西安交通大学

操作系统专题实验报告

班级：_____

学号：_____

姓名：_____

2021 年 12 月 25 日

目　录

一 OpenEuler 系统环境实验

1.1 实验目的

学习多进程模型并分析结果，学习 Linux 操作系统系的 shell 指令

1.2 实验内容

（1） 观察进程调度；观察进程调度中的全局变量改变；在子进程中调用 system 函数；在子进程中调用 exec 族函数

（2） 定义全局变量，在父子进程分别操作，观察结果并解释现象

（3） 父子进程运行不同程序（可使用 exec 函数实现）

（4） 使用 pthread 实现多线程

1.3 实验步骤

（1） 进程调度：

用 fork()函数，若返回值大于 1 则为进入父进程，否则进入子进程

（2） 定义全局变量，在父子进程分别操作：

父进程+1，子进程输出；子进程+1，父进程输出；

（3） pthread：

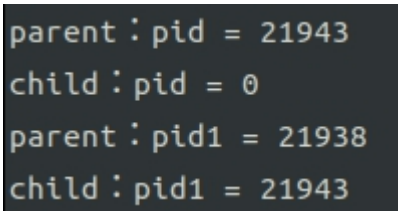
包含 pthread.h 头文件；pthread_create()创建并传入参数；在线程中打印参数

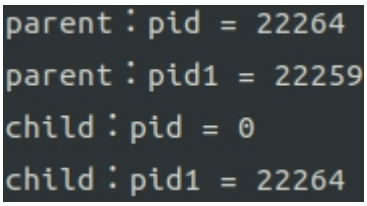
1.4 测试数据设计

不需要设计测试数据。

1.5 程序结果分析

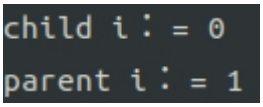
1. 结果：有两种情况





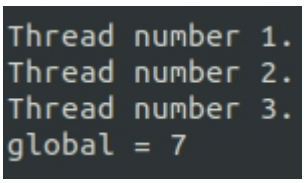
分析：The diffrences between are caused by context switching of the operating system.

2. 结果：子进程和父进程变量互不影响



分析：TWhen executing fork(), the child program will make a copy on both codes and varialbes of the parent. Thus the operation on i is seperatble.

3. 结果：线程中变量共享大小



分析：TThreads shared the global varibale and their operations affect the parent program.

1.6 实验总结

1.6.1 问题以及解决过程

问题：操作系统打开各个软件底层是否是用 fork.

解决：咨询同学得到答案：fork()运用非常广泛。遵循着 Copy-On-Write 机制。

1.6.2 实验收获

熟悉了环境配置.通过编写和运行简单的进程、线程相关程序，体会进程调度、进程间变量管理、进程调用其他程序、线程等方面在操作系统中的实际操作

1.6.3 意见与建议

可以先让同学们熟悉一下 linux 环境。

验收时可以两到三个人同时验收，互相讨论求解。

1.7 附件

1.7.1 源代码

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>


int main(int argc, char ** argv){
    pid_t pid, pid1;
    pid = fork();
    if ( pid < 0){
        printf("Fork failed, exiting...");
        return -1;
    }
    else if (pid == 0) {
        pid1 = getpid();
        printf("child : pid = %d", pid);
        printf("child : pid1 = %d", pid1);
    }
    else {
        pid1 = getpid();
        printf("parent : pid = %d", pid);
        printf("parent : pid1 = %d", pid1);
        wait(NULL);
    }
```

