

**课 程 实 验 报 告**

**课程名称： 操作系统原理实验**

**专业班级： 软件2202班**

**学 号： U202217216**

**姓 名： 郑德凯**

**报告日期： 2024/4/23**

**软件学院**

目录

[实验二：第4章进程管理 1](#_Toc164796365)

[一、实验目的 1](#_Toc164796366)

[二、实验内容 2](#_Toc164796367)

[三、实验要求 2](#_Toc164796368)

[四、实验指南 2](#_Toc164796369)

[（1）Windows/Linux模拟实现FIFO或LRU淘汰算法。 2](#_Toc164796370)

[（2）Linux下利用/proc/pid/pagemap计算某个变量或函数虚拟地址对应的物理地址等信息。 2](#_Toc164796371)

[（3）研读并修改Linux内核缺页处理函数do\_no\_page（不同版本该函数名可能有变化）并用printk打印调试信息。注意：需要编译内核。建议优麒麟或麒麟系统。 3](#_Toc164796372)

[五、实验过程 3](#_Toc164796373)

[(1) Windows/Linux模拟实现FIFO或LRU淘汰算法。 3](#_Toc164796374)

[（2）Linux下利用/proc/pid/pagemap计算某个变量或函数虚拟地址对应的物理地址等信息 6](#_Toc164796375)

[（3）研读并修改Linux内核缺页处理函数do\_no\_page（不同版本该函数名可能有变化）并用printk打印调试信息。注意：需要编译内核。建议优麒麟或麒麟系统。 9](#_Toc164796376)

实验二：第4章进程管理

1. 实验目的

* 理解页面淘汰算法原理，编写程序演示页面淘汰算法。
* 验证Linux虚拟地址转化为物理地址的机制
* 理解和验证缺页处理的流程。

1. 实验内容

* Windows/Linux模拟实现FIFO或LRU页面淘汰算法。
* Linux下利用/proc/pid/pagemap计算某个变量或函数虚拟地址对应的物理地址等信息。建议优麒麟或麒麟系统。
* 研读并修改Linux内核的缺页处理函数do\_no\_page 或页框分配函数get\_free\_page，并用printk打印调试信息。注意：需要编译内核。建议优麒麟或麒麟系统。

三、实验要求

* 第1必做，第2或3任选一个必做

四、实验指南

（1）Windows/Linux模拟实现FIFO或LRU淘汰算法。

* 提示1：程序指令执行过程采用遍历数组的操作来模拟；
* 提示2：用1个较大数组A（例如2400个元素）模拟进程，数组里面放的是随机数，每个元素被访问时就使用printf将其打印出来，模仿指令的执行。数组A的大小必须是设定的页大小（例如10个元素或16个元素等等）的整数倍。
* 提示3：用3-8个小数组(例如数组B，数组C，数组D等)模拟分得的页框。小数组的大小等于页大小（例如10条指令的大小，即10的元素）。小数组里面复制的是大数组里面的相应页面的内容（自己另外构建页表，描述大数组的页与小数组序号的关系）。
* 提示4：利用不同的次序访问数组A，次序可以是：顺序，跳转，分支，循环，或随机，自己构建访问次序。不同的次序也一定程度反映程序局部性。
* 提示5：大数组的访问次序可以用 rand( )函数定义，模拟产生指令访问序列，对应到大数组A的访问次序。然后将指令序列变换成相应的页地址流，并针对不同的页面淘汰算法统计“缺页”情况。缺页即对应的“页面”没有装到小数组中（例如数组B，数组C，数组D等）。

（2）Linux下利用/proc/pid/pagemap计算某个变量或函数虚拟地址对应的物理地址等信息。

* 提示1：Linux的/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。该文件记录有每个虚拟页的信息：64位，内含物理地址相关信息
* 提示2：获取当前进程的pagemap文件的全名
* 提示3：可以输出进程中某个或多个全局变量或自定义函数的虚拟地址，所在页号，所在物理页框号，物理地址等信息。
* 思考/扩展：(1)在此实验基础上，编写一个工具程序，展示指定进程中指定虚拟地址的物理地址。

（3）研读并修改Linux内核缺页处理函数do\_no\_page（不同版本该函数名可能有变化）并用printk打印调试信息。注意：需要编译内核。建议优麒麟或麒麟系统。

* 提示1：编写一个名字叫“hello.c”的应用程序，实现双重for循环遍历一个大数组（例如2000x4000，数据处理方法自定，不限）。
* 提示2：在缺页处理函数do\_no\_page( )或类似函数中（不同版本的此函数名有差异），统计”hello”程序的缺页次数，并在它退出时（在do\_exit( )中写代码，参考实验2对do\_fork的调试）打印缺页总次数。
* 【功能扩充】【可选]：在get\_free\_page( )函数中（不同版本的此函数名有差异）采用printk方式添加调试信息，打印hello进程运行过程中分配的页框数量，并统计相关信息在进程结束时（修改并过滤do\_exit( )函数，参考实验二）打印出来。

