Report-Project6

付昊源
517021910753

2019 年 6 月 12 日

1 Banker's Algorithm

1.1 程序设计思想

本 project 用于模拟银行家算法，从而进行系统资源的分配。银行家算法中，需要一个数组 available 用于记录系统中各种资源的空闲量；三个二维数组 maximum，allocation 和 need，分别用于记录每一个用户（进程）对于各种资源的最大需求量，已分配的量，以及还所需的量。根据定义，

$$need[i,j] = maximum[i,j] - allocation[i,j]$$

我们需要为用户留有的接口为，为某个进程分配资源、释放某个进程的部分资源以及显示当前状态。其中，为进程分配资源需要用到银行家算法来进行判断死锁的出现。银行家算法需要判断，假设按照用户的意愿分配了相应资源后，当前的 available 是否可以满足某一个进程的 need，如果可以，则分配给该进程 need 的资源，回收过后继续判断其他进程，直到可以把所有进程都执行完或出现死锁状态。如果出现死锁状态，则拒绝用户的资源申请。

1.2 运行结果截图

首先，从命令行输入 available，从文件 maximum.txt 读入 maximum 矩阵，默认各进程的 allocation 均为 0，显示状态（图 1）。

接下来，为 0 号进程分配资源，可知该分配成功，并显示分配后的状态（图 2）。

```
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project6$ ./banker 10 5 7 8
Initialize finish!
user> *
available:
R1 R2 R3 R4
10  5  7  8

maximum:
R1 R2 R3 R4
6  4  7  3
4  2  3  2
2  5  3  3
6  3  3  2
5  3  7  5

allocation:
R1 R2 R3 R4
0  0  0  0
0  0  0  0
0  0  0  0
0  0  0  0
0  0  0  0

need:
R1 R2 R3 R4
6  4  7  3
4  2  3  2
2  5  3  3
6  3  3  2
5  3  7  5
```

图 1: Initialization State

```
user> RQ 0 3 3 3 3
0
user> *
available:
R1 R2 R3 R4
7  2  4  5

maximum:
R1 R2 R3 R4
6  4  7  3
4  2  3  2
2  5  3  3
6  3  3  2
5  3  7  5

allocation:
R1 R2 R3 R4
3  3  3  3
0  0  0  0
0  0  0  0
0  0  0  0
0  0  0  0

need:
R1 R2 R3 R4
3  1  4  0
4  2  3  2
2  5  3  3
6  3  3  2
5  3  7  5
```

图 2: Request For Customer0

下面为 1 号进程分配资源，并将其回收，以判断回收指令的正确性，分别显示状态（图 3、图 4）。

```
user> RQ 1 2 2 2 2
0
user> *
available:
R1 R2 R3 R4
5  0  2  3

maximum:
R1 R2 R3 R4
6  4  7  3
4  2  3  2
2  5  3  3
6  3  3  2
5  3  7  5

allocation:
R1 R2 R3 R4
3  3  3  3
2  2  2  2
0  0  0  0
0  0  0  0
0  0  0  0

need:
R1 R2 R3 R4
3  1  4  0
2  0  1  0
2  5  3  3
6  3  3  2
5  3  7  5
```

图 3: Request For Customer1

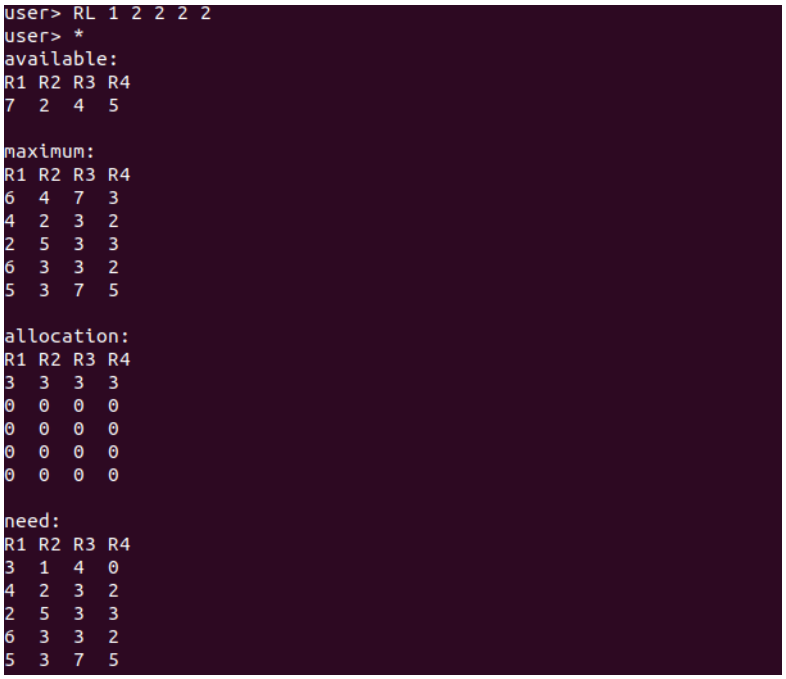

图 4: Release Customer1

最后，构造死锁状态，判断银行家算法的正确性。如果我们为 4 号进程分配 2, 2, 2, 2，则会出现死锁状态（R2 不足），分配结果如图 5。

图 5: Request With a Dead Lock

1.3 核心数据结构及代码解释

主函数中首先进行的是命令行数据的获取，文件的写入，以及四个全局数组的初始化。以上初始化完成后会出现”Initialize finish!” 提示符，下面进入用户操作阶段。每次读入用户的输入指令以判别用户的行为，对于 RQ 和 RL，需要获取相关的参数，并调用 request_resources() 和 release_resources() 函数，这两个函数的实现会在后文中详细描述。“*” 指令用于显示当前状态，则只需分别输出四个数组的内容即可。

接下来讲解 request_resources() 函数。首先我们需要对申请的数据进行基本的判别，如进程号是否在范围内，所要申请的资源量是否小于其 need 量等等。之后使用银行家算法进行死锁的判断，为了不影响实际的 available 和 need，这里采用复制一个副本的方式进行模拟运算，只修改副本中 available 和 need 的值，判断无死锁后再修改实际的数组。判断的过程我采用 check_safe_state() 函数进行检查，检查的过程我采用了递归函数的方式（也可以使用循环），在检查的过程中，同样要使用复制副本，修改副本，保留原数组的实现方式。需要注意的是，对于某一些 need 均为 0 的进程，需要跳过它们，因为已经无需检查这样的进程，否则可能会出现死循环的现象。最终 check_safe_state() 函数会返回 bool 变量 true 或 false，从而告知死锁的出现与否。

最后讲解 release_resources() 函数。在本函数中，我们要判断进程号是否在范围内，以及所要释放的资源是否小于等于其 allocation。如果以上两个条件均满足，则直接修改 allocation、available 和 need 数组，即可完成。