```
24.
25.     return 0;
26. }
```

```
1.  #include <sys/types.h>
2.  #include <stdio.h>
3.  #include <unistd.h>
4.  #include <stdlib.h>
5.
6.  static int i = 0;
7.  int return_;
8.
9.  int main(int argc, char **argv)
10. {
11.     pid_t pid, pid1;
12.     pid = fork();
13.     if (pid < 0)
14.     {
15.         printf("Fork failed, exiting...");
16.         return -1;
17.     }
18.     else if (pid == 0)
19.     {
20.         // return_ = system("echo hello");
21.     }
22.     else
23.     {
24.         pid1 = getpid();
25.         return_ = execl("/bin/echo", "echo", "Hello", NULL, (char *)0);
26.
27.         printf("parent : pid = %d", pid);
28.         printf("parent : pid1 = %d", pid1);
29.         wait(NULL);
30.     }
31.
32.     return 0;
33. }
```

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include <pthread.h>
4.
5.  static g=10;
```

```
6.
7.  void *thread(void *arg)
8.  {
9.      printf("Thread number %d.\n", *(int *)arg);
10.     g--;
11.     return arg;
12. }
13. int main(int argc, char *argv[])
14. {
15.     pthread_t th[3];
16.     int args[3] = {1, 2, 3};
17.     for (int i = 0; i < 3; i++)
18.     {
19.         if (pthread_create(&th, NULL, thread, &args[i]))
20.         {
21.             printf("Error occured!");
22.         }
23.     }
24.
25.     // pthread_exit()
26.
27.     printf("global = %d",g);
28.     return 0;
29. }
```

1.7.2 README

Experiment1

Basic:

Output1:

```
parent: pid = 21943
child: pid = 0
parent: pid1 = 21938
child: pid1 = 21943
```

Output2:

```
parent: pid = 22264
parent: pid1 = 22259
child: pid = 0
child: pid1 = 22264
```

Illustration

The diffrences between are caused by context switching of the operating system.

操作系统专题实验报告

Global varaible:

Situation1:

```
else if (pid == 0)  # child
{
    wait(1);
    printf("child i:  = %d\n", i);

}
else  # parrent
{
    wait(0.5);
    i ++;
    printf("parent i:  = %d\n", i);
    wait(NULL);
}
```

Output1:

```
child i:  = 0
parent i:  = 1
```

Situation2:

```
else if (pid == 0)  # child
{
    wait(0.5);
    i++;
    printf("child i:  = %d\n", i);
}
else  # parrent
{
    wait(1);
    printf("parent i:  = %d\n", i);
    wait(NULL);
}
```

Output1:

```
child i:  = 0
parent i:  = 1
```

Situation2:

```
else if (pid == 0)  # child
{
    wait(0.5);
    i++;
    printf("child i:  = %d\n", i);
}
else  # parrent
{
    wait(1);
    printf("parent i:  = %d\n", i);
    wait(NULL);
}
```

Output2:

```
child i:  = 1
parent i:  = 0
```

Illustration

When executing fork(), the child program will make a copy on both codes and varialbes of the parent. Thus the operation on i is seperatble.

操作系统专题实验报告

Pthread:

```
##### Codes:
static g=10;

void *thread(void *arg)
{
    printf("Thread number %d.\n", *(int *)arg);
    g--;
    return arg;
}
int main(int argc, char *argv[])
{
    pthread_t th[3];
    int args[3] = {1, 2, 3};
    for (int i = 0; i < 3; i++)
    {
        if (pthread_create(&th, NULL, thread, &args[i]))
        {
            printf("Error occured!");
        }
    }
    printf("global = %d\n",g);
    return 0;
}
```

Output:

```
Thread number 1.
Thread number 2.
Thread number 3.
global = 7
```

Illustration

> Threads shared the global varibale and their operations affect the parent program.

二 进程通信与内存管理

1.1 实验目的

本实验在用户态下，根据教材所学习的操作系统原理，完成 Linux 下进程通信与内存管理算法的实现，通过实验，进一步理解所学理论知识。

1.2 实验内容

（1） 软中断；

使用系统调用 fork()创建两个子进程，再用系统调用 signal()让父进程捕捉键盘上发出的中断信号（即按 delete 键），当父进程接收到这两个软中断的某一个后，父进程用系统调用 kill()向两个子进程分别发出整数值为 16 和 17 软中断信号，子进程获得对应软中断信号，然后分别输出下列信息后终止：

Child process 1 is killed by parent !!

Child process 2 is killed by parent !!

父进程调用 wait()函数等待两个子进程终止后，输入以下信息，结束进程执行：

Parent process is killed!!

