实验三 实现请求页式存储管理模拟程序

1120152035 王奥博

> 语言:python2.7
>
> OS:deepin-linux 15.5 amd64

数据结构及符号分析:

本次实验定义了一个类Memory用来表示内存页,类变量**method表示算法(OPT,FIFO,LRU), size表示内存页的大小, page表示当前页的占用情况(最后一位表示当前是否命中), ans用于暂时存储每次请求后的输出序列, pageFault表示缺页的次数**

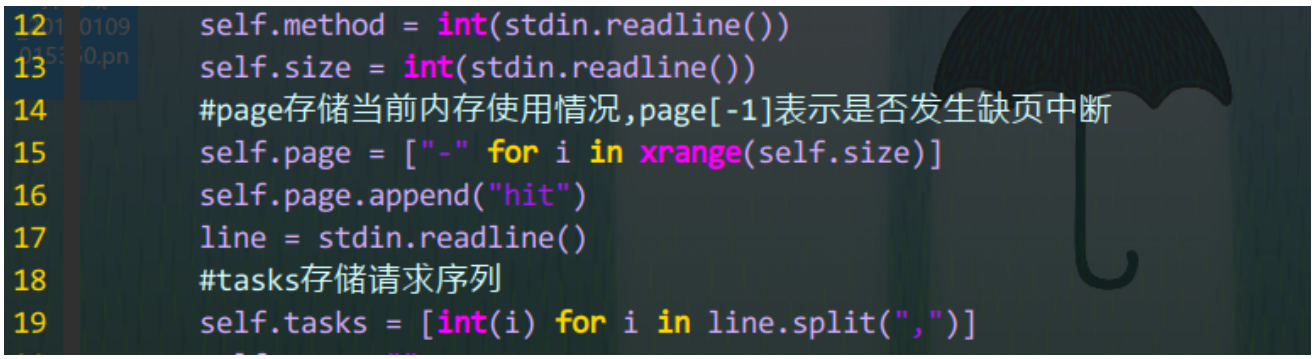
如下:

```
10 class Memory(object):
11     def __init__(self):
12         self.method = int(stdin.readline())
13         self.size = int(stdin.readline())
14         #page存储当前内存使用情况,page[-1]表示是否发生缺页中断
15         self.page = ["-" for i in xrange(self.size)]
16         self.page.append("hit")
17         line = stdin.readline()
18         #tasks存储请求序列
19         self.tasks = [int(i) for i in line.split(",")]
20         self.ans = ""
21         self.pageFault = 0
22         #  print self.method
23         #  print self.size
24         #  pprint(self.page)
25         #  pprint(self.tasks)
26
```

其中,使用**列表tasks存储请求的页面序列**

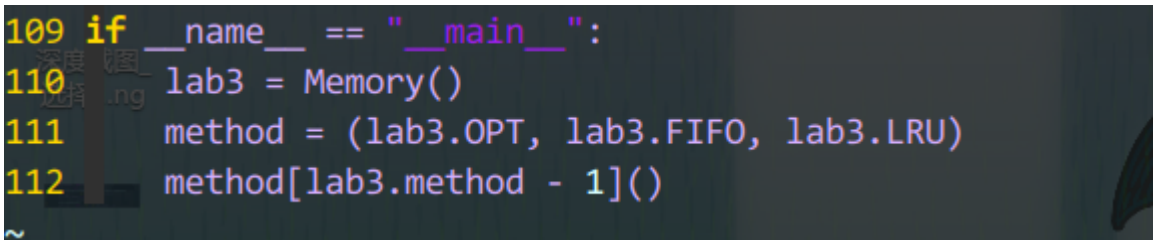
调度算法的处理流程

此次实验比起前两次的实验输入简单,只需读取三行即可,使用了os.stdin.readline()获取输入,读取的页面序列使用生成列表存储到了self.tasks中,如下:



