n):
            if np.all(Need[i, :] == 0):
                Available = Available + Allocation[i, :]
                Allocation[i, :] = 0
                Max[i, :] = 0
                finished.add(i)

        print("Finished process: {}".format(finished))
        print_resource(Available, Need, Allocation, n)
    return Available, Max, Allocation
```

Banker Test

```python
def all_random():
    strs = []

    N = random.randint(4, 8)
    M = random.randint(3, 6)
    R = [random.randint(1, 10) for _ in range(M)]
    T = [[random.randint(0, R[i]) for i in range(M)] for _ in range(N)]
    seq_len = random.randint(10, 20)
    S = []
    for _ in range(seq_len):
        for i in range(M):
            ps = random.randint(0, N-1)
            request = [random.randint(0, min(R[i], T[ps][i])) for i in range(M)]
            S.append([ps, request])

    strs.append(str(N))
    strs.append(str(M))
    strs.append(' '.join(map(str, R)))
    for i in range(N):
        strs.append(' '.join(map(str, T[i])))

    for s in S:
        strs.append(str(s[0]) + ' ' + ' '.join(map(str, s[1])))

    with open("random.in", "w") as f:
        f.write('\n'.join(strs))
        f.write('\n\n')
```

```python
def pass_random():
    strs = []

    N = random.randint(4, 8)
    M = random.randint(3, 6)
    R = [random.randint(1, 20) for _ in range(M)]
    T = [[random.randint(0, R[i] // N) for i in range(M)] for _ in range(N)]
    seq_len = random.randint(20, 50)
    S = []
    for _ in range(seq_len):
        for i in range(M):
            ps = random.randint(0, N-1)
            request = [random.randint(0, min(R[i], T[ps][i])) for i in range(M)]
            S.append([ps, request])

    strs.append(str(N))
    strs.append(str(M))
    strs.append(' '.join(map(str, R)))
    for i in range(N):
        strs.append(' '.join(map(str, T[i])))

    for s in S:
        strs.append(str(s[0]) + ' ' + ' '.join(map(str, s[1])))

    with open("passrandom.in", "w") as f:
        f.write('\n'.join(strs))
        f.write('\n\n')
```