多运行几次编写的程序，简略分析出现不同结果的原因。

（2） 管道通信：

写程序实现进程的管道通信。用系统调用 pipe( )建立一管道，二个子进程 P1 和 P2 分别向管道各写一句话：

Child P1 is sending a message!

Child P2 is sending a message!

父进程从管道中读出二个来自子进程的信息并显示（要求先接收 P1，后 P2）。

延迟 5 秒后显示

child 1 process is sending message!

再延迟 5 秒

child 2 process is sending message!

并实现循环 5 次读写

（3） 置换算法：

模拟实现 FIFO 算法，LRU 算法。

1.3 实验步骤

（1） 软中断：

用 fork()函数创建子进程，用 signal()函数捕捉信号量，若收到信号则调用回调函数打印。

（2） 管道通信：

使用 pipe()建立管道，用 fork()函数创建进程，进程之间通信使用 write()和 read()来实现管道两侧的读和写。

（3） 置换算法：

FIFO 按照“先进先出”的原理淘汰数据：

新访问的数据插入 FIFO 队列尾部，数据在 FIFO 队列中顺序移动；淘汰 FIFO 队列头部的数据；

LRU 算法根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。

实现是使用一个链表保存缓存数据：

新数据插入到链表头部；每当缓存命中（即缓存数据被访问），则将数据移到链表头部；当链表满的时候，将链表尾部的数据丢弃。

1.4 测试数据设计

在置换算法中，输入如下：

Block num:

3

Page num:

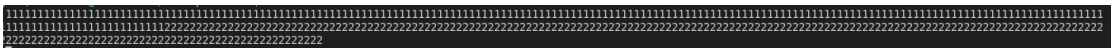
8

Pages to visit:

1 2 3 4 2 4 3 1

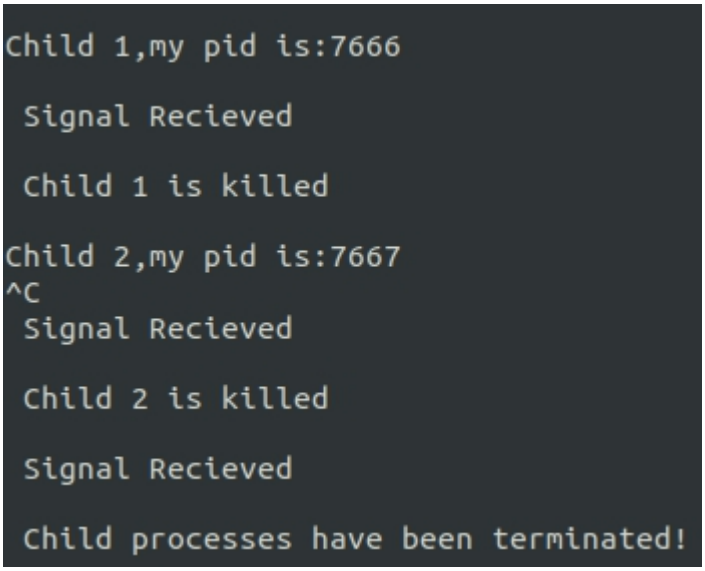
1.5 程序结果分析

1. 结果：管道通信输入输出成功。



分析：当管道通信量设置较长时且不加锁时，管道输出会乱。加锁时管道输出正常。

2. 结果：子进程受到信号，执行回调函数



分析：达到实验预期

3. 结果：

Replacement algorithm：FIFO

```
1
 |1|
2
 |1| |2|
3
 |1| |2| |3|
4
 |4| |2| |3|
2
3
5
 |4| |5| |3|
1
 |4| |5| |1|
Page fault:6   Page fault ratio:6/8
Replacement:3  Hit ratio:0.25
```

Replacement algorithm : LRU

```
1
 |1|
2
 |1| |2|
3
 |1| |2| |3|
4
 |4| |2| |3|
2
3
5
 |5| |2| |3|
1
 |5| |1| |3|
Page fault:6   Page fault ratio:6/8
Replacement:3  Hit ratio:0.25
```

