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：

理解用户进程是如何通过系统调用与操作系统内核进行交互的；

理解系统调用是如何实现的；

理解系统调用参数传递与返回数据的回传机制；

理解核心进程如何调度执行应用程序进程；

理解进程退出后如何释放内存等为其分配的资源；

理解进程号 pid 的含义与使用；

实验环境：WSL、Ubuntu

先言：我原本是计划一边读指南并在报告里记录重要信息，一边改代码，但是我发现相当乱，实验八需要改动太多了，当我记到 3000 字报告的时候，写下了如下图所示报告内容

我遇到了很多 error，我原本计划，如果遇到 error 并解决完之后，就去修改我前文实验报告，使得别人跟着我的实验报告做的时候可以顺顺利利运行不遇到报错，但是我发现，实验八实在是遇到太多问题了，一堆报错，所以我决定，还是把目前尚未修改的错误全部整理到后边吧，就当是打补丁，而不是修改前文报告。↵

补充：不打补丁了，太乱了太乱了，我把补丁全删了，改过来改过去的，一边跟着指南改一边写实验报告记录，很多地方指南只是说了“要加一个…样的变量”，但是并没有给代码，还有很多函数也是，报告都写 3000 字了，一半都没做完，我决定不记录了，一口气做完，直接给出改完给出最终代码，毕竟报告已经够长够冗余了：↵

最终代码：↵

快速跳转：最终代码

快速跳转：检验代码

任务

（1）阅读../userprog/exception.cc，理解系统调用 Halt()的实现原理；

（2）基于实现 6、7 中所完成的工作，利用 Nachos 提供的文件管理、内存 管理及线程管理等功能，编程实现系统调用 Exec()与 Exit()（至少实现这两个）

观察../userprog/exception.cc 中的函数 ExceptionHandler 如下，发现确实只实现了 halt 系统调用，若是 which 等于其他值，则会报错

```
51  void
52  ExceptionHandler(ExceptionType which)
53  {
54      int type = machine->ReadRegister(2);
55
56      if ((which == SyscallException) && (type == SC_Halt)) {
57      DEBUG('a', "Shutdown, initiated by user program.\n");
58      interrupt->Halt();
59      } else {
60      printf("Unexpected user mode exception %d %d\n", which, type);
61      ASSERT(FALSE);
62      }
63  }
```

待实现的系统调用在 userprog/syscall.h 提示如下

```
21    #define SC_Halt      0
22    #define SC_Exit      1
23    #define SC_Exec      2
24    #define SC_Join      3
25    #define SC_Create    4
26    #define SC_Open      5
27    #define SC_Read      6
28    #define SC_Write     7
29    #define SC_Close     8
30    #define SC_Fork      9
31    #define SC_Yield     10
```

未修改过的情况下，lab7-8 文件夹下有如下文件

```
zhang@zhang:~/OS/nachos-3.4/code/lab7-8$ ls
Makefile  Makefile.local  arch  exception.cc
```

根据实验二复制文件(暂时不知道复制什么)

补充：“最终代码”处有说明

然后修改 Makefile.local

```
20   INCPATH += -I../lab7-8 -I../bin -I../userprog -I../filesys #修改代码，原代码为INCPATH += -I../bin -I../
```

9.2 编写自己的 Nachos 应用程序

我们需要编写一些 Nachos 应用程序，以测试 Nachos 相应的功能。

在 test 下创建 exec.c

C exec.c 1 ×

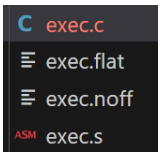
OS > nachos-3.4 > code > test > C exec.c > ...

```c
// 此文件为新建文件，全部为新建代码
#include "syscall.h"
int  main()
{
    SpaceId pid;
    pid=Exec("../test/halt.noff");//利用实现的Exec()执行../test/halt.noff
    Halt();
    /* not reached */
}
```

将 exec 添加到../test/Makefile 文件的 targets 列表中

执行 make 可得到 exec.flat，exec.noff

执行 make exec.s 可生成 exec.s



注：未经修改的 test 中有一个 exec.s，没有用，删除即可

9.3 设计与实现的有关问题

9.3.1 在哪里编写系统调用的代码

1、每一条用户程序中的指令在虚拟机中被读取后，被包装成一个 Instruction 对象

2、当 Nachos 的 CPU 检测到该条指令是执行一个 Nachos 的系统调用，则抛出一个异常 SyscallException 以便从用户态陷入到核心态去处理该系统调用，该异常 SyscallException 在 ../userprog/exception.cc 中进行处理

