2017/11/29

操作系统实验报告

实验三 预防进程死锁的银行家算法

软件工程一班_秦源_1525161007

一　需求分析

问题描述：

设计程序模拟预防进程死锁的银行家算法的工作过程。假设系统中有 n 个进程 P1, … ,Pn，有 m 类可分配的资源 R1, … ,Rm，在 T0 时刻，进程 Pi 分配到的 j 类资源为 Allocationij 个，它还需要 j 类资源 Need ij 个，系统目前剩余 j 类资源 Workj 个，现采用银行家算法进行进程资源分配预防死锁的发生。

程序要求：

1）判断当前状态是否安全，如果安全给出安全序列；如果不安全给出理由。

2）对于下一个时刻 T1，某个进程 Pk 会提出请求 Request(R1, … ,Rm)，判断分配给 P k 进程请求的资源之后系统是否安全。

3）输入：进程个数 n，资源种类 m，T0 时刻各个进程的资源分配情况（可以运行输入，也可以在程序中设置）；

4）输出：如果安全，输出安全的进程序列，不安全则提示信息。

二　概要设计

可利用资源向量 Available。这是一个含有 m 个元素的数组，其中的每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。

当所有进程都能获取足够资源并完成，则状态安全。

三　详细设计

main.sh文件声明了静态变量以及默认数据。

Func.sh文件定义函数以及主要的算法。(细节见原代码注释)

四　调试分析

设Requesti是进程Pi的请求向量，如果Requesti [j] =K，表示进程Pi需要K个Rj类型的资源。当Pi发出资源请求后，系统按下述步骤进行检查：

(1) 如果Requesti [j] ≤Need [i,j] ，便转向步骤2；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

(2) 如果Requesti [j] ≤Available [j] ，便转向步骤(3)；否则， 表示尚无足够资源，Pi须等待。

(3) 系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值：

Available [j] ：=Available [j] -Requesti [j] ;

Allocation [i,j] ：=Allocation [i,j] +Requesti [j] ;

Need [i,j] ：=Need [i,j] -Requesti [j] ;

(4) 系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程Pi，以完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程Pi等待。

五　用户使用说明

使用终端，将运行目录切换到源代码所在路径，运行main.sh。根据提示即可使用。

六　测试结果

```
bogon:SourceCode qinyuan$ ./main.sh
实验三　预防进程死锁的银行家算法
----------默认数据-------------
进程名　　MAX　　Allocation
  P0　　7 5 3　　0 1 0
  P1　　3 2 2　　2 0 0
  P2　　9 0 2　　3 0 2
  P3　　2 2 2　　2 1 1
  P4　　4 3 3　　0 0 2
可用资源：　3 3 2
--------------------------------
实验数据选择　1-使用默认数据，　2-输入新数据
1
算法开始：
当前状态安全
安全序列为：　P1　P3　P0　P2　P4
是否添加输入下一时刻进程资源请求Request?
1-添加资源请求　2-不添加资源请求
1
请输入进程ID:(可供选择的进程ID有012345)
1
请输入资源请求向量:(长度为3)
1 0 2
当前状态安全
安全序列为：　P1　P3　P0　P2　P4
--------------下一时刻---------------
是否添加输入下一时刻进程资源请求Request?
1-添加资源请求　2-不添加资源请求
2
无新请求,程序退出.
```

