实验四 --段页式虚拟存储管理

- 姓名：庞晓宇
- 学号：2020118100

实验目的

1. 加深理解段页式虚拟存储管理的概念和原理。
2. 掌握段页式存储管理中存储分配（和回收）方法；
3. 深入了解段页式虚拟存储管理中地址重定位（即地址映射）方法。
4. 深入理解段页式虚拟存储管理中缺段、缺页中断处理方法。

目录结构

```
SPVSM\SRC\MAIN\JAVA
│  config.ini
│
└─com
    └─xftxyz
        └─spvsm
            ├─controller
            │  ├─activity
            │  │      FileUtil.java
            │  │
            │  └─service
            │          OS.java
            │          XFSetting.java
            │
            ├─model
            │  └─domain
            │          Frame.java
            │          Memory.java
            │          PageEntry.java
            │          PCB.java
            │          SegmentEntry.java
            │
            └─view
                ├─ui
                │      Shell.java
                │
                └─utils
                        Input.java
```

实验内容

编写程序完成段页式虚拟存储管理存储分配、地址重定位和缺页中断处理。

1. 为一个进程的内存申请（多少个段，每个段多大）分配内存，当一个进程（完成）结束时回收内存；
2. 对一个给定逻辑地址，判断其是否缺段、缺页，若不缺段、不缺页，则映射出其物理地址；
3. 若缺段则进行缺段中断处理，若缺页则进行缺页中断处理。

假定内存64K，内存块（页框）大小为1K，进程逻辑地址空间最多4个段，每个段最大16K，进程驻留集大小为8页。假设进程运行前未预先装入任何地址空间，页面淘汰策略采用局部（驻留集内）置换策略。

```
memorySize=65536
pageSize=1024
maxSegmentNum=4
maxSegmentSize=16384
maxResidentSetNum=8
```

输出每次存储分配/回收时，内存自由块分布情况、相关进程的段表和页表信息。

```
>>> create process 1 13 15 12 7
IO: 将进程 1 段(0) 页(0) 读入页框 0 中
IO: 将进程 1 段(1) 页(0) 读入页框 1 中
IO: 将进程 1 段(2) 页(0) 读入页框 2 中
IO: 将进程 1 段(3) 页(0) 读入页框 3 中
创建进程 1 成功
>>> show memory
内存使用情况:
0-7:    | 1     | 1     | 1     | 1     |       |       |       |       |
8-15:   |       |       |       |       |       |       |       |       |
16-23:  |       |       |       |       |       |       |       |       |
24-31:  |       |       |       |       |       |       |       |       |
32-39:  |       |       |       |       |       |       |       |       |
40-47:  |       |       |       |       |       |       |       |       |
48-55:  |       |       |       |       |       |       |       |       |
56-63:  |       |       |       |       |       |       |       |       |
>>> create process 3 564 123 789
IO: 将进程 3 段(0) 页(0) 读入页框 4 中
IO: 将进程 3 段(1) 页(0) 读入页框 5 中
IO: 将进程 3 段(2) 页(0) 读入页框 6 中
创建进程 3 成功
>>> create process 4 1164 1223 1789
IO: 将进程 4 段(0) 页(0) 读入页框 7 中
IO: 将进程 4 段(0) 页(1) 读入页框 8 中
IO: 将进程 4 段(1) 页(0) 读入页框 9 中
IO: 将进程 4 段(1) 页(1) 读入页框 10 中
IO: 将进程 4 段(2) 页(0) 读入页框 11 中
IO: 将进程 4 段(2) 页(1) 读入页框 12 中
创建进程 4 成功
>>> show memory
内存使用情况:
0-7:    | 1     | 1     | 1     | 1     | 3     | 3     | 3     | 4     |
8-15:   | 4     | 4     | 4     | 4     | 4     |       |       |       |
16-23:  |       |       |       |       |       |       |       |       |
24-31:  |       |       |       |       |       |       |       |       |
```

```
32-39:  |       |       |       |       |       |       |       |       |
40-47:  |       |       |       |       |       |       |       |       |
48-55:  |       |       |       |       |       |       |       |       |
56-63:  |       |       |       |       |       |       |       |       |
>>> destroy process 1
销毁进程1成功
>>> show memory
内存使用情况:
0-7:    |       |       |       |       | 3     | 3     | 3     | 4     |
8-15:   | 4     | 4     | 4     | 4     | 4     |       |       |       |
16-23:  |       |       |       |       |       |       |       |       |
24-31:  |       |       |       |       |       |       |       |       |
32-39:  |       |       |       |       |       |       |       |       |
40-47:  |       |       |       |       |       |       |       |       |
48-55:  |       |       |       |       |       |       |       |       |
56-63:  |       |       |       |       |       |       |       |       |
>>> create process 5 1164 4223 3789
IO: 将进程 5 段(0) 页(0) 读入页框 0 中
IO: 将进程 5 段(0) 页(1) 读入页框 1 中
IO: 将进程 5 段(1) 页(0) 读入页框 2 中
IO: 将进程 5 段(1) 页(1) 读入页框 3 中
IO: 将进程 5 段(1) 页(2) 读入页框 13 中
IO: 将进程 5 段(1) 页(3) 读入页框 14 中
IO: 将进程 5 段(1) 页(4) 读入页框 15 中
IO: 将进程 5 段(2) 页(0) 读入页框 16 中
创建进程 5 成功
>>> show memory
内存使用情况:
0-7:    | 5     | 5     | 5     | 5     | 3     | 3     | 3     | 4     |
8-15:   | 4     | 4     | 4     | 4     | 4     | 5     | 5     | 5     |
16-23:  | 5     |       |       |       |       |       |       |       |
24-31:  |       |       |       |       |       |       |       |       |
32-39:  |       |       |       |       |       |       |       |       |
40-47:  |       |       |       |       |       |       |       |       |
48-55:  |       |       |       |       |       |       |       |       |
56-63:  |       |       |       |       |       |       |       |       |
```