3、在 exception.cc 中，ExceptionHandler 函数将 SyscallException 接收作为参数 which 处理，语句 type = machine->ReadRegister(2)从寄存器$2 中获取系统调用号,0 号系统调用对应宏 SC_Halt,则执行 Halt 系统调用的处理程序 interrupt->Halt();

4、各系统调用的系统调用号在 userprog/syscall/h 中给出：

```
#define SC_Halt     0
#define SC_Exit     1
#define SC_Exec     2
#define SC_Join     3
#define SC_Create   4
#define SC_Open     5
#define SC_Read     6
#define SC_Write    7
#define SC_Close    8
#define SC_Fork     9
#define SC_Yield    10
```

5、修改 execption.cc，因为要改的内容过多，所以我干脆将原函数注释起来了，再写个新的

并且很明显，这个函数还会接着改，所以我们将其写做第 1 版，后续需要时再重写一个

```
/* 修改代码 新ExceptionHandler第1版*/
void ExceptionHandler(ExceptionType which){
    int type = machine->ReadRegister(2);
    if (which == SyscallException) {
        switch (type){
            case SC_Halt:{
                DEBUG('a', "Shutdown, initiated by user program.\n");
                interrupt->Halt();
                break;
            }
            case SC_Exec:{
                break;
            }
            //其他系统调用程序
            default:{
                printf("Unexpected user mode exception %d %d\n", which, type);
                ASSERT(FALSE);
            }
        }// switch(type)
    } else {
        printf("Unexpected user mode exception %d %d\n", which, type);
        ASSERT(FALSE);
    }
}// ExceptionHandler(ExceptionType which)
```

9.3.2 Nachos 系统调用机制

Exec()的入口汇编代码如下：

```
63  Exec:
64      addiu $2,$0,SC_Exec ;SC_Exec+$0->$2,其中S0=0, SC_Exec:系统调用号, 即$2存放系统调用号
65      syscall             ;执行系统调用
66      j   $31             ;从系统调用中返回到原程序中继续执行,$31存放的是系统调用的返回地址
67      .end Exec
```

9.3.3

1、汇编程序函数调用相关处理

在基于 MIPS 架构中，对于一般的函数调用，一般利用$4-$7（4 到 7 号寄存器）传递函数的前四个参数给子程序，参数多于 4 个时，其余的利用堆栈进行传递；

MIPS 架构将寄存器$2 和$3 存放返回值。

如前面的 exec.c 对应的汇编代码中，在执行系统调用 Exec()之前，利用指令 la $4,$LC0 将 Exec("../test/halt.noff”）中的参数"../test/halt.noff”在内存中的地址$LC0 传给$4，然后执行 Exec，因此内核在处理系统调用 Exec 时，应该首先从$4 中获取"../test/halt.noff”的内存地址，然后将参数从内存中读出。

