2017/11/29

操作系统实验报告

实验四 动态分区分配算法

软件工程一班_秦源_1525161007

一　需求分析

问题描述：

设计程序模拟四种动态分区分配算法：首次适应算法、循环首次适应算法、最佳适应算法和最坏适应算法的工作过程。假设内存中空闲分区个数为 n，空闲分区大小分别为 P1，…，Pn，在动态分区分配过程中需要分配的进程个数为 m（m≤n），它们需要的分区大小分别为 S1，…，Sm，分别利用四种动态分区分配算法将 m 个进程放入 n 个空闲分区，给出进程在空闲分区中的分配情况。

程序要求：

1）利用首次适应算法、循环首次适应算法、最佳适应算法和最坏适应算法四种动态分区分配算法模拟分区分配过程。

2）模拟四种算法的分区分配过程，给出每种算法进程在空闲分区中的分配情况。

3）输入：空闲分区个数 n，空闲分区大小 P1，…，Pn，进程个数 m，进程需要的分区大小 S1，…，Sm，算法选择 1-首次适应算法，2-循环首次适应算法，3-最佳适应算法，4-最坏适应算法。

4）输出：最终内存空闲分区的分配情况。

二　概要设计

1）首次适应算法

在分配内存时，从链首开始顺序查找，直至找到一个大小能满足要求的空闲分区为止；

然后再按照作业的大小，从该分区中划出一块内存空间分配给请求者，余下的空闲分区仍留在空闲链中。

若从链首直至链尾都不能找到一个能满足要求的分区，则此次内存分配失败，返回。

2）循环首次适应算法(Next Fit)

在为进程分配内存空间时，不再是每次都从链首开始查找，而是从上次找到的空闲分区的下一个空闲分区开始查找，直至找到一个能满足要求的空闲分区，从中划出一块与请求大小相等的内存空间分配给作业。

3）最佳适应算法(Best Fit)

所谓“最佳”是指每次为作业分配内存时，总是把能满足要求、又是最小的空闲分区分配给作业，避免“大材小用”。

4）最坏适应算法(Worst Fit)

最坏适应分配算法要扫描整个空闲分区表或链表，总是挑选一个最大的空闲区分割给作业使用。

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

首次适应算法倾向于优先利用内存中低址部分的空闲分区，从而保留了高址部分的大空闲区。这给为以后到达的大作业分配大的内存空间创造了条件。其缺点是低址部分不断被划分，会留下许多难以利用的、很小的空闲分区，而每次查找又都是从低址部分开始，这无疑会增加查找可用空闲分区时的开销。

循环首次适应算法能使内存中的空闲分区分布得更均匀，从而减少了查找空闲分区时的开销，但这样会缺乏大的空闲分区。

最佳适应算法似乎是最佳的，然而在宏观上却不一定。因为每次分配后所切割下来的剩余部分总是最小的，这样，在存储器中会留下许多难以利用的小空闲区。

最坏适应算法优点是可使剩下的空闲区不至于太小，产生碎片的几率最小，对中、小作业有利，同时最坏适应分配算法查找效率很高。但是该算法的缺点也是明显的，它会使存储器中缺乏大的空闲分区。

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

1）首次适应算法

```
bogon:SourceCode qinyuan$ ./main.sh
实验二　动态分区分配算法
----------默认数据-------------
进程名            A  B  C  D  E
进程分区大小      12 10 22 15 6
空闲分区名        P1 P2 P3 P4 P5
空闲分区大小      16 16 32 64 20
------------------------------
实验数据选择 1-使用默认数据，2-输入新数据
1
算法选择：1-首次适应算法 2-循环首次适应算法 3-最佳适应算法 4-最坏适应算法
1
------首次适应算法-----
内存空闲分区的分配情况：
进程A分配至空闲分区：  P1
进程B分配至空闲分区：  P2
进程C分配至空闲分区：  P3
进程D分配至空闲分区：  P4
进程E分配至空闲分区：  P2
分配结束后空闲分区大小情况：
空闲分区P1剩余空间：4K
空闲分区P2剩余空间：0K
空闲分区P3剩余空间：10K
空闲分区P4剩余空间：49K
空闲分区P5剩余空间：20K
```

