《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 |  | 学号 |  | 专业班级 |  | 时间 | 2023. |

**一、实验目的**

1. 理解页面淘汰算法原理,编写程序演示页面淘汰算法。
2. 验证Linux虚拟地址转化为物理地址的机制。
3. 理解和验证程序运行局部性的原理。
4. 理解和验证缺页处理的流程。

**二、实验内容**

1. Linux模拟实现OPT或FIFO或LRU淘汰算法。
2. Linux下利用/proc/pid/pagemap技术计算某个变量或函数虚拟地址对应的物理地址等信息。

**三、实验环境和核心代码**

**3.1 页面淘汰算法模拟**

程序使用数组模拟页框，页框大小有限，无法存入所有的页面，需要访问指定地址时，需要将地址所在页调入页框中，再从页框中进行访问。

页表结构也使用数组进行模拟，页表中的每一项有页号，页框号，计数器，访问计数器，缺页状态等。

1. #define PAGE\_SIZE 0x10    // 页大小
2. #define MEM\_SIZE 0x100    // 内存大小
3. #define PAGE\_FRAME\_SIZE 4 // 页框大小
5. #define REPLACE\_ALGORITHM lru\_algorithm // 使用的算法
6. #define SEQ 0                           // 顺序访问
7. #define JMP 1                           // 跳转访问
8. #define LOP 2                           // 循环访问
9. #define RDM 3                           // 随机访问
10. #define ACCESS\_TYPE JMP                 // 访问类型
12. #define INSTRUCTION\_NUM 0x64 // 指令条数
14. // 页表结构体
15. **struct** \_\_page\_item
16. {
17. **int** page\_index;        // 页号
18. **int** page\_frame\_index;  // 页框号
19. \_\_uint16\_t cnt;        // 计数器
20. \_\_uint16\_t access\_cnt; // 访问计数器
21. \_\_uint8\_t page\_fault;  // 缺页
22. };
24. \_\_uint8\_t process\_mem[MEM\_SIZE][PAGE\_SIZE];       // 进程内存
25. \_\_uint8\_t page\_frame[PAGE\_FRAME\_SIZE][PAGE\_SIZE]; // 页框
27. **struct** \_\_page\_item page\_table[PAGE\_FRAME\_SIZE]; // 页表
29. **int** ins\_seq[INSTRUCTION\_NUM]; // 访问次序
30. **int** cur\_ins\_index;            // 当前访问指令的位置
32. #define RECORD
34. #ifdef RECORD
35. **int** page\_fault\_cnt; // 缺页计数器
36. **int** replace\_record[INSTRUCTION\_NUM][3];    // 置换记录
37. #endif

程序访问内存时，对于给定的地址，首先分离出页号以及页内偏移，根据页号查找是否在页表中，如果在页表中，则提取出页框号，与页内偏移组合得到物理地址，如果访问的页不在页表中，则触发缺页异常，使用指定的页面调度算法将所需的页调入页框中，模拟内存访问的代码如下。

1. \_\_uint8\_t access\_mem(**int** addr, **int** (\*replace\_algorithm)(**int**))
2. {
3. **int** page\_index = addr / PAGE\_SIZE;
4. **int** page\_offset = addr % PAGE\_SIZE;
5. **if** (page\_index >= MEM\_SIZE || page\_index < 0)
6. {
7. // 访问的地址不存在
8. exit(-1);
9. }
10. // 查找页号是否在页表中
11. **int** page\_frame\_index = -1;
12. **for** (**int** i = 0; i < PAGE\_FRAME\_SIZE; i++)
13. {
14. **if** (page\_index == page\_table[i].page\_index)
15. {
16. page\_frame\_index = page\_table[i].page\_frame\_index;
17. // 访问的页面计数器置0
18. page\_table[i].access\_cnt = 0;
19. **break**;
20. }
21. }
22. **if** (page\_frame\_index == -1)
23. {
24. // 缺页异常
25. page\_frame\_index = replace\_algorithm(page\_index);
26. #ifdef RECORD
27. page\_fault\_cnt += 1;
28. #endif
29. }
30. // 所有页面访问计数器+1
31. **for** (**int** i = 0; i < PAGE\_FRAME\_SIZE; i++)
32. {
33. page\_table[i].access\_cnt += 1;
34. }
35. **return** page\_frame[page\_frame\_index][page\_offset];
36. }