系统调用 Exit(1)对应的汇编代码为：

li s4,1 #将立即数 0x1 存入寄存器$4；传递 Exit(1)中的参数

jal Exit #转到 start.s 中的 Exit 的调用入口可以看出，对于参数为数值的，系统调用时系统将参数值按顺序依次传入$4-$7 中。

2、Nachos 一些函数

①Machine::ReadRegister(int num)可读取寄存器 num 中的内容

②Machine::ReadMem(int addr, int size, int *value)从内存 addr 处读取 size 个字节的内容存放到 value 所指向的单元中

```
/* 修改代码 新ExceptionHandler第2版*/
void ExceptionHandler(ExceptionType which){
    int type = machine->ReadRegister(2);
    if (which == SyscallException) {
        switch (type){
            case SC_Halt:{
                DEBUG('a', "Shutdown, initiated by user program.\n");
                interrupt->Halt();
                break;
            }
            case SC_Exec:{
                printf("Execute system call of Exec()\n");
                //read argument
                char filename[50];// 用以存储需要执行的函数的名字
                int addr=machine->ReadRegister(4);// 参数地址
                int i=0;
                do{
                    //read filename from mainMemory
                    machine->ReadMem(addr+i,1,(int *)&filename[i]);
                }while(filename[i++]!='\0');
                // 获得了参数，即需要执行的那个文件的名字

                printf("Exec(%s):\n",filename);// 假装执行一下函数，将被注释
                //return spaceID//获得了返回值—执行线程的空间space

                machine->WriteRegister(2,space->getSpaceID());

                break;
            }
            //其他系统调用程序
            default:{
                printf("Unexpected user mode exception %d %d\n", which, type);
                ASSERT(FALSE);
            }
        }// switch(type)
    } else {
        printf("Unexpected user mode exception %d %d\n", which, type);
        ASSERT(FALSE);
    }
}// ExceptionHandler(ExceptionType which)
```

9.3.4 Openfile for the User program

1、Nachos 启动时（主函数在 main.cc 中），通过语句 (void) Initialize(argc, argv)初始化了一个 Nachos 基本内核

2、其中通过 Thread 类（参见 thread.cc）创建了一个 Nachos 的主线程”main”作为当前线程，并将其 状态设为 RUNNING，全局变量

currentThread 指向当前正在执行的线程

3、Nachos 中只有第一个线程即主线程 main 是通过内核直接创建的，其它线程 均需通过调用 Thread::Fork(…)创建

4、当通过命令 nachos -x filename.noff 加载运行 Nachos 应用程序 filename.noff 时，通过../userprog/ progtest.cc 中的函数 StartProcess(char *filename)为该应用程序创建一个用户进程，分配相应的内存，建立用户进程（线程）与核心线程的映射关系，然后启动运行。

4.1、StartProcess(char *filename)如下

```
23  void
24  StartProcess(char *filename)
25  {
26      OpenFile *executable = fileSystem->Open(filename);
27      AddrSpace *space;
28
29      if (executable == NULL) {
30      printf("Unable to open file %s\n", filename);
31      return;
32      }
33      space = new AddrSpace(executable);
34  // 为应用程序filename分配内存空间并将其装入所分配的内存空间中，然后建立页表，并建立虚页与实
35  // space 就是该进程的标识，而不是用pid来当进程标识
36      currentThread->space = space;// 将该进程映射到一个核心进程
37      space->Print();//新增代码 输出该作业的页表信息
38      delete executable;          // close file
39
40      space->InitRegisters();     // set the initial register values
41      // 初始化 CPU 的寄存器
42      space->RestoreState();      // load page table register
43
44      machine->Run();         // jump to the user progam
45      // 从程序入口开始，完成取指令、译码、执行的过程，直到进程遇到Exit()语句或者异常才退出
46      ASSERT(FALSE);          // machine->Run never returns;
47                              // the address space exits
48                              // by doing the syscall "exit"
49  }
```

5、Nachos 自己定义了一种可执行文件格式（.noff）文件，由.coff 转换而来，通 过命令../bin/coff2noff 将 coff 文件转换成 noff 文件。

与常用的 COFF 文件相比， NOFF 文件格式相对简单，便于 Nachos 编程。

6、注 1：进程的 PCB 比较简单，主要包括进程 pid（SpaceID）、页表（代码、 数据、栈）。

7、注 2：由于 Nachos 加载用户程序为其分配的内存空间时，总是将该用户程序分配到从 0 号帧开始的连续区域中，因此目前 Nachos 不支持多进程机制。

7.1、因此在实现系统调用 Exec(filename)时，需要为 filename 所对应的程序创建 一个新的进程，并为其分配另外的内存空间，这一部分是需要改的

注 3：用户线程与核心线程采用 one-to-one 线程映射模型。

注 4：文件../userprog/ progtest.cc 中函数 StartProcess(char *filename)的参数 filename 是一个 Nachos 内核中的对象（string），而系统调用 Exec(filename)中的 参数 filename 是一个用户程序中的对象（string）