七　附录

----------main.sh----------

```
#! /bin/bash

. func.sh

#进程个数n
```

```
n=5
#资源种类m
m=3
#可用资源
declare -a Available
Available=(3 3 2)
#最大需求资源
declare -a Max
Max=(7 5 3 3 2 2 9 0 2 2 2 2 4 3 3)
#分配资源
declare -a Allocation
Allocation=(0 1 0 2 0 0 3 0 2 2 1 1 0 0 2)
echo "实验三  预防进程死锁的银行家算法"
echo "----------默认数据-------------"
echo -e "进程名\t MAX\tAllocation"
for i in `seq 0 $[$n-1]`; do
     echo -n -e  "  P$i\t"
     for j in `seq 0 $[$m-1]`; do
          echo -n -e "${Max[$[$i*$m+$j]]} "
     done
     echo -n -e "\t  "
     for j in `seq 0 $[$m-1]`; do
          echo -n -e "${Allocation[$[$i*$m+$j]]} "
     done
```

```
    echo ""
done
echo "可用资源:  ${Available[@]}"
echo "--------------------------------"
echo "实验数据选择 1-使用默认数据, 2-输入新数据"
read keypress
case "$keypress" in
    1 )
        echo "算法开始:"
        ;;
    2 )
        echo "请输入进程个数n:"
        read new_n
        n=$new_n
        echo "请输入资源种类m:"
        read new_m
        m=$new_m
        unset Max
        unset Allocation
        declare -a Max
        declare -a llocation
        echo "请输入最大需求资源(MAX)和分配资源(Allocation):"
        for i in `seq 0 $[$n-1]`; do
            echo "请输入P$[$i+1]进程的最大需求资源(MAX) (长度
$m)"
```

```
                read -a arr1
                for j in `seq 0 $[$m-1]`; do
                    site=$[$i*$m+$j]
                    Max[${site}]=${arr1[$j]}
                done

                echo "请输入P$[$i+1]进程的分配资源(Allocation)(长度$m)"
                read -a arr2
                for j in `seq 0 $[$m-1]`; do
                    site=$[$i*$m+$j]
                    Allocation[${site}]="${arr2[$j]}"
                done
            done
            echo "请输入可用资源(Available):"
            read -a new_Available
            Available=("${new_Available[@]}")

            echo "Max : ${Max[@]}   len: ${#Max[@]}"
            echo "Allocation : ${Allocation[@]}   len: ${#Allocation[@]}"
            echo "Available : ${Available[@]}"

            echo "算法开始:"
            ;;
```

```
    * )
        echo "输入无效,请输入 '1' 或 '2' 选择!"
        exit
        ;;
esac
#初始化
INIT
CALRESULT
while [[ 0 ]]; do
    CALREQUEST
    echo "--------------下一时刻---------------"
done
```

```
-------------------------func.sh-------------------------------
```

```
#! /bin/bash

#初始化
INIT(){
    #安全序列
    declare -a SafeOrder
    OranAvailable=("${Available[@]}")
    #需求资源
    declare -a Need
    #请求资源
    declare -a Request
```

```
}

#根据Max和Allocation计算Need
CALNEED(){
    local len=$[$m*$n-1]
    for i in `seq 0 $len`; do
        Need[$i]=$[${Max[$i]}-${Allocation[$i]}]
        if [[ ${Need[$i]} -lt 0 ]]; then
            echo "错误!Need资源不能为负值."
            echo "错误!MAX资源不应该小于Allocation资源."
        fi
    done
    #echo "Need are ${Need[@]}"
    #echo "Available are ${Available[@]}"
}

#银行家算法
CALRESULT(){
    #计算Need资源
    CALNEED
    #进程个数n减1
    local len1=$[$n-1]
    #资源种类m减1
    local len2=$[$m-1]
```

```
#判断是否安全
until [[ "${#SafeOrder[@]}" -eq n ]]; do
    for i in `seq 0 "${len1}"`; do
        #存在进程Need小于Available安全,释放资源,将该进程加入到安全队列
        for j in `seq 0 "${len2}"`; do
            #定位当前资源位置
            site=$[$i*$m+$j]
            if [[ "${Need[$site]}" -gt "${Available[$j]}" ]]; then
                #当前进程不合适切换到下一进程
                continue 2
            fi
        done
        #当前进程安全
        SafeOrder[${#SafeOrder[@]}]="$i"
        for k in `seq 0 ${len2}`; do
            #修改Need，Available
            site=$[$i*$m+$k]
            #标记999为已经判断过
            Need[$site]=999

Available[$k]=$[${Available[$k]}+${Allocation[$site]}]
        done
        continue 2
```

```
            done
            echo "当前状态不安全"
            exit
        done
        echo "当前状态安全"
        echo -n "安全序列为:"
        for (( l = 0; l < "${#SafeOrder[@]}"; l++ )); do
            echo -n "  P${SafeOrder[$l]}"
        done
        echo
}
#调用request资源请求
CALREQUEST(){
        echo "是否添加输入下一时刻进程资源请求Request?"
        echo "1-添加资源请求  2-不添加资源请求"
        read ifRequest
        case "${ifRequest}" in
            1 )
                echo -n "请输入进程ID:(可供选择的进程ID有"
                for idNumber in `seq 0 $n`; do
                    echo -n "${idNumber}"
                done
                echo ")"
                read ifProcessId
```

```
echo "请输入资源请求向量:(长度为$m)"
read -a requestVector
Request=("${requestVector[@]}")
#执行状态
Available=("${OranAvailable[@]}")
unset SafeOrder
state=0
len_m=$[$m-1]
#如果Requesti[j]>Need[i,j],资源数已超过它所宣布的最大值
for i in `seq 0 $len_m`; do
    site=$[${ifProcessId}*$m+$i]
    need=$[${Max[$site]}-${Allocation[$site]}]
    if [[ "${Request[$i]}" -gt "${need}" ]]; then
        echo "资源数已超过它所宣布的最大值!"
        state=1
        break
    fi
done
#Requesti[j]>Available[j],尚无足够资源, Pi须等待
if [[ "$state" -eq 0 ]]; then
    for i in `seq 0 $len_m`; do
        if [[ "${Request[$i]}" -gt "${Available[$i]}" ]]; then
```

```
                                echo "尚无足够资源,
P${ifProcessId}须等待.(当前可用资源: ${Available[@]})"
                                state=1
                                break
                            fi
                        done
                    fi
                    if [[ "$state" -eq 0 ]]; then
                        #更新状态
                        for i in `seq 0 $len_m`; do
                            site=$[${ifProcessId}*$m+$i]
                            Available[$i]=$[${Available[$i]}-
${Request[$i]}]

      Allocation[$site]=$[${Allocation[$site]}+${Request[$i]}]
                            #Need[$site]=$[${Allocation[$site]}-
${Request[$i]}]
                        done
                        #执行银行家算法
                        CALRESULT
                    fi
                    ;;
                2 )
                    echo "无新请求,程序退出."
                    exit
                    ;;
```

```
        * )
                echo "无效输入!"
                exit
                ;;
    esac
}
```