山东大学 计算机科学与技术 学院

操作系统 课程实验报告

学号：202200101007	姓名：张祎乾	班级：22.3 班
实验题目：实验 5：扩展 Nachos 的文件系统		
实验学时：2	实验日期：2025/2/23	

实验目的：

通过考察系统加载应用程序过程，如何为其分配内存空间、创建页表并建立 虚页与实页帧的映射关系，理解 Nachos 的内存管理方法；
理解如何系统对空闲帧的管理；
理解如何加载另一个应用程序并为其分配地址空间，以支持多进程机制；
理解进程的 pid；
理解进程退出所要完成的工作；

实验环境：WSL、Ubuntu

任务：

该实验与下一个实验（实验 8）可在目录../lab7-8 中完成，参照实验 2 介绍的方法将该实验中需要修改的模块、头文件，以及依赖这些头文件的模块复制到该目录中。

如将需要的模块从../userprog 目录复制到该目录中，还需要复制 arch 目录及其子目录、Makefile、Makefile.local 等文件，并对 Makefile 及 Makefile.local 做相应的修改。

需要注意的是，code/test/Makefile 第 75 行有 INCDIR=-I../userprog-I../threads，意味着在/code/test 中运行 make，使用了目录 userprog 与 threads 中的头文件。例如汇编 start.s 时使用了 userprog/syscall.h，而不是 lab7-8/syscall.h。

如果要使用 lab7-8 中的头文件，需要将 code/test/Makefile 第 75 行的 INCDIR=-I../userprog-I../threads 修改成 INCDIR=-I../lab7-8-I../threads。

该实验中，需要完成：

（1）阅读../prog/protest.cc，深入理解 Nachos 创建应用程序进程的详细过程

（2）阅读理解类 AddrSpace，然后对其进行修改，使 Nachos 能够支持多进程机制，允许 Nachos 同时运行多个用户线程；

（3）在类 AddrSpace 中添加完善 Print()函数（在实验 6 中已经给出）

（4）在类 AddrSpace 中实例化类 Bitmap 的一个全局对象，用于管理空闲帧；

（5）如果将 SpaceId 直接作为进程号 Pid 是否合适？如果感觉不是很合适，应该如何为进程分配相应的 pid？

（6）为实现 Join(pid)，考虑如何在该进程相关联的核心线程中保存进程号；

（7）根据进程创建时系统为其所做的工作，考虑进程退出时应该做哪些工作；

（8）考虑系统调用 Exec()与 Exit()的设计实现方案；

（9）拓展：可以进一步考虑如何添加自己所需要的系统调用，即../userprog/syscall.h 中没有定

义的系统调用，如 Time，以获取当前的系统时间。

任务处理:

（1）答：protest.cc 是 Nachos 的测试程序，通过调用 StartProcess 函数启动用户程序。
StartProcess 函数会调用 AddrSpace 类加载用户程序到内存，并初始化寄存器状态，最后通过 Fork 创建线程执行用户程序。
重点关注 AddrSpace 类的构造函数和 Fork 函数的实现。
（2）答：内存管理：修改 AddrSpace 类的构造函数，确保每个进程的地址空间独立，避免冲突。
页表管理：为每个进程分配独立的页表，确保物理内存的隔离。
进程调度：确保多个进程的线程能够被调度器公平调度。
同步机制：使用信号量或锁保护共享资源（如空闲帧管理）
（3）答：实验六中已实现
（4）答：我想进行如下修改，但是我怀疑实验八会教我们如何修改，毕竟先前的实验中，指南都给出了实现，所以这里我又改回去了

```
19  #include "bitmap.h"//新增代码 引入BitMap类
20
21  #define UserStackSize   1024  // increase this as necessary!
22
23  class AddrSpace {
24    public:
25      AddrSpace(OpenFile *executable);  // Create an address space,
26                  // initializing it with the program
27                  // stored in the file "executable"
28      ~AddrSpace();     // De-allocate an address space
29
30      void InitRegisters();   // Initialize user-level CPU register
31                  // before jumping to user code
32
33      void SaveState();     // Save/restore address space-specific
34      void RestoreState();    // info on a context switch
35
36      void Print();//新增代码 输出程序的页表
37
38    private:
39      BitMap* bitmap;// 新增代码 实例化类Bitmap的一个全局对象
```

（5）SpaceId 是地址空间的标识符，而 Pid 是进程的唯一标识符。
直接使用 SpaceId 作为 Pid 可能导致冲突，因为多个进程可能共享地址空间。
（6）在 Thread 类中添加 pid 字段，保存进程号。
在 Join 函数中，通过 pid 查找目标线程，并等待其结束。
（7）释放物理内存帧。关闭打开的文件。
从进程表中移除进程。通知父进程（如果存在）。

8.2 背景知识

1、假设我们希望在一个 Nachos 应用程序中通过系统调用 Exec()装入并执 行另一个 Nachos 应用程序../test/exec.noff
2、AddrSpace::AddrSpace 函数内容如下

```
85    // first, set up the translation
86        pageTable = new TranslationEntry[numPages];
87        for (i = 0; i < numPages; i++) {
88        pageTable[i].virtualPage = i;   // for now, virtual page # = phys page #
89        pageTable[i].physicalPage = i;
90        pageTable[i].valid = TRUE;
91        pageTable[i].use = FALSE;
92        pageTable[i].dirty = FALSE;
93        pageTable[i].readOnly = FALSE;  // if the code segment was entirely on
94                                        // a separate page, we could set its
95                                        // pages to be read-only
96        }
```

当加载一个应用程序并为其分配 内存时，总是将程序的第 0 号页面分配到内存的第 0 号帧
程序显然无法正确执行，因为当我们创造第二个程序的时候，就会把第一个进程的数据覆盖

8.3 Bitmap Class

阅读并理解../userprog/bitmap.h 与../userprog/bitmap.cc
通过阅读对比这两个文件，我已清楚 BitMap 类的全部函数用法和作用
学习过程中，我加了大量注释如图所示（未修改代码）

```
34  class BitMap {
35    public:
36      BitMap(int nitems);   // Initialize a bitmap, with "nitems" bits
37      // 初始化一个拥有“nitems”比特的BitMap
38          // initially, all bits are cleared.
39      ~BitMap();      // De-allocate bitmap
40
41      void Mark(int which);     // Set the "nth" bit
42      // 标记which位置的bit位
43      void Clear(int which);    // Clear the "nth" bit
44      // 清除which位置的bit位
45      bool Test(int which);     // Is the "nth" bit set?
46      // 测试which位置的bit位是0还是1
47      int Find();// Return the # of a clear bit, and as a side effect, set the
48      // 找一个为0的bit位，置1并返回位置which，若没有，则返回-1
49      int NumClear();   // Return the number of clear bits
50      // 返回全为0的比特位
51      void Print();   // Print contents of bitmap
52      // 打印全部位示图
53      // These aren't needed until FILESYS, when we will need to read and
54      // write the bitmap to a file
55      void FetchFrom(OpenFile *file);   // fetch contents from disk
56      // 从file文件中得到位示图
57      void WriteBack(OpenFile *file);   // write contents to disk
58      // 将位示图写入file文件
59    private:
60      int numBits;      // number of bits in the bitmap
61      // 比特位的数量
62      int numWords;   // 字的数量
63      // number of words of bitmap storage
64      // (rounded up if numBits is not a
65      //  multiple of the number of bits in
66      //  a word)
67      unsigned int *map;      // bit storage
68      // 位示图
```