9.3.5

采用第二种处理方法：在系统调用处理程序中添加 PC 的推进操作

```
132  //新增代码 PC推荐代码AdvancePC
133  void  AdvancePC() {
134      machine->WriteRegister(PCReg, machine->ReadRegister(PCReg) + 4);
135      machine->WriteRegister(NextPCReg, machine->ReadRegister(NextPCReg) + 4);
136  }
137
138  /* 修改代码 新ExceptionHandler第3版*/
139  void ExceptionHandler(ExceptionType which){
140      int type = machine->ReadRegister(2);
141      if (which == SyscallException) {
142          switch (type){
143              case SC_Halt:{
144                  DEBUG('a', "Shutdown, initiated by user program.\n");
145                  interrupt->Halt();
146                  break;
147              }
148              case SC_Exec:{
149                  printf("Execute system call of Exec()\n");
150                  //read argument
151                  char filename[50];// 用以存储需要执行的函数的名字
152                  int addr=machine->ReadRegister(4);// 参数地址
153                  int i=0;
154                  do{
155                      //read filename from mainMemory
156                      machine->ReadMem(addr+i,1,(int *)&filename[i]);
157                  }while(filename[i++]!='\0');
158                  // 获得了参数，即需要执行的那个文件的名字
159
160                  printf("Exec(%s):\n",filename);// 假装执行一下函数，将被注释
161                  //return spaceID//获得了返回值——执行线程的空间space
162
163                  machine->WriteRegister(2,space->getSpaceID());
164                  AdvancePC();    //PC增量指向下条指令
165                  break;
166              }// case SC_Exec
167              //其他系统调用程序
168              default:{
169                  printf("Unexpected user mode exception %d %d\n", which, type);
170                  ASSERT(FALSE);
171              }
172          }// switch(type)
173      } else {
174          printf("Unexpected user mode exception %d %d\n", which, type);
175          ASSERT(FALSE);
176      }
177  }// ExceptionHandler(ExceptionType which)
```

9.3.6

Exec()返回类型为 SpaceId 的值(就是 pid)，该值将来作为系统调用 Join()的参数以识别新建的用户进程。

（1）它是如何产生的

从代码../userprog/protest.cc 中的 StartProcess()可以看出，加载运行一个 应用程序的过程就是首先打开这个程序文件，为该程序分配一个新的内存空间，并将该程序装入到该空间中，然后为该进程映射到一个核心线程，根据 文件的头部信息设置相应的寄存器运行该程序。

这里进程地址空间的首地址是唯一的,理论上可以利用该值识别该进程， 但该值不连续，且值过大。

我们可以为一个地址空间或该地址空间对应的进程分配一个唯一的整数，例如 0~99 预留为核心进程使用（目前没有核心进程 的概念，核心中只有线程），用户进程号从 100 开始使用。

一句话概括：目前 nachos 使用空间首地址标识进程，但该地址值过大且不连续，我们将为其分配唯一整数

（2）在内核中如何记录他

Thread:: Fork(VoidFunctionPtr func, _int arg)有两个参数，一个是线程要运行的代码，另一个是一个整数，可以考虑将用户进程映射到核心线程时，利用 Fork()的第二个参数将进程的 pid 传入到核心中。

9.3.7Join

int Join(SpaceId id)的功能是调用 Join(SpaceId id)的程序等待进程 id 结束，当进程 id 结束后，Join()返回进程 id 的退出状态 （退

出码)。

当子线程执行结束后，父线程执行 Join()时应该直接返回，不需等待；

由于目前 Nachos 没有实现线程家族树的概念，给 Join()的实现带来一些困难。前面已经提到，AddrSpace 类代替了 PCB 的功能，因此可以在 AddrSpace 中设法建立进程之间的家族关系；

1、在 Thread 类中实现

（1）在 thread.cc 中添加函数 Join(int spaceId)，系统调用 int Join(int spaceId) 通过调用 Thread::Join()实现

（2）Thread::Join()将当前线程睡眠

（a）在 Scheduler::Scheduler()中初始化一个线程等待队列及线程终止队列；

由于要修改 Scheduler.cc 代码，所以我们将 Scheduler.cc 复制到 lab7-8 文件夹下：

```
zhang@zhang:~/OS/nachos-3.4/code$ cp threads/scheduler.cc lab7-8/
```

修改 lab7-8/scheduler.cc：

```
30  Scheduler::Scheduler()
31  {
32      readyList = new List; //就绪队列 thread ready queue
33      // 新增代码5行
34      #ifdef USER_PROGRAM
35      waitingList = new List;     //等待队列 如果Joinee 没有退出，Joiner进入等待
36      terminatedList =new List;  //终止队列 线程调用 Finish()后进入该状态
37      //Joiner 通过检查该队列确定Joinee是否已经退出
38      #endif
39  }
```