最佳淘汰算法OPT查找将来最远会被使用到的页面，替换将来最远才会用到的页面。

1. **int** opt\_algorithm(**int** page\_index)
2. {
3. **int** replace\_index = 0;
4. **int** cnt = -1;
5. **for** (**int** i = 0; i < PAGE\_FRAME\_SIZE; i++)
6. {
7. // 优先查找空项
8. **if** (page\_table[i].page\_fault == 1)
9. {
10. page\_table[i].page\_fault = 0;
11. replace\_index = i;
12. **break**;
13. }
14. // 查找将来最远会被用到的页面
15. **int** cur\_page\_index = page\_table[i].page\_index;
16. **for** (**int** j = cur\_ins\_index + 1; j < INSTRUCTION\_NUM; j++)
17. {
18. **if** (ins\_seq[j] / PAGE\_SIZE == cur\_page\_index)
19. {
20. **if** (j - cur\_ins\_index > cnt)
21. {
22. cnt = j - cur\_ins\_index;
23. replace\_index = i;
24. }
25. **break**;
26. }
27. }
28. }
29. // 将页面调入页框
30. page\_table[replace\_index].page\_index = page\_index;
31. page\_table[replace\_index].cnt = 0;
32. page\_table[replace\_index].access\_cnt = 0;
33. **int** page\_frame\_index = page\_table[replace\_index].page\_frame\_index;
34. **for** (**int** i = 0; i < PAGE\_SIZE; i++)
35. {
36. page\_frame[page\_frame\_index][i] = process\_mem[page\_index][i];
37. }
38. **return** page\_frame\_index;
39. }

先进先出算法FIFO查找最先进入页表的项，将计数器值最大的页表换出。

1. **int** fifo\_algorithm(**int** page\_index)
2. {
3. **int** replace\_index = 0;
4. **int** cnt = -1;
5. **for** (**int** i = 0; i < PAGE\_FRAME\_SIZE; i++)
6. {
7. // 优先查找空项
8. **if** (page\_table[i].page\_fault == 1)
9. {
10. page\_table[i].page\_fault = 0;
11. replace\_index = i;
12. **break**;
13. }
14. // 找cnt最大的项
15. **if** (page\_table[i].cnt > cnt)
16. {
17. cnt = page\_table[i].cnt;
18. replace\_index = i;
19. }
20. }
21. // 将页面调入页框
22. page\_table[replace\_index].page\_index = page\_index;
23. page\_table[replace\_index].cnt = 0;
24. page\_table[replace\_index].access\_cnt = 0;
25. **int** page\_frame\_index = page\_table[replace\_index].page\_frame\_index;
26. **for** (**int** i = 0; i < PAGE\_SIZE; i++)
27. {
28. page\_frame[page\_frame\_index][i] = process\_mem[page\_index][i];
29. }
30. // 更新所有页面cnt
31. **for** (**int** i = 0; i < PAGE\_FRAME\_SIZE; i++)
32. {
33. page\_table[i].cnt += 1;
34. }
35. **return** page\_frame\_index;
36. }

最近未使用算法LRU根据访问情况，找到访问计数器值最大的项，将该项换出。

1. **int** lru\_algorithm(**int** page\_index)
2. {
3. **int** replace\_index = 0;
4. **int** cnt = -1;
5. **for** (**int** i = 0; i < PAGE\_FRAME\_SIZE; i++)
6. {
7. // 优先查找空项
8. **if** (page\_table[i].page\_fault == 1)
9. {
10. page\_table[i].page\_fault = 0;
11. replace\_index = i;
12. **break**;
13. }
14. // 找access\_cnt最大的项
15. **if** (page\_table[i].access\_cnt > cnt)
16. {
17. cnt = page\_table[i].access\_cnt;
18. replace\_index = i;
19. }
20. }
21. // 将页面调入页框
22. page\_table[replace\_index].page\_index = page\_index;
23. page\_table[replace\_index].cnt = 0;
24. page\_table[replace\_index].access\_cnt = 0;
25. **int** page\_frame\_index = page\_table[replace\_index].page\_frame\_index;
26. **for** (**int** i = 0; i < PAGE\_SIZE; i++)
27. {
28. page\_frame[page\_frame\_index][i] = process\_mem[page\_index][i];
29. }
30. **return** page\_frame\_index;
31. }

模拟程序使用多种访问次序生产器，可生成顺序访问、跳转访问、循环访问和随机访问，模拟实际应用程序内存访问的各种情况。

**3.2 Linux计算物理地址**

Linux下/proc/pid/pagemap文件允许用户查看当前进程虚拟页的物理地址相关信息，每个虚拟页包含一个64位的值，其中第63位为页面是否存在标志，第62位为页面是否被交换标志，第56位为页面是否独占标志，第0-54位为页框号PFN。