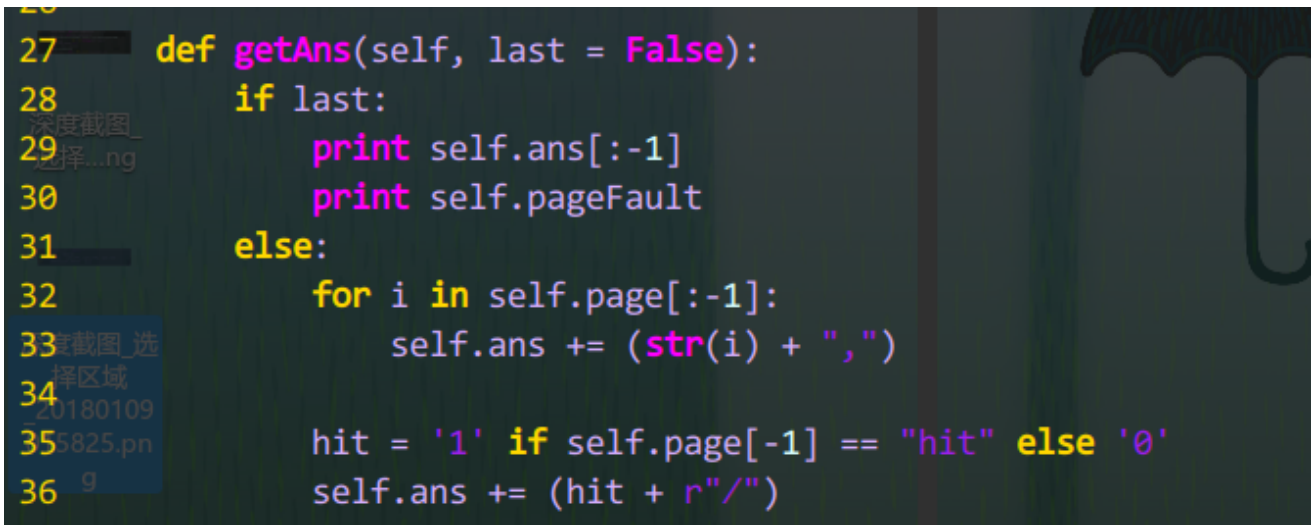
```
12      self.method = int(stdin.readline())
13      self.size = int(stdin.readline())
14      #page存储当前内存使用情况,page[-1]表示是否发生缺页中断
15      self.page = ["-" for i in xrange(self.size)]
16      self.page.append("hit")
17      line = stdin.readline()
18      #tasks存储请求序列
19      self.tasks = [int(i) for i in line.split(",")]
```

类似于前两种方法,在选取算法上,使用了元组+下标的方法:



```
109 if __name__ == "__main__":
110     lab3 = Memory()
111     method = (lab3.OPT, lab3.FIFO, lab3.LRU)
112     method[lab3.method - 1]()
```

同时定义了一个**getAns函数**用于更新操作结果或者输出,通过传入的参数决定使用哪种方法:



```
27  def getAns(self, last = False):
28      if last:
29          print self.ans[:-1]
30          print self.pageFault
31      else:
32          for i in self.page[:-1]:
33              self.ans += (str(i) + ",")
34
35          hit = '1' if self.page[-1] == "hit" else '0'
36          self.ans += (hit + r"/")
```

算法调度:

本次试验完成了三种算法:OPT,FIFO和LRU,分别解释如下:

OPT:

借助代码解释:

```
38  def OPT(self):
39      #  pdb.set_trace()
40      for idx, t in enumerate(self.tasks):
41          #  if idx == 10:
42              #  pdb.set_trace()
43          hit = "hit" if t in self.page else "unhit"
44          if hit == "unhit":
45              dis = []
46              for i, j in enumerate(self.page[: -1]):
47                  x = self.tasks[idx + 1: ].index(j) if j in self.tasks[idx + 1: ] else 0xff
48                  y = self.tasks[: idx][::-1].index(j) if j in self.tasks[: idx] else 0xff
49                  dis.append((x, y, i))
50
51              #  pdb.set_trace()
52              dis = copy(sorted(dis, key = lambda x: (-x[0], -x[1])))
53              fix = dis[0][-1]
54              self.page[fix] = t
55              self.pageFault += 1
56
57          self.page[-1] = hit
58          self.getAns()
59
60      self.getAns(True)
```

整体结构是循环处理所有的请求序列,对每一个请求:

- 先判断该请求是否命中(第43行)
- 若该请求没有命中,则寻找符合OPT条件的内存页,然后将该内存块更新为请求,同时将缺页次数加1(44~55行)

> OPT条件:
>
> - 寻找下次访问距当前页最远的页(第47行)
> - 若有多个符合条件的页,则选择最早进入的页(第48行)