（b）

为线程增加一个 TERMINATED 状态，相应地增加一个 terminated

队列，将所有的线程在调用 Finish()后先进入该队列，再伺机销毁

① 当 Joiner 执行 Thread::Join(spaceId)时，若 Joinee 在 terminated 队列，则 从 terminated 队列移除 Joinee 并将其销毁，然后返回 Joinee 的退出码；

② 如果 Joinee 不在 terminated 队列,说明其尚未终止,则 Joiner 进入睡眠 队列 waitingList，当 Joinee 退出调用 Finish()时通过检查 waitingList 以确定 是否需要唤醒 Joiner

注 2：需要在 Thread 中增加一个成员变量，保存与之相关联的进程 spaceId（pid）

注 1：系统调用 Exit(exitcode)应将进程的返回码传递给与其相关联的核心线程

修改 lab7-8/exception.cc：

```
/* 修改代码 新ExceptionHandler第4版，增加case SC_Join*/
void ExceptionHandler(ExceptionType which){
    int type = machine->ReadRegister(2);
    if (which == SyscallException) {
        switch (type){
            case SC_Halt:{
                DEBUG('a', "Shutdown, initiated by user program.\n");
                interrupt->Halt();
                break;
            }
            case SC_Exec:{
                printf("Execute system call of Exec()\n");
                //read argument
                char filename[50];// 用以存储需要执行的函数的名字
                int addr=machine->ReadRegister(4);// 参数地址
                int i=0;
                do{
                    //read filename from mainMemory
                    machine->ReadMem(addr+i,1,(int *)&filename[i]);
                }while(filename[i++]!='\0');
                // 获得了参数，即需要执行的那个文件的名字

                printf("Exec(%s):\n",filename);// 假装执行一下函数，将被注释
                //return spaceID//获得了返回值—执行线程的空间space

                machine->WriteRegister(2,space->getSpaceID());
                AdvancePC();    //PC增量指向下条指令
                break;
            }// case SC_Exec
            case SC_Join:{
                int SpaceId=machine->ReadRegister(4);  //ie. ThreadId or SpaceId
                currentThread->Join(spaceId);
                //返回 Joinee 的退出码waitProcessExitCode
                machine->WriteRegister(2, currentThread->waitProcessExitCode);
                AdvancePC();
                break;
            }
            //其他系统调用程序
            default:{
                printf("Unexpected user mode exception %d %d\n", which, type);
                ASSERT(FALSE);
            }
        }// switch(type)
    } else {
        printf("Unexpected user mode exception %d %d\n", which, type);
        ASSERT(FALSE);
    }
}// ExceptionHandler(ExceptionType which)*/
```

现在我们需要修改 thread.h，但是我使用 grep 发现，设计 thread.h 的文件很多，也就是说，如果我想将 thread.h 复制到 lab7-8 文件夹下再修改，需要修改的文件就很多了，并且这也是最后一个实验（没有后续

实验会被影响），遂决定：直接在 threads 文件夹下修改

修改 threads/thread.h

```
104     void Print() { printf("%s, ", name); }
105
106     void Join(int SpaceId); // 新增代码 增加Join函数
107     void Terminated();      // 新增代码 增加Terminated函数
108
109   private:
110     // some of the private data for this class is listed above
```