对于给定的虚拟地址，需要先计算出虚拟地址对应的页号，再根据pagemap中每一项的大小，找到虚拟地址所在的页，从pagemap中读取对应的64位数据，分析第62位查看页面是否被交换，分析第63位查看页面是否存在，最后根据页框号PFN与页内偏移组合，得到虚拟地址对应的物理地址。

1. // 计算页号
2. file\_offset = virt\_addr / page\_size \* PAGEMAP\_ENTRY;
3. fseek(pagemap\_file, file\_offset, SEEK\_SET);
4. fread(&pagemap\_data, PAGEMAP\_ENTRY, 1, pagemap\_file);
5. fclose(pagemap\_file);
6. printf("pagemap : %lx\n", pagemap\_data);
7. **if** ((pagemap\_data >> 56) & 1)
8. {
9. // 页面独占
10. printf("page exclusively mapped\n");
11. }
12. \_\_uint64\_t pfn = pagemap\_data & 0x7FFFFFFFFFFFFF;
13. printf("PFN : 0x%lx\nphysical addr : 0x%lx\n", pfn, pfn \* page\_size + virt\_addr % page\_size);
14. }

**四、实验结果**

**4.1 页面淘汰算法模拟**

在页框大小为4的情况下，使用跳转访问测试三种页面淘汰算法。

如图4-1 OPT算法缺页率为16.3%

如图4-2 FIFO算法缺页率为19.4%

如图4-3 LRU算法缺页率为17.1%

根据模拟结果可以发现，OPT算法的缺页率最低，但实现较为困难，需要知道未来的地址访问次序，FIFO的实现简单，但缺页率较高，综合分析，LRU的实现难度不大，且缺页率较低。