1.6 实验总结

1.6.1 问题以及解决过程

问题：

管道实验中，管道长度定义不足时不影响功能

软中断实验中，用户触发信号(Ctrl+C)和等待系统调用信号输出结果不一样。

解决：询问同学，自己思考得知，管道长度定义为预定义、软中

断若由用户触发效果不同。

1.6.2 实验收获

对进程间通信及管道有了了解，通过本实验，我了解到所谓管道，是指能够连接一个写进程和一个读进程、并允许它们以生产者—消费者方式进行通信的一个共享文件，又称为 pipe 文件。由写进程从管道的写入端（句柄 1）将数据写入管道，而读进程则从管道的读出端（句柄 0）读出数据。学会了使用软中断发送接收信号量。将课本上的置换算法自己实现，对其内容理解更深了。

1.6.3 意见与建议

管道实验中对操作系统内部机制不是很了解；软中断实验中有些信号量不清楚，可以提供一些相关参考资料。

1.7 附件

源代码:

```
#include<unistd.h>
#include<signal.h>
#include<stdlib.h>
#include<sys/wait.h>

int pid1, pid2;
# define LENGTh 20
int main(){
    int fd[2];
    char OutPipe[40], InPipe[400];
    char c1 = '1', c2 = '2';
    pipe(fd);
    while((pid1 = fork()) == -1);
    if(pid1 == 0){
        lockf(fd[1], 1, 0);
        sprintf(OutPipe,"1");
        for(int i = 0; i < 200; i++){
```

```
            write(fd[1], OutPipe, 1);
        }
        sleep(1);
        lockf(fd[1], 0, 0);
        exit(0);

    }

    else{
        while((pid2=fork()) == -1);
        if(pid2 == 0){
            lockf(fd[1], 1, 0);
            sprintf(OutPipe,"2");
            for(int i = 0; i < 200; i++){

                write(fd[1], OutPipe, 1);
            }
            lockf(fd[1], 0, 0);
            sleep(1);
            exit(0);
        }
        else{
            wait(0);
            wait(0);
            read(fd[0], InPipe, 400);
            InPipe[400] = '\0';
            printf(InPipe);
            exit(0);
        }
    }
    return 0;
}
```

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
int wait_flag;

void stop()
{
    printf("\n Signal Recieved\n");
    wait_flag = 0;
}
```

```
12.
13. int main()
14. {
15.     int pid1, pid2;
16.     signal(SIGINT, stop);
17.     while ((pid1 = fork()) == -1);
18.     if (pid1 > 0)
19.     {
20.         while ((pid2 = fork()) == -1);
21.         if (pid2 > 0)
22.         {
23.             wait_flag = 1;
24.             // sleep(10);
25.             kill(pid1, 17);
26.             int kpid1 = wait(0);
27.             int kpid2 = wait(0);
28.             printf("\n Child processes have been terminated! \n");
29.             exit(0);
30.         }
31.         else
32.         {
33.             printf("\nChild 2,my pid is:%d\n", getpid());
34.             wait_flag = 1;
35.             signal(17, stop);
36.             while (1)
37.             {
38.                 if (wait_flag == 0)
39.                 {
40.                     printf("\n Child 2 is killed\n");
41.                     exit(0);
42.                 }
43.             }
44.         }
45.     }
46.     else
47.     {
48.         printf("\nChild 1,my pid is:%d\n", getpid());
49.         wait_flag = 1;
50.         signal(17, stop);
51.         while (1)
52.         {
53.             if (wait_flag == 0)
54.             {
55.                 printf("\n Child 1 is killed\n");
```

```
                exit(0);
            }
        }
    }
}
```

```
#include <stdio.h>

void initializeList(int list[], int number)
{
    for (int i = 0; i < number; i++)
    {
        list[i] = -1;
    }
}

void showList(int list[], int number)
{
    for (int i = 0; i < number; i++)
    {
        printf("%2d", list[i]);
    }
    printf("\n");
}

void showMemoryList(int list[], int phyBlockNum)
{
    for (int i = 0; i < phyBlockNum; i++)
    {
        if (list[i] == -1)
        {
            break;
        }
        printf(" |%d|", list[i]);
    }
    printf("\n");
}

void informationCount(int missingCount, int replaceCount, int pageNum)
{
    printf("Page fault:%d   Page fault ratio:%d/%d\n", missingCount, missingCount, pageNum);
```