修改 threads/thread.cc

```
399 // 新增代码39行 实现Thread::Join函数
400 #ifdef USER_PROGRAM
401 void Thread::Join(int SpaceId)  { //int join(SpaceId)
402     IntStatus oldLevel = interrupt->SetLevel(IntOff);
403     currentThread->waitingProcessSpaceId = SpaceId;
404     //if joinee is still in not in terminated list?
405     Thread *thread;
406     List *terminatedList = scheduler->getTerminatedList();
407     List *waitingList = scheduler->getWaitingList();
408     bool interminatedList = FALSE;
409     int listLength = terminatedList->ListLength();  //length of ready queue
410     for (int i = 1; i <= listLength; i++)
411     {
412         thread = (Thread *)terminatedList->getItem(i);
413         if (thread == NULL)
414             interminatedList = FALSE;    // joinee not finished
415         //joinee is still in Ready queue, not finished
416         if (thread->UserProgramId == SpaceId)
417         {
418             interminatedList = TRUE;  // joinee alreday finished
419             break;
420         }
421         interminatedList = FALSE;    // joinee not finished
422     }
423     //Joinee is not finished, still in not in
424     // terminated List(not at TERMINALTE),
425     if (!interminatedList)
426     // still in Nanchos system, maybe at READY or BLOCKED
427     {
428         waitingProcessSpaceId = SpaceId;
429         waitingList->Append((void *)this);    //blocked Joiner
430         currentThread->Sleep();
431     }
432     //joinee alreday finished, in terminated List, empty it, and return
433     currentThread->waitProcessExitCode = waitingThreadExitCode;
434     //delete terminated thread "Joinee"
435     scheduler->deleteTerminatedThread(SpaceId);
436     interrupt->SetLevel(oldLevel);
437     }
438 #endif
```

```
147 // 新增代码32行 实现Thread::Finish函数
148 void Thread::Finish ()
149 {
150     (void) interrupt->SetLevel(IntOff);
151     ASSERT(this == currentThread);
152 #ifdef USER_PROGRAM
153     waitingThreadExitCode = currentThread->getExitStatus();
154     //joinee finised, wakeup the join user program
155     List *ReadyList = scheduler->getReadyList();
156     List *waitingList = scheduler->getWaitingList();
157     Thread *waitingThread;
158 
159     // if joiner is sleeping and in waitinglist, joinee wait up joiner
160     // when joinee finish
161     int listLength = waitingList->ListLength();
162     for (int i = 1; i <= listLength; i++)
163     {
164         waitingThread = (Thread *)waitingList->getItem(i);
165         if (currentThread->UserProgramId == waitingThread->waitingProcessSpaceId)
166         {
167             scheduler->ReadyToRun((Thread *)waitingThread);
168             waitingList->RemoveItem(i);
169             break;
170         }
171     }
172     Terminated();
173 #else
174    threadToBeDestroyed = currentThread;
175    Sleep();     // invokes SWITCH
176    // not reached
177    #endif
178 }
```

```
474  // 新增代码21行 实现Thread::Terminated函数
475  #ifdef USER_PROGRAM
476  void Thread::Terminated()
477  {
478      List *terminatedList = scheduler->getTerminatedList();
479
480      Thread *nextThread;
481
482      ASSERT(this == currentThread);  // a thread sleep by itsef
483      ASSERT(interrupt->getLevel() == IntOff);
484      status = TERNINATED;
485      terminatedList->Append((void *)this);
486
487      nextThread = scheduler->FindNextToRun();
488      while(nextThread == NULL)
489      {
490          interrupt->Idle();
491          nextThread = scheduler->FindNextToRun();
492      }
493      scheduler->Run(nextThread); // returns when we've been signalled
494  }
495  #endif
```

9.3.8 Exec()

系统调用 pid=Exec(filename)的功能是加载运行应用程序 filename

1、设计思路：

（1）从 2 号寄存器中获取当前的系统调用号（type=machine->ReadRegister(2)），根据 type 对系统调用分别处理

（2）获取系统调用参数（寄存器 4、5、6、7，可以携带 4 个参数）

（a）从第 4 号寄存器中获取 Exec()的参数 filename

（b）利用 Machine::ReadMem()从该地址读取应用程序文件名 filename

（c）打开该应用程序（OpenFile *executable = fileSystem->Open(filename)）

(d) 为其分配内存空间、创建页表、分配 pid（space = new AddrSpace(executable)），至此为应用程序创建了一个进程

需要修改 AddrSpace:: AddrSpace()与 AddrSpace:: ~AddrSpace()

我们可以重载函数 StartProcess(int spaceId)，作为新建线程执行的代码

2、修改 progtest.cc，重载函数 StatProcess(char *filename)

```
23  // 新增代码9行 重载StartProcess(char *filename)
24  void StartProcess(int spaceId)
25  {
26      space->InitRegisters(); // set the initial register values
27      space->RestoreState();  // load page table register
28
29      machine->Run();    // jump to the user progam
30      ASSERT(FALSE);  // machine->Run never returns;
31                      // the address space exits by doing the syscall "exit"
32  }
```