2）循环首次适应算法(Next Fit)

```
bogon:SourceCode qinyuan$ ./main.sh
实验二  动态分区分配算法
----------默认数据-------------
进程名           A  B  C  D  E
进程分区大小     12 10 22 15 6
空闲分区名       P1 P2 P3 P4 P5
空闲分区大小     16 16 32 64 20
--------------------------------
实验数据选择 1-使用默认数据，2-输入新数据
1
算法选择：1-首次适应算法 2-循环首次适应算法 3-最佳适应算法 4-最坏适应算法
2
------循环首次适应算法-----
内存空闲分区的分配情况：
进程A分配至空闲分区：  P1
进程B分配至空闲分区：  P2
进程C分配至空闲分区：  P3
进程D分配至空闲分区：  P4
进程E分配至空闲分区：  P5
分配结束后空闲分区大小情况：
空闲分区P1剩余空间：4K
空闲分区P2剩余空间：6K
空闲分区P3剩余空间：10K
空闲分区P4剩余空间：49K
空闲分区P5剩余空间：14K
```

3）最佳适应算法(Best Fit)

```
bogon:SourceCode qinyuan$ ./main.sh
实验二  动态分区分配算法
----------默认数据-------------
进程名           A  B  C  D  E
进程分区大小     12 10 22 15 6
空闲分区名       P1 P2 P3 P4 P5
空闲分区大小     16 16 32 64 20
--------------------------------
实验数据选择 1-使用默认数据，2-输入新数据
1
算法选择：1-首次适应算法 2-循环首次适应算法 3-最佳适应算法 4-最坏适应算法
3
------最佳适应算法-----
内存空闲分区的分配情况：
进程A分配至空闲分区：  P1
进程B分配至空闲分区：  P2
进程C分配至空闲分区：  P3
进程D分配至空闲分区：  P5
进程E分配至空闲分区：  P2
分配结束后空闲分区大小情况：
空闲分区P1剩余空间：4K
空闲分区P2剩余空间：0K
空闲分区P3剩余空间：10K
空闲分区P4剩余空间：64K
空闲分区P5剩余空间：5K
```

4) 最坏适应算法(Worst Fit)

```
bogon:SourceCode qinyuan$ ./main.sh
实验二  动态分区分配算法
----------默认数据-------------
进程名           A  B  C  D  E
进程分区大小     12 10 22 15 6
空闲分区名       P1 P2 P3 P4 P5
空闲分区大小     16 16 32 64 20
--------------------------------
实验数据选择 1-使用默认数据, 2-输入新数据
1
算法选择: 1-首次适应算法 2-循环首次适应算法 3-最佳适应算法 4-最坏适应算法
4
------最坏适应算法-----
内存空闲分区的分配情况:
进程A分配至空闲分区:  P4
进程B分配至空闲分区:  P4
进程C分配至空闲分区:  P4
进程D分配至空闲分区:  P3
进程E分配至空闲分区:  P4
分配结束后空闲分区大小情况:
空闲分区P1剩余空间: 16K
空闲分区P2剩余空间: 16K
空闲分区P3剩余空间: 17K
空闲分区P4剩余空间: 14K
空闲分区P5剩余空间: 20K
```