```
    double result = (double)(pageNum - missingCount) / (double)pageNum;
    printf("Replacement:%d  Hit ratio:%.2f\n", replaceCount, result);
}

int getNextPosition(int currentPage, int currentPosition, int strList[], int pageNum)
{

    for (int i = currentPosition + 1; i < pageNum; i++)
    {
        if (strList[i] == currentPage)
        {
            return i;
        }
    }

    return -1;
}
void replacePageByFIFO(int memoryList[], int phyNum, int strList[], int pageNum)
{

        int replaceCount = 0;
        int missingCount = 0;

        int pointer = 0;

        int isVisited = 0;
    for (int i = 0; i < pageNum; i++)
    {
         isVisited = 0;

              for (int j = 0; j < phyNum; j++)
        {
            if (memoryList[j] == strList[i])
            {
                                        isVisited = 1;
                                        printf("%d\n", strList[i]);
                break;
            }
            if (memoryList[j] == -1)
            {
                            memoryList[j] = strList[i];
```

```
                        isVisited = 1;
            missingCount++;
                        printf("%d\n", strList[i]);
            showMemoryList(memoryList, phyNum);
            break;
          }
        }

        if (!isVisited)
        {
                          memoryList[pointer] = strList[i];

                pointer++;

                if (pointer > phyNum - 1)
          {
            pointer = 0;
          }

          missingCount++;
          replaceCount++;

                printf("%d\n", strList[i]);
              showMemoryList(memoryList, phyNum);
            }
        }
        informationCount(missingCount, replaceCount, pageNum);
    }

    void replacePageByLRU(int memoryList[], int phyNum, int strList[],
    int pageNum)
    {

          int replaceCount = 0;
          int missingCount = 0;

          int timeRecord[phyNum];
          initializeList(timeRecord, phyNum);

          int isVisited = 0;

          int pageCount = 0;
        for (int i = 0; i < pageNum; i++)
        {
```

```
120.          isVisited = 0;
121.
122.               for (int p = 0; p < pageCount; p++)
123.          {
124.             if (memoryList[p] != -1)
125.             {
126.                timeRecord[p]++;
127.             }
128.          }
129.
130.               for (int j = 0; j < phyNum; j++)
131.          {
132.             if (memoryList[j] == strList[i])
133.             {
134.                                          isVisited = 1;
135.                          timeRecord[j] = -1;
136.                          printf("%d\n", strList[i]);
137.                break;
138.             }
139.             if (memoryList[j] == -1)
140.             {
141.                          memoryList[j] = strList[i];
142.                pageCount++;
143.                          isVisited = 1;
144.                          timeRecord[j]++;
145.
146.                missingCount++;
147.                          printf("%d\n", strList[i]);
148.                showMemoryList(memoryList, phyNum);
149.                break;
150.             }
151.          }
152.          if (!isVisited)
153.          {
154.                             int max = 0;
155.             for (int k = 0; k < phyNum; k++)
156.             {
157.                if (timeRecord[max] < timeRecord[k])
158.                {
159.                   max = k;
160.                }
161.             }
162.
163.                      memoryList[max] = strList[i];
```