> 这里需要解释48行,寻找下次访问距离当前页最远的页时,只需从当前下标开始寻找下标最大的点即可,但此时可能会有多个页符合条件,需要再次根据进入时间选择最优解.
>
> 寻找进入时间最早的页只需从当前下标向前找最近的请求序列即可

- 讲self.page的最后以为更新为是否命中
- 调用self.getAns更新答案
- 循环完成后,调用self.getAns(True)输出结果

FIFO

FIFO比起OPT较简单,只需按顺序处理每次请求页即可,若没有命中,则更新内存,若命中,只需更新self.page的最后一位,不再详细解释,代码如下:

```python
def FIFO(self):
    #  pdb.set_trace()
    idx = 0
    for t in self.tasks:
        hit = "hit" if t in self.page else "unhit"
        if hit == "unhit":
            self.pageFault += 1
            self.page[idx] = t

        self.page[-1] = hit
        self.getAns()
        idx = (idx + 1) % self.size if hit == "unhit" else idx

    self.getAns(True)
```

LRU

此处的LRU与上课时所讲的LRU有一些区别,此处的LRU规则是:

> 对每一个请求,若命中,则将请求中的页后移到内存的最后一位
>
> 若没有命中,则讲第一位淘汰,其余页向前进一位,新页放入最后一位

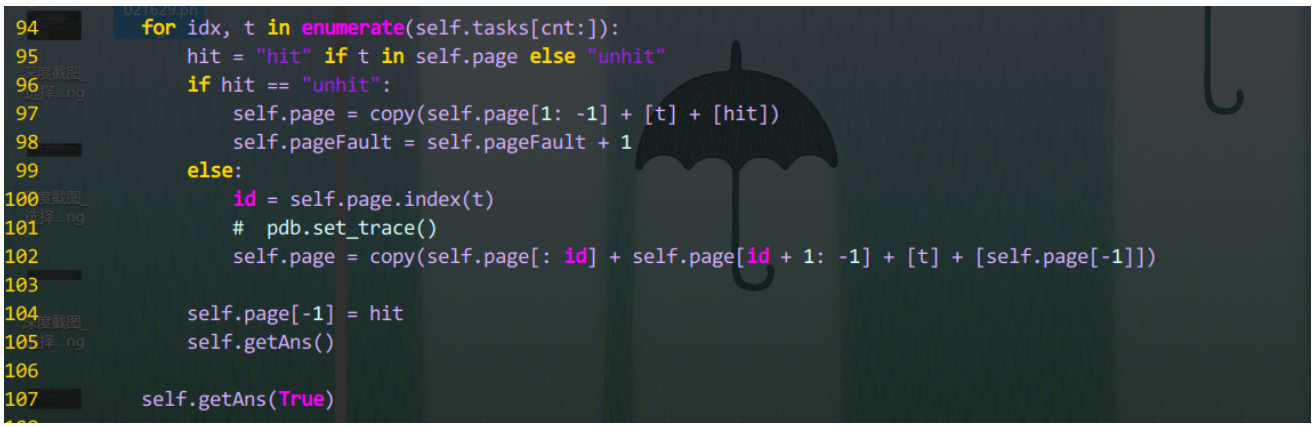
这里用列表模拟了一个栈简化操作,需要注意的是,在预留集未满时也可能出现命中的情况,为了方便 编码,我把这一块单独进行了处理:

```python
def LRU(self):
    #  pdb.set_trace()
    cnt = 0
    i = 0
    while "-" in self.page:
        hit = "hit" if self.tasks[cnt] in self.page else "unhit"
        if hit == "unhit":
            self.page[i] = self.tasks[cnt]
            self.pageFault += 1
            i += 1
        elif i > 1:
            self.page = copy(self.page[1: i] + [self.tasks[i]] + self.page[i:])

        cnt += 1
        self.page[-1] = hit
        self.getAns()
```

cnt为请求序列的下标,i为预留集的下标

当预留集满后,即可简单的模拟栈操作进行处理:



```
94      for idx, t in enumerate(self.tasks[cnt:]):
95          hit = "hit" if t in self.page else "unhit"
96          if hit == "unhit":
97              self.page = copy(self.page[1: -1] + [t] + [hit])
98              self.pageFault = self.pageFault + 1
99          else:
100             id = self.page.index(t)
101             #  pdb.set_trace()
102             self.page = copy(self.page[: id] + self.page[id + 1: -1] + [t] + [self.page[-1]])
103
104         self.page[-1] = hit
105         self.getAns()
106
107     self.getAns(True)
```