七　附录

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

```
#! /bin/bash

. func.sh

#进程名对照数组

PID=('A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L')

#空闲分区个数为n

n=5

#空闲分区大小P1,…,Pn

P=(16 16 32 64 20)

#进程个数m
```

```
m=5
#进程需要的分区大小S1,…,Sm
S=(12 10 22 15 6)
echo "实验二  动态分区分配算法"
echo "----------默认数据-------------"
echo -en "进程名\t\t"
for i in `seq 0 $[$m-1]`; do
     echo -en "${PID[$i]}  "
done
echo
echo -e "进程分区大小\t${S[@]}"
echo -en "空闲分区名  \t"
for i in `seq 0 $[$n-1]`; do
     echo -en "P$[$i+1] "
done
echo
echo -e "空闲分区大小\t${P[@]}"
echo "------------------------------"
echo "实验数据选择 1-使用默认数据，2-输入新数据"

read keypressData
case "$keypressData" in
     1 )
          ;;
```

```
    2 )
        echo "请输入空闲分区个数:"
        read new_n
        n=new_n
        echo "请按顺序输入空闲分区大小:(长度$n)"
        read -a new_P
        P=("${new_P[@]}")
        echo "请输入进程个数个数:"
        read new_m
        m=new_m
        echo "请按顺序输入进程分区大小:(长度$m)"
        read -a new_S
        S=("${new_S[@]}")
        ;;
    * )
        echo "输入无效,请输入 '1' 或 '2' 选择!"
        exit
        ;;
esac
echo "算法选择: 1-首次适应算法 2-循环首次适应算法 3-最佳适应算法 4-最坏适应算法"
read keypressKind
case "$keypressKind" in
    1 )
        FF
```

```
            ;;
        2 )
            NF
            ;;
        3 )
            BF
            ;;
        4 )
            WF
            ;;
        * )
            echo "输入无效,请输入 '1' 或 '2' 或 '3'或 '4' 选择!"
            exit
            ;;
esac
```