3、修改 AddrSpace::AddrSpace()及 AddrSpace::~AddrSpace()

修改 userprog/addrspace.h：

```
#ifndef ADDRSPACE_H
#define ADDRSPACE_H

#include "copyright.h"
#include "filesys.h"

#include"syscall.h" // 新增代码 包含了SpaceId变量的定义

#define UserStackSize   1024  // increase this as necessary!

class AddrSpace {
  public:
    AddrSpace(OpenFile *executable);  // Create an address space,
        // initializing it with the program
        // stored in the file "executable"
    ~AddrSpace();     // De-allocate an address space

    void InitRegisters();   // Initialize user-level CPU registers,
        // before jumping to user code

    void SaveState();     // Save/restore address space-specific
    void RestoreState();    // info on a context switch

    void Print();//新增代码 输出程序的页表

  private:
    TranslationEntry *pageTable;  // Assume linear page table translation
        // for now!
    unsigned int numPages;    // Number of pages in the virtual
        // address space
    SpaceId spaceID;// 新增代码 声明pid
};

#endif // ADDRSPACE_H
```

修改 userprog/addrspace.cc：

```
120  // 新增代码60行 新AddrSpace
121  #define MAX_USERPROCESSES 256       //最大进程数
122  BitMap *bitmap; //for free frame
123  bool ThreadMap[MAX_USERPROCESSES];  //pid or SpaceId
124  AddrSpace::AddrSpace(OpenFile *executable)
125  {
126      NoffHeader noffH;
127      unsigned int i, size;
128
129      executable->ReadAt((char *)&noffH, sizeof(noffH), 0);// 获取noffh文件头
130      // 下边这个文件应该是确认该文件为noff的，不用改
131      if ((noffH.noffMagic != NOFFMAGIC) &&
132          (WordToHost(noffH.noffMagic) == NOFFMAGIC))
133          SwapHeader(&noffH);
134      ASSERT(noffH.noffMagic == NOFFMAGIC);
135
136  // how big is address space?计算地址空间大小
137      size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
138              + UserStackSize;    // we need to increase the size
139                      // to leave room for the stack
140      numPages = divRoundUp(size, PageSize);
141      size = numPages * PageSize;
142
143      ASSERT(numPages <= NumPhysPages);       // check we're not trying
144                      // to run anything too big --
145                      // at least until we have
146                      // virtual memory
147
148      bool hasAvailabePid = false;
149      // 寻找一个防止头文件的页
150      for(int i = 100; i < MAX_USERPROCESSES; i++) {
151          if(!ThreadMap[i]){
152              ThreadMap[i] = true;
153              spaceID = i;  //may be should reserved 0-99 for kernel Process,even though there is no any process at present
154            hasAvailabePid = true;  //ther is available Pid for new process
155            break;
156          }
157      }// for
158      if (!hasAvailabePid)   //no available Pid for new process
159      {
160          printf("Too many processes in Nachos!\n");
161          return;
162      }
163
164      if(bitmap == NULL)   //used for free frames
165          bitmap = new BitMap(NumPhysPages);
166      //the remaining code
167      //set up a new PageTable for a process, first, set up the translation
168      DEBUG('a', "Initializing address space, num pages %d, size %d\n",numPages, size);
169      pageTable = new TranslationEntry[numPages];
170      for (int i = 0; i < numPages; i++) {
171          pageTable[i].virtualPage = i;   // virtual page #
172          pageTable[i].physicalPage = bitmap->Find();  // find a free frame
173          ASSERT(pageTable[i].physicalPage!=-1);
174          pageTable[i].valid = TRUE;
175          pageTable[i].use = FALSE;
176          pageTable[i].dirty = FALSE;
177          pageTable[i].readOnly = FALSE;  // if the code segment was entirely on
178                                          // a separate page, we could set its
179                                          // pages to be read-only
180      }
181
182  // zero out the entire address space, to zero the unitialized data segment
183  // and the stack segment
184      bzero(machine->mainMemory, size);
185
186  // then, copy in the code and data segments into memory
187      if (noffH.code.size > 0) {
188          DEBUG('a', "Initializing code segment, at 0x%x, size %d\n",
189              noffH.code.virtualAddr, noffH.code.size);
190          executable->ReadAt(&(machine->mainMemory[noffH.code.virtualAddr]),
191              noffH.code.size, noffH.code.inFileAddr);
192      }
193      if (noffH.initData.size > 0) {
194          DEBUG('a', "Initializing data segment, at 0x%x, size %d\n",
195              noffH.initData.virtualAddr, noffH.initData.size);
196          executable->ReadAt(&(machine->mainMemory[noffH.initData.virtualAddr]),
197              noffH.initData.size, noffH.initData.inFileAddr);
198      }
199
200  }
201
```

```
208 AddrSpace::~AddrSpace()
209 {
210     // 新增代码4行
211     ThreadMap[spaceID] = 0; //false
212     for (int i = 0; i < numPages; i++) {
213       bitmap->Clear(pageTable[i].physicalPage);
214     }
215    delete [] pageTable;
216 }
```