```
            timeRecord[max] = -1;

            missingCount++;
            replaceCount++;

                    printf("%d\n", strList[i]);
            showMemoryList(memoryList, phyNum);
        }
    }
    informationCount(missingCount, replaceCount, pageNum);
}

int main(int argc, const char *argv[])
{

    int phyBlockNum;
    printf("Block num:\n");
    scanf("%d", &phyBlockNum);

    int memoryList[phyBlockNum];
    initializeList(memoryList, phyBlockNum);

    int pageNum;
    printf("Page num:\n");
    scanf("%d", &pageNum);

    int pageNumStrList[pageNum];
    printf("Pages to visit?:\n");
    for (int i = 0; i < pageNum; i++)
    {
        scanf("%d", &pageNumStrList[i]);
    }
    printf("\n");
    int chose;
    while (1)
    {
        printf("Choose your algorithm:\n");
        printf("1.FIFO 2.LRU 3.exit\n");
        scanf("%d", &chose);
        switch (chose)
        {
        case 1:
            replacePageByFIFO(memoryList, phyBlockNum, pageNumS
trList, pageNum);
```

```
207.                initializeList(memoryList, phyBlockNum);
208.                printf("\n");
209.
210.                break;
211.            case 2:
212.                 replacePageByLRU(memoryList, phyBlockNum, pageNumS
    trList, pageNum);
213.                initializeList(memoryList, phyBlockNum);
214.                printf("\n");
215.                break;
216.            default:
217.                return 0;
218.                break;
219.            }
220.        }
221.        return 0;
222.    }
```

三 虚拟存储

1.1 实验目的

请求页式虚存管理是常用的虚拟存储管理方案之一。通过请求页式虚存管理中对页面置换算法的模拟,有助于理解虚拟存储技术的特点，并加深对请求页式虚存管理的页面调度算法的理解。

1.2 实验内容

分析研究虚拟存储的思想

模拟虚拟存储（包括数据缓冲、交换文件）

编程模拟一个拥有若干个虚页的进程在给定的若干个实页中运行、并在缺页中断发生时分别使用 FIFO 和 LRU 算法进行页面置换的情形。

要求： 输入虚页的个数、对这些虚页访问的页地址流。要求程序运行时屏幕能显示出置换过程中的状态信息并输出访问结束时的页面命中率。程序应允许通过为该进程分配不同的实页数，来比较两种置换算法的稳定性。

1.3 实验步骤

在之前的置换算法基础上，增加虚拟存储的分页式基本功能。

可以将逻辑地址转换为物理地址。用户可以定义一页的大小和主存大小。

当访问某个页面时，若内存无该页且仍有空间，则直接装入；若

无空间，可以通过 FIFO 或 LRU 做页面置换。

虚拟存储将置换的页从虚存中替换到主存里，再读主存。

1.4 测试数据设计

define PAGE_LENGTH 4

Block num = 3

Page num = 8

Pages to visit:

1 2 3 4 2 3 5 1

1.5 程序结果

Replacement algorithm : FIFO

```
1
 |1|
Primary_memory[0] :b
bbbb
2
 |1| |2|
Primary_memory[4] :c
bbbbcccc
3
 |1| |2| |3|
Primary_memory[8] :d
bbbbccccdddd
4
 |4| |2| |3|
Primary_memory[0] :e
eeeeccccdddd
2
Primary_memory[4] :c
eeeeccccdddd
3
Primary_memory[8] :d
eeeeccccdddd
5
 |4| |5| |3|
Primary_memory[4] :f
eeeeffffdddd
1
 |4| |5| |1|
Primary_memory[8] :b
eeeeffffbbbb
Page fault:6   Page fault ratio:6/8
Replacement:3  Hit ratio:0.25
```

Replacement algorithm : LRU

```
1
 |1|
Primary_memory[0] :b
bbbbffffbbbb
2
 |1| |2|
Primary_memory[4] :c
bbbbccccbbbb
3
 |1| |2| |3|
Primary_memory[8] :d
bbbbccccdddd
4
 |4| |2| |3|
Primary_memory[0] :e
eeeeccccdddd
2
Primary_memory[4] :c
eeeeccccdddd
3
Primary_memory[8] :d
eeeeccccdddd
5
 |5| |2| |3|
Primary_memory[0] :f
ffffccccdddd
1
 |5| |1| |3|
Primary_memory[4] :b
ffffbbbbdddd
Page fault:6   Page fault ratio:6/8
Replacement:3  Hit ratio:0.25
```

1.6 实验总结

1.6.1 问题以及解决过程

问题：

对虚存运行的原理理解不是很透彻，包括分页分段地址构成及转换、快表运行机理

memcpy 函数问题

解决：上网搜索学习。自己尝试解决。

参考资料：

https://blog.csdn.net/shuxnhs/article/details/80956155

1.6.2 实验收获

进一步巩固了自己对于虚存的理解，学到了段页管理等虚拟存储的知识，提升了自己的代码能力。

1.6.3 意见与建议

可以增加一些讲解。

1.7 附件

源代码:

```
#include <stdio.h>
#include <string.h>

#define PAGE_NUM 25
#define PAGE_LENGTH 4

char second_memory[PAGE_NUM * PAGE_LENGTH];

int page_table[PAGE_NUM];

void init()
{
    for (int i = 0; i < PAGE_NUM; i++)
    {
        page_table[i] = -1;
        for (int j = 0; j < PAGE_LENGTH; j++)
        {
            second_memory[i * PAGE_LENGTH + j] = 'a' + i;
        }
    }
}

int Logic2Phy(int pageNum)
{
    int address = page_table[pageNum] * PAGE_LENGTH;
    return address;
}
```

```
28.
29. void read(char primary_memory[], int logic, int phyNum)
30. {
31.     int address = Logic2Phy(logic);
32.      printf("Primary_memory[%d] :%c \n", address, primary_memory[ad
    dress]);
33.     showList(primary_memory, PAGE_LENGTH * phyNum);
34. }
35.
36. void readFromSecondMemory(char primary_memory[], int logic)
37. {
38.     int address = Logic2Phy(logic);
39.      memcpy(&primary_memory[address], &second_memory[logic * PA
    GE_LENGTH], PAGE_LENGTH);
40. }
41.
42. void showList(char list[], int number)
43. {
44.     for (int i = 0; i < number; i++)
45.     {
46.         printf("%c", list[i]);
47.     }
48.     printf("\n");
49. }
50.
51. void initializeList(int list[], int number)
52. {
53.     for (int i = 0; i < number; i++)
54.     {
55.         list[i] = -1;
56.     }
57. }
58.
59. void showMemoryList(int list[], int phyBlockNum)
60. {
61.     for (int i = 0; i < phyBlockNum; i++)
62.     {
63.         if (list[i] == -1)
64.         {
65.             break;
66.         }
67.         printf(" |%d|", list[i]);
68.     }
69.     printf("\n");
```

```
}

void informationCount(int missingCount, int replaceCount, int pageNum)
{
    printf("Page fault:%d   Page fault ratio:%d/%d\n", missingCount, missingCount, pageNum);
    double result = (double)(pageNum - missingCount) / (double)pageNum;
    printf("Replacement:%d  Hit ratio:%.2f\n", replaceCount, result);
}

void replacePageByFIFO(int memoryList[], char primary_memory[], int phyNum, int strList[], int pageNum)
{

    int replaceCount = 0;
    int missingCount = 0;

    int pointer = 0;

    int isVisited = 0;
   for (int i = 0; i < pageNum; i++)
   {
      isVisited = 0;

          for (int j = 0; j < phyNum; j++)
    {
       if (memoryList[j] == strList[i])
       {
                                isVisited = 1;
                                printf("%d\n", strList[i]);
          break;
       }
            if (memoryList[j] == -1)
            {
                        memoryList[j] = strList[i];
              page_table[strList[i]] = j;
               readFromSecondMemory(primary_memory, memoryList[j]);

                        isVisited = 1;
              missingCount++;
                        printf("%d\n", strList[i]);
```