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

```
#! /bin/bash
INIT(){
    #剩余空间
    declare -a FreePartition
    #首次适应算法结果ID;-1为分配失败
    declare -a FirstPartition
    #循环首次适应算法ID;-1为分配失败
    declare -a CycleFirstPartition
```

```
    #最佳适应算法结果ID;-1为分配失败
    declare -a BestPartition
    #最坏适应算法结果ID;-1为分配失败
    declare -a WorstPartition
}
#剩余空间输出
LEFT(){
    echo "分配结束后空闲分区大小情况:"
    #循环空闲分区个数为n
    for i in `seq 0 "$[$n-1]"`; do
        echo "空闲分区P$[$i+1]剩余空间: ${FreePartition[$i]}K"
    done
}
#1-首次适应算法
FF(){
    INIT
    FreePartition=("${P[@]}")
    #是否分配失败,失败为1
    local success=1
    #循环进程个数m
    for i in `seq 0 "$[$m-1]"`; do
        success=1
        #循环空闲分区个数为n
        for j in `seq 0 "$[$n-1]"`; do
```

```
                #S[i]进程需要分区小于等于空闲分区大小
FreePartition[i]
                if [[ "${S[$i]}" -le "${FreePartition[$j]}" ]];
then
                    success=0
                    FirstPartition[$i]=$j
                    FreePartition[$j]=$[${FreePartition[$j]}-
${S[$i]}]
                    break
                fi
            done
            #分配失败
            if [[ "$success" -eq 1 ]]; then
                FirstPartition[$i]=-1
            fi
        done
        echo -e "------首次适应算法-----\n内存空闲分区的分配情况:"
        #循环进程个数m
        for i in `seq 0 "$[$m-1]"`; do
            if [[ ${FirstPartition[$i]} -ge 0 ]]; then
                echo -e "进程${PID[$i]}分配至空闲分区:
P$[${FirstPartition[$i]}+1]"
            else
                echo "进程${PID[$i]}分配至空闲分区失败"
            fi
        done
```

```
	LEFT
}
#2-循环首次适应算法
NF(){
	INIT
	FreePartition=("${P[@]}")
	#是否分配失败,失败为1
	local success=1
	#循环进程个数m
	#定义循环标记
	CycleIndex=0
	for i in `seq 0 "$[$m-1]"`; do
		success=1
		#循环空闲分区个数为n
		for j in `seq 0 "$[$n-1]"`; do
			#S[i]进程需要分区小于等于空闲分区大小 FreePartition[i]
			if [[ "${S[$i]}" -le "${FreePartition[${CycleIndex}]}" ]]; then
				success=0
				CycleFirstPartition[$i]="${CycleIndex}"

	FreePartition[${CycleIndex}]=$[${FreePartition[${CycleIndex}]}-${S[$i]}]
				CycleIndex=$[(${CycleIndex}+1)%$n]
				break
```

```
                fi
                CycleIndex=$[(${CycleIndex}+1)%$n]
            done
            #分配失败
            if [[ "$success" -eq 1 ]]; then
                CycleFirstPartition[$i]=-1
            fi
        done
        echo -e "------循环首次适应算法-----\n内存空闲分区的分配情况:"
        #循环进程个数m
        for i in `seq 0 "$[$m-1]"`; do
            if [[ ${CycleFirstPartition[$i]} -ge 0 ]]; then
                echo -e "进程${PID[$i]}分配至空闲分区:P$[${CycleFirstPartition[$i]}+1]"
            else
                echo "进程${PID[$i]}分配至空闲分区失败"
            fi
        done
        LEFT
}
#3-最佳适应算法
BF(){
        INIT
        FreePartition=("${P[@]}")
```

```
#是否分配失败,失败为1
local success=1
#循环进程个数m
for i in `seq 0 "$[$m-1]"`; do
    success=1
    #定义最佳标记
    BestIndex=999
    BestSize=999
    #循环空闲分区个数为n
    for j in `seq 0 "$[$n-1]"`; do
        #S[i]进程需要分区小于等于空闲分区大小 FreePartition[i]
        if [[ "${S[$i]}" -le "${FreePartition[$j]}" ]]; then
            if [[ "${BestSize}" -gt ${FreePartition[$j]} ]]; then
                BestIndex="$j"
                BestSize="${FreePartition[$j]}"
            fi
        fi
    done
    if [[ "${BestIndex}" -ne 999 ]]; then
        success=0
        BestPartition[$i]="${BestIndex}"
```

```
        FreePartition[${BestIndex}]=$[${FreePartition[${BestIndex}]
}-${S[$i]}]
            fi
            #分配失败
            if [[ "$success" -eq 1 ]]; then
                BestPartition[$i]=-1
            fi
        done
        echo -e "------最佳适应算法-----\n内存空闲分区的分配情况:"
        #循环进程个数m
        for i in `seq 0 "$[$m-1]"`; do
            if [[ ${BestPartition[$i]} -ge 0 ]]; then
                echo -e "进程${PID[$i]}分配至空闲分区:
P$[${BestPartition[$i]}+1]"
            else
                echo "进程${PID[$i]}分配至空闲分区失败"
            fi
        done
        LEFT
}
#4-最坏适应算法
WF(){
        INIT
        FreePartition=("${P[@]}")
```

```
#是否分配失败,失败为1
local success=1
#循环进程个数m
for i in `seq 0 "$[$m-1]"`; do
    success=1
    #定义最坏标记
    WorstIndex=-1
    WorstSize=-1
    #循环空闲分区个数为n
    for j in `seq 0 "$[$n-1]"`; do
        #S[i]进程需要分区小于等于空闲分区大小
FreePartition[i]
        if [[ "${S[$i]}" -le "${FreePartition[$j]}" ]];
then
            if [[ "${WorstSize}" -lt
${FreePartition[$j]} ]]; then
                WorstIndex="$j"
                WorstSize="${FreePartition[$j]}"
            fi
        fi
    done
    if [[ "${WorstIndex}" -ne -1 ]]; then
        success=0
        WorstPartition[$i]="${WorstIndex}"
```

```
        FreePartition[${WorstIndex}]=$[${FreePartition[${WorstIndex
}]}-${S[$i]}]
            fi
            #分配失败
            if [[ "$success" -eq 1 ]]; then
                WorstPartition[$i]=-1
            fi
        done
        echo -e "------最坏适应算法-----\n内存空闲分区的分配情况:"
        #循环进程个数m
        for i in `seq 0 "$[$m-1]"`; do
            if [[ ${WorstPartition[$i]} -ge 0 ]]; then
                echo -e "进程${PID[$i]}分配至空闲分区:
P$[${WorstPartition[$i]}+1]"
            else
                echo "进程${PID[$i]}分配至空闲分区失败"
            fi
        done
        LEFT
}
```