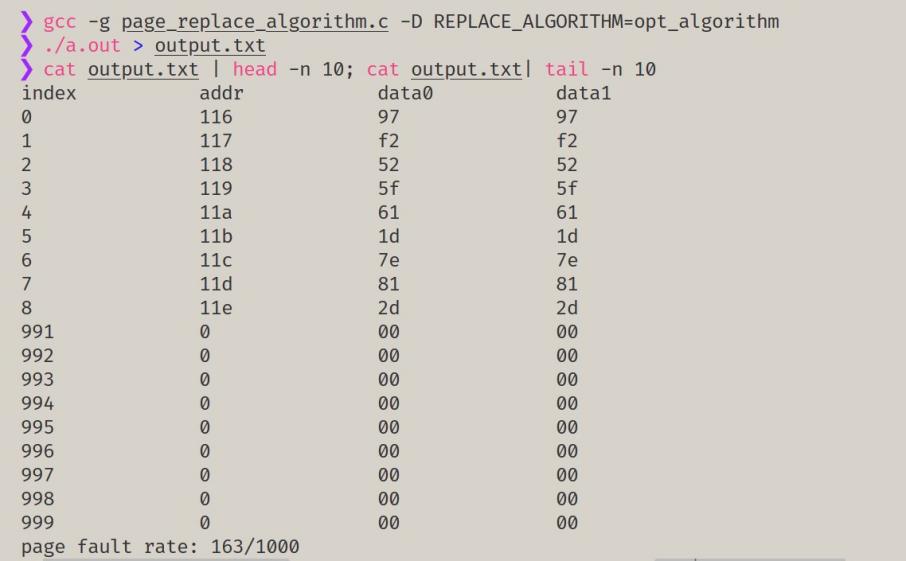


图4-1 OPT算法

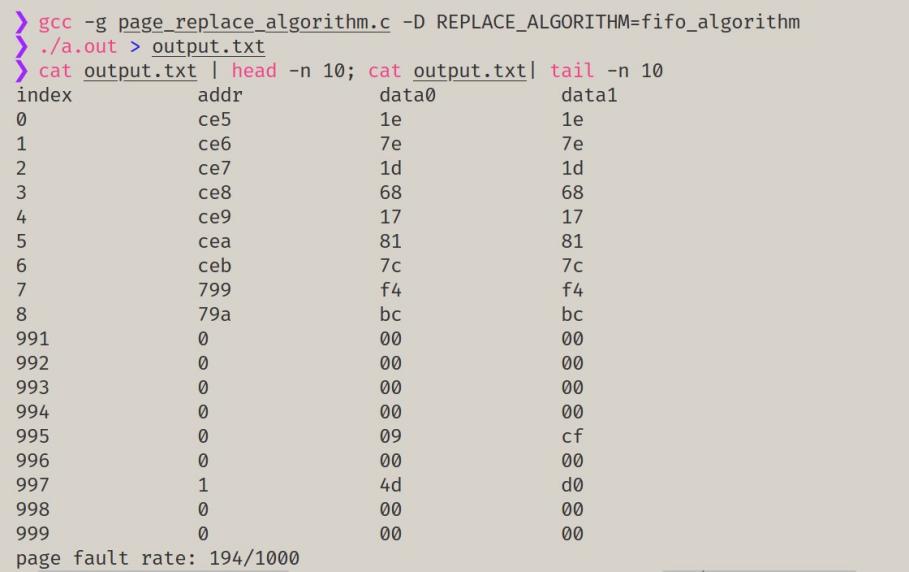


图4-2 FIFO算法

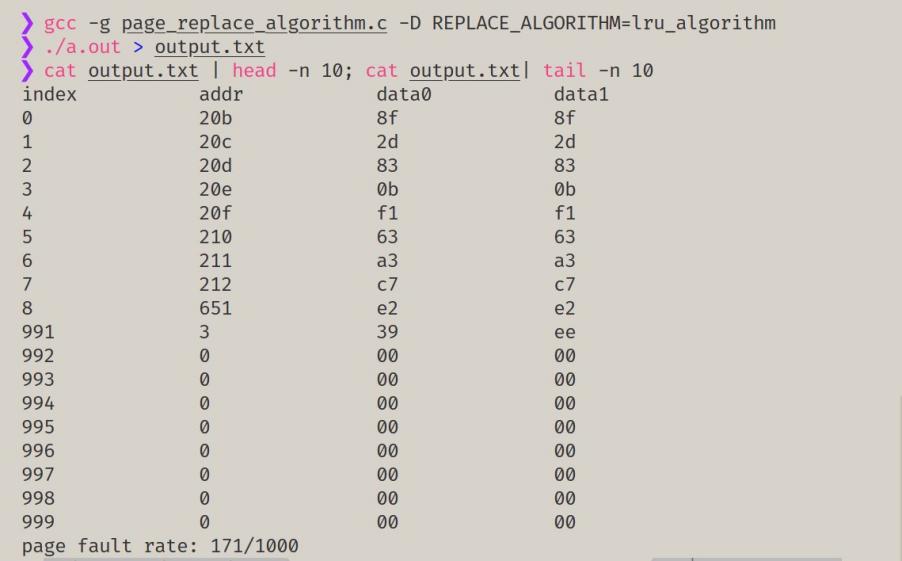


图4-3 LRU算法

**4.2 Linux计算物理地址**

编译并运行虚拟地址与物理地址转换的程序代码，对系统中的两个进程381707和381842进行分析，首先查看两个进程的maps信息，如图4-4所示。

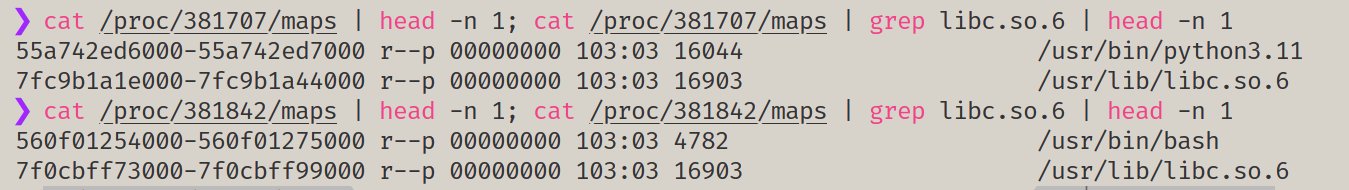


图4-4 进程maps信息

使用pagemap计算进程381707的虚拟地址55a742ed6000对应的物理地址，计算进程381842的虚拟地址560f01254000对应的物理地址，如图4-5所示。



图4-5 进程物理地址计算

如图4-5，进程381707的虚拟地址55a742ed6000对应的物理地址为0x683184000，进程381842的虚拟地址560f01254000对应的物理地址为0x712add000。

使用pagemap计算进程381707和进程381842中共享库文件libc.so.6的物理地址。

如图4-6所示，虽然在进程381707中libc.so.6的虚拟地址为7fc9b1a1e000，进程381842中libc.so.6的虚拟地址为7f0cbff73000，两者的虚拟地址并不相同，但其物理地址都是0x59ad22000，说明不同进程中的共享库的物理地址是相同的，这样可以使多个进程都需要使用的文件只在内存中加载一次，节省内存资源。