修改 lab7-8/exception.cc：

```
/* 修改代码 新ExceptionHandler第5版，实现SC_Exec*/
void ExceptionHandler(ExceptionType which){
    int type = machine->ReadRegister(2);
    if (which == SyscallException) {
        switch (type){
            case SC_Halt:{
                DEBUG('a', "Shutdown, initiated by user program.\n");
                interrupt->Halt();
                break;
            }
            case SC_Exec:{
                printf("Execute system call of Exec()\n");
                //read argument
                char filename[128];// 用以存储需要执行的函数的名字
                int addr=machine->ReadRegister(4);// 参数地址
                int i=0;
                do{
                    //read filename from mainMemory
                    machine->ReadMem(addr+i,1,(int *)&filename[i]);
                }while(filename[i++]!='\0');
                // 获得了参数，即需要执行的那个文件的名字

                //printf("Exec(%s):\n",filename);// 假装执行一下函数，将被注释
                OpenFile *executable = fileSystem->Open(filename);
                if (executable == NULL) {
                    printf("Unable to open file %s\n", filename);
                    return;
                }
                //new address space
                space = new AddrSpace(executable);
                delete executable;   // close file
                //new and fork thread
                char *forkedThreadName=filename;
                Thread* thread = new Thread(forkedThreadName);
                thread->Fork(StartProcess, space->getSpaceID());
                thread->space = space;  //用户线程映射到核心线程
                //return spaceID//获得了返回值—执行线程的空间space

                machine->WriteRegister(2,space->getSpaceID());
                AdvancePC();    //PC增量指向下条指令
                break;
            }// case SC_Exec
            case SC_Join:{
                int SpaceId=machine->ReadRegister(4);  //ie. ThreadId or SpaceId
                currentThread->Join(spaceId);
                //返回 Joinee 的退出码waitProcessExitCode
                machine->WriteRegister(2, currentThread->waitProcessExitCode);
                AdvancePC();
                break;
            }
            //其他系统调用程序
            default:{
                printf("Unexpected user mode exception %d %d\n", which, type);
                ASSERT(FALSE);
            }
        }// switch(type)
    } else {
        printf("Unexpected user mode exception %d %d\n", which, type);
        ASSERT(FALSE);
    }
}// ExceptionHandler(ExceptionType which)*/
```

我遇到了很多 error，我原本计划，如果遇到 error 并解决完之后，就去修改我前文实验报告，使得别人跟着我的实验报告做的时候可以顺顺利利运行不遇到报错，但是我发现，实验八实在是遇到太多问题了，一堆报错，所以我决定，还是把目前尚未修改的错误全部整理到后边吧，就当是打补丁，而不是修改前文报告。

补充：不打补丁了，太乱了太乱了，我把补丁全删了，改过来改过去的，一边跟着指南改一边写实验报告记录，很多地方指南只是说了“要加一个…样的变量”，但是并没有给代码，还有很多函数也是，报告都写 3000 字了，一半都没做完，我决定不记录了，一口气做完，直接给出改完给出最终代码，毕竟报告已经够长够冗余了：

最终代码：

需要修改的文件:
threads/scheduler.h
threads/scheduler.cc
threads/thread.h
threads/thread.cc
userprog/addrspace.h
userprog/addrspace.cc
userprog