五、实验过程

(1) Windows/Linux模拟实现FIFO或LRU淘汰算法。

1.代码：

#include <stdio.h>

#include <stdlib.h>

#define PAGE\_SIZE 10

#define NUM\_FRAMES 3

#define ARRAY\_SIZE 2400

int arrayA[ARRAY\_SIZE];

int frames[NUM\_FRAMES][PAGE\_SIZE];

int page\_table[ARRAY\_SIZE / PAGE\_SIZE];

int fifo\_queue[NUM\_FRAMES];

int front = 0, rear = -1, itemCount = 0;

void initializeArrayA() {

for (int i = 0; i < ARRAY\_SIZE; i++) {

arrayA[i] = rand() % 1000;

}

}

void initializePageTable() {

for (int i = 0; i < ARRAY\_SIZE / PAGE\_SIZE; i++) {

page\_table[i] = -1;

}

}

void enqueue(int data) {

if (itemCount < NUM\_FRAMES) {

if (rear == NUM\_FRAMES - 1) {

rear = -1;

}

fifo\_queue[++rear] = data;

itemCount++;

}

}

int dequeue() {

int data = fifo\_queue[front++];

if (front == NUM\_FRAMES) {

front = 0;

}

itemCount--;

return data;

}

void accessPage(int pageNum, int pageIndex) {

int frameIndex = page\_table[pageNum];

if (frameIndex == -1) {

int evictedPageIndex = dequeue();

int evictedPageNum = evictedPageIndex \* PAGE\_SIZE;

frameIndex = pageIndex % NUM\_FRAMES;

page\_table[pageNum] = frameIndex;

for (int i = 0; i < PAGE\_SIZE; i++) {

frames[frameIndex][i] = arrayA[pageNum \* PAGE\_SIZE + i];

}

printf("Page %d loaded into frame %d\n", pageNum, frameIndex);

enqueue(frameIndex);

} else {

printf("Page %d is already in frame %d\n", pageNum, frameIndex);

}

}

int main() {

initializeArrayA();

initializePageTable();

for (int i = 0; i < ARRAY\_SIZE / PAGE\_SIZE; i++) {

int pageNum = rand() % (ARRAY\_SIZE / PAGE\_SIZE);

int pageIndex = rand() % PAGE\_SIZE;

accessPage(pageNum, pageIndex);

}

return 0;

}

2.运行结果：图形用户界面, 文本, 应用程序

描述已自动生成

（2）Linux下利用/proc/pid/pagemap计算某个变量或函数虚拟地址对应的物理地址等信息

1.代码：

#include <stdio.h>

#include <stdint.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/mman.h>

#define PAGE\_SIZE 4096

#define GET\_PFN\_INDEX(virtual\_addr) ((virtual\_addr) / PAGE\_SIZE)

#define GET\_PFN\_OFFSET(virtual\_addr) ((virtual\_addr) % PAGE\_SIZE)

#define PFN\_SIZE sizeof(uint64\_t)

#define GET\_PHYSICAL\_ADDR(virtual\_addr, pfn\_index, pagemap\_fd) ({ \

off\_t offset = pfn\_index \* PFN\_SIZE; \

uint64\_t pfn; \

if (lseek(pagemap\_fd, offset, SEEK\_SET) == -1 || \

read(pagemap\_fd, &pfn, PFN\_SIZE) != PFN\_SIZE) { \

perror("Failed to read pagemap"); \

exit(EXIT\_FAILURE); \

} \

uint64\_t physical\_addr = (pfn & 0x7FFFFFFFFFFFFF) \* PAGE\_SIZE + GET\_PFN\_OFFSET(virtual\_addr); \

physical\_addr; \

})

int main() {

pid\_t pid = getpid();

char pagemap\_path[256];

snprintf(pagemap\_path, sizeof(pagemap\_path), "/proc/%d/pagemap", pid);

int pagemap\_fd = open(pagemap\_path, O\_RDONLY);

if (pagemap\_fd == -1) {

perror("Failed to open pagemap");

exit(EXIT\_FAILURE);

}

void \*variable\_addr = (void \*)main;

uint64\_t pfn\_index = GET\_PFN\_INDEX((uint64\_t)variable\_addr);

uint64\_t physical\_addr = GET\_PHYSICAL\_ADDR((uint64\_t)variable\_addr, pfn\_index, pagemap\_fd);

printf("Virtual Address: %p\n", variable\_addr);

printf("Page Number: %lu\n", pfn\_index);

printf("Physical Address: 0x%lx\n", physical\_addr);

close(pagemap\_fd);

return 0;

}

2.运行结果：

图形用户界面, 文本, 应用程序, Word

描述已自动生成

（3）研读并修改Linux内核缺页处理函数do\_no\_page（不同版本该函数名可能有变化）并用printk打印调试信息。注意：需要编译内核。建议优麒麟或麒麟系统。