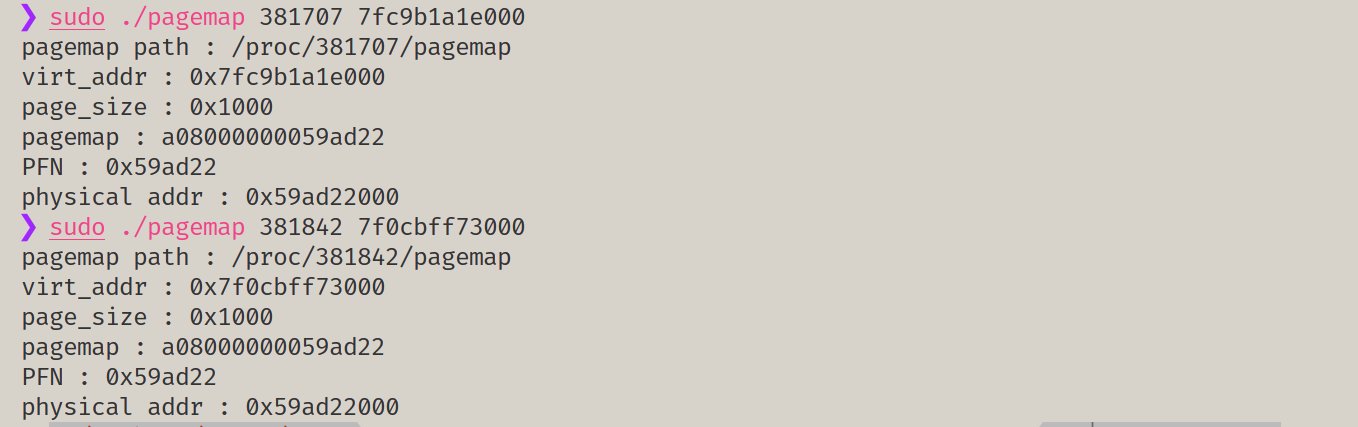


图4-6 libc.so.6物理地址

**五、实验错误排查和解决方法**

**5.1 Linux计算物理地址**

对程序进行测试时，最开始使用的是docker环境，发现多组测试下，都无法得到虚拟地址对应的物理地址，如图5-1所示。

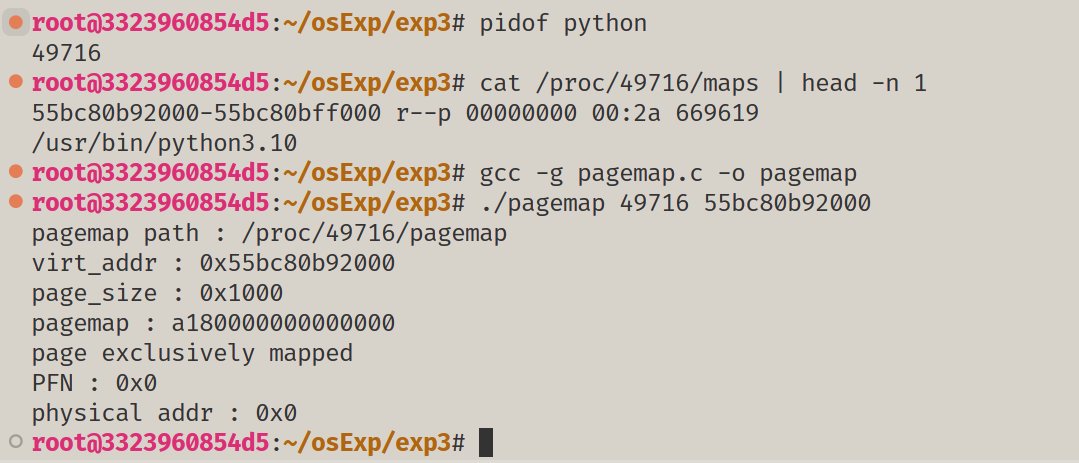


图5-1 docker环境测试

以为程序编写错误，反复调试测试，发现对于任意虚拟地址，其页面存在位、页面交换位以及页面独占位均正常，但是页框号PFN始终为0，查看linux在线文档发现，从pagemap中获取页框号PFN需要进程拥有CAP\_SYS\_ADMIN权限，进一步测试发现，及时使用的是root用户，docker容器仍然不具有CAP\_SYS\_ADMIN权限，在linux主机上重新测试，发现程序功能正常，可以获取页框号PFN，能够将虚拟地址转换为物理地址。

**六、实验参考资料和网址**

**（1）教学课件**

**（2）页面淘汰算法**

**<https://chyyuu.gitbooks.io/simple_os_book/content/zh/chapter-3/swap_algors.html>**

1. **pagemap文件信息**

**<https://www.kernel.org/doc/html/latest/admin-guide/mm/pagemap.html>**