```
                showMemoryList(memoryList, phyNum);
                break;
            }
        }

        if (!isVisited)
        {
                            memoryList[pointer] = strList[i];
            page_table[strList[i]] = pointer;
            readFromSecondMemory(primary_memory, memoryList[po
inter]);

                    pointer++;

                    if (pointer > phyNum - 1)
            {
                pointer = 0;
            }

            missingCount++;
            replaceCount++;

                    printf("%d\n", strList[i]);
            showMemoryList(memoryList, phyNum);
        }
        read(primary_memory, strList[i], phyNum);
      }
      informationCount(missingCount, replaceCount, pageNum);
   }

   void replacePageByLRU(int memoryList[], char primary_memory[],
 int phyNum, int strList[], int pageNum)
   {

        int replaceCount = 0;
        int missingCount = 0;

        int timeRecord[phyNum];
        initializeList(timeRecord, phyNum);

        int isVisited = 0;

        int pageCount = 0;
      for (int i = 0; i < pageNum; i++)
```

```
151.        {
152.            isVisited = 0;
153.
154.                for (int p = 0; p < pageCount; p++)
155.            {
156.                if (memoryList[p] != -1)
157.                {
158.                    timeRecord[p]++;
159.                }
160.            }
161.
162.                for (int j = 0; j < phyNum; j++)
163.            {
164.                if (memoryList[j] == strList[i])
165.                {
166.                                        isVisited = 1;
167.                            timeRecord[j] = -1;
168.                            printf("%d\n", strList[i]);
169.                    break;
170.                }
171.                if (memoryList[j] == -1)
172.                {
173.                            memoryList[j] = strList[i];
174.                    page_table[strList[i]] = j;
175.                    readFromSecondMemory(primary_memory, memoryList[
    j]);
176.
177.                    pageCount++;
178.                            isVisited = 1;
179.                            timeRecord[j]++;
180.
181.                    missingCount++;
182.                            printf("%d\n", strList[i]);
183.                    showMemoryList(memoryList, phyNum);
184.                    break;
185.                }
186.            }
187.
188.            if (!isVisited)
189.            {
190.                                int max = 0;
191.                for (int k = 0; k < phyNum; k++)
192.                {
193.                    if (timeRecord[max] < timeRecord[k])
```

```
194.                {
195.                   max = k;
196.                }
197.             }
198.
199.                    memoryList[max] = strList[i];
200.             page_table[strList[i]] = max;
201.              readFromSecondMemory(primary_memory, memoryList[m
    ax]);
202.
203.             timeRecord[max] = -1;
204.
205.             missingCount++;
206.             replaceCount++;
207.
208.                    printf("%d\n", strList[i]);
209.             showMemoryList(memoryList, phyNum);
210.          }
211.          read(primary_memory, strList[i], phyNum);
212.       }
213.       informationCount(missingCount, replaceCount, pageNum);
214.    }
215.
216.    int main(int argc, const char *argv[])
217.    {
218.
219.       int phyBlockNum;
220.       printf("Block num:\n");
221.       scanf("%d", &phyBlockNum);
222.
223.       int memoryList[phyBlockNum];
224.       initializeList(memoryList, phyBlockNum);
225.
226.       char primary_memory[phyBlockNum * PAGE_LENGTH];
227.       initializeList(primary_memory, phyBlockNum * PAGE_LENGTH);
228.
229.       init();
230.       showMemoryList(page_table, PAGE_NUM);
231.       showList(second_memory, PAGE_NUM * PAGE_LENGTH);
232.
233.       int pageNum;
234.       printf("Page num:\n");
235.       scanf("%d", &pageNum);
236.
```

```
        int pageNumStrList[pageNum];
        printf("Pages to visit?:\n");
        for (int i = 0; i < pageNum; i++)
        {
            scanf("%d", &pageNumStrList[i]);
        }
        printf("\n");
        int chose;
        while (1)
        {
            printf("Choose your algorithm:\n");
            printf("1.FIFO 2.LRU 3.exit\n");
            scanf("%d", &chose);
            switch (chose)
            {
            case 1:
                 replacePageByFIFO(memoryList, primary_memory, phyBlo
    ckNum, pageNumStrList, pageNum);
                initializeList(memoryList, phyBlockNum);
                printf("\n");

                break;
            case 2:
                 replacePageByLRU(memoryList, primary_memory, phyBloc
    kNum, pageNumStrList, pageNum);
                initializeList(memoryList, phyBlockNum);
                printf("\n");
                break;
            default:
                return 0;
                break;
            }
        }

        return 0;
    }
```