Operating System CH8

Susie Glitter

2025 年 7 月 7 日

注：本次实验使用了 VMware 中的 ubuntu-16.04.6-desktop

1 任务一：实现银行家算法

1.1 获取并打印四个数组

银行家算法涉及四个数组或矩阵，分别为 available 数组，表示剩余可分配资源，allocation 数组，表示各个任务已经被分配的资源，maximum 数组，表示各个任务完成所需的资源总量，need 数组，表示各个任务完成所需的剩余资源。其中 allocation 与 need 的和即为 maximum 数组

我们使用 c 语言的主函数参数获得 available 数组，使用文件重定向，从文件中获得 maximum 数组，need 数组初始与 maximum 数组一致，而 allocation 数组初始为 0

1.2 实现指令 RL 释放资源

释放资源较为简单，只需要判断目标任务是否拥有能满足释放要求的资源即可，即比较 allocation 与 release 数组即可。若申请释放资源成功，则 allocation 数组减去对应值，need 与 available 数组加上对应值

1.3 实现指令 RQ 请求资源

请求资源需要使用到银行家算法，判断申请是否安全。一共有两种请求不会被满足，一是超过现有的 available 数组的请求，二是申请后，若不释放资源，则会导致死锁的请求

对于第二种非法请求，我们递归地进行判断，每次递归寻找可以被处理的任务，模拟对其进行处理并且夺回其 allocation 的资源，递归至所有任务均被完成，则说明这个请求资源是合法的，否则说明这次申请会导致死锁，不予满足

1.4 程序实现

使用简单的循环程序完成 QL 请求。使用递归的程序完成 QR，每次递归进行模拟的申请，返回前恢复状态并且返回是否成功的状态码，即可实现银行家算法。

1.5 结果展示

```
gg@ubuntu:~/Desktop/final-src-osc10e/ch8/banker$ ./banker 5 6 7 8 <input.txt
>RQ 0 1 1 1 2
The state is not safe!

>RL 0 1 0 0 0
0 customer doesn't have this much resources!

>*
available array is
5 6 7 8
maximum matrix is
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
allocation matrix is
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
need matrix is
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5

>exitgg@ubuntu:~/Desktop/final-src-osc10e/ch8/banker$
```