拓展

- 采用LRU页面置换算法实现页面淘汰。

提示

1. 内存状态描述
    1. 分块（页框）说明表内容：编号、状态
    2. 组织方式：线性表，位图?
    3. 设置初始内存分配状态：随机设定若干块为已分配。
2. 段表、页表设计及其关系
3. 逻辑地址的表示
4. 缺段、缺页中断处理中的页面淘汰
    1. 使用最简单的FIFO策略，选择要淘汰的页

测试输出

- 输出当前内存分配情况（有多少可用块、哪些块可用？）；

```
>>> show memory
内存使用情况:
0-7:    | 5     | 5     | 5     | 5     | 3     | 3     | 3     | 4     |
8-15:   | 4     | 4     | 4     | 4     | 4     | 5     | 5     | 5     |
16-23:  | 5     | 6     | 6     | 6     | 6     | 6     | 6     | 6     |
24-31:  | 6     |       |       |       |       |       |       |       |
32-39:  |       |       |       |       |       |       |       |       |
40-47:  |       |       |       |       |       |       |       |       |
48-55:  |       |       |       |       |       |       |       |       |
56-63:  |       |       |       |       |       |       |       |       |
```

- 手工输入进程的内存总需求（多少段，每个段多大）；
- 手工输入某进程的内存申请（哪几个段，各自需要多少块？），输出系统为其分配内存后的段表和页表内容。

```
>>> create process 6 6048 7192 5681
IO: 将进程 6 段(0) 页(0) 读入页框 17 中
IO: 将进程 6 段(0) 页(1) 读入页框 18 中
IO: 将进程 6 段(0) 页(2) 读入页框 19 中
IO: 将进程 6 段(0) 页(3) 读入页框 20 中
IO: 将进程 6 段(0) 页(4) 读入页框 21 中
IO: 将进程 6 段(0) 页(5) 读入页框 22 中
IO: 将进程 6 段(1) 页(0) 读入页框 23 中
IO: 将进程 6 段(1) 页(1) 读入页框 24 中
创建进程 6 成功
```

- 模拟内存访问指令的地址映射：比如手工输入一个逻辑地址，系统提示是否缺段、缺页，若不缺段、不缺页，则输出其物理地址；

```
>>> address 5 5 5
操作失败，进程5 段(5)不存在
>>> address 5 2 3
进程5段(2) 段偏移(3) 物理地址为: 16387
```

- 若缺段或缺页，则输出装入（被淘汰）的块号，输出缺段或缺页中断处理后的段表和页表信息。

```
>>> h
create process 进程id 各个段大小          创建一个进程
destroy process 进程id           销毁一个进程
show memory                       显示内存使用情况
show process 进程id               显示该进程驻留集、置换策略、段表、页表
```

```
address 进程名 段号 段偏移                将逻辑地址映射为物理地址
help or h                        获取帮助
quit or q                        退出
```