同样分成了命中和未命中两种

完整代码:

> 代码及测试用例均已上传到https://github.com/M4xW4n9/personal_repository/blob/master/OS/lab3/
>
> 使用
>
> wget https://raw.githubusercontent.com/M4xW4n9/personal_repository/master/OS/lab3/lab3.py
>
> 即可下载

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from sys import stdin
from copy import deepcopy as copy
from pprint import pprint
import pdb

class Memory(object):
    def __init__(self):
        self.method = int(stdin.readline())
        self.size = int(stdin.readline())
        #page存储当前内存使用情况,page[-1]表示是否发生缺页中断
        self.page = ["-" for i in xrange(self.size)]
        self.page.append("hit")
        line = stdin.readline()
        #tasks存储请求序列
        self.tasks = [int(i) for i in line.split(",")]
        self.ans = ""
        self.pageFault = 0
        #  print self.method
        #  print self.size
        #  pprint(self.page)
        #  pprint(self.tasks)

    def getAns(self, last = False):
        if last:
            print self.ans[:-1]
            print self.pageFault
        else:
            for i in self.page[:-1]:
                self.ans += (str(i) + ",")

            hit = '1' if self.page[-1] == "hit" else '0'
            self.ans += (hit + r"/")

    def OPT(self):
        #  pdb.set_trace()
        for idx, t in enumerate(self.tasks):
            #  if idx == 10:
                #  pdb.set_trace()
            hit = "hit" if t in self.page else "unhit"
            if hit == "unhit":
                dis = []
                for i, j in enumerate(self.page[: -1]):
                    x = self.tasks[idx + 1: ].index(j) if j in self.tasks[idx + 1: ] else
0xff
                    y = self.tasks[: idx][::-1].index(j) if j in self.tasks[: idx] else 0xff
                    dis.append((x, y, i))

                #  pdb.set_trace()

                dis = copy(sorted(dis, key = lambda x: (-x[0], -x[1])))
```

```
                fix = dis[0][-1]
                self.page[fix] = t
                self.pageFault += 1

            self.page[-1] = hit
            self.getAns()

        self.getAns(True)

    def FIFO(self):
        #  pdb.set_trace()
        idx = 0
        for t in self.tasks:
            hit = "hit" if t in self.page else "unhit"
            if hit == "unhit":
                self.pageFault += 1
                self.page[idx] = t

            self.page[-1] = hit
            self.getAns()
            idx = (idx + 1) % self.size if hit == "unhit" else idx

        self.getAns(True)

    def LRU(self):
        #  pdb.set_trace()
        cnt = 0
        i = 0
        while "-" in self.page:
            hit = "hit" if self.tasks[cnt] in self.page else "unhit"
            if hit == "unhit":
                self.page[i] = self.tasks[cnt]
                self.pageFault += 1
                i += 1
            elif i > 1:
                self.page = copy(self.page[1: i] + [self.tasks[i]] + self.page[i:])

            cnt += 1
            self.page[-1] = hit
            self.getAns()

        for idx, t in enumerate(self.tasks[cnt:]):
            hit = "hit" if t in self.page else "unhit"
            if hit == "unhit":
                self.page = copy(self.page[1: -1] + [t] + [hit])
                self.pageFault = self.pageFault + 1
            else:
                id = self.page.index(t)
                #  pdb.set_trace()
                self.page = copy(self.page[: id] + self.page[id + 1: -1] + [t] +
[self.page[-1]])


            self.page[-1] = hit
```

```
            self.getAns()

        self.getAns(True)

if __name__ == "__main__":
    lab3 = Memory()
    method = (lab3.OPT, lab3.FIFO, lab3.LRU)
    method[lab3.method - 1]()
```

本次实验经验及体会

本次实验较为简单,除了在LRU算法的处理中遇到了一些麻烦外没有遇到太大问题.

通过这次实验,对操作系统中存储管理的替换策略理解更加深刻了,同时尝试使用了python的默认形参***def getAns(self, last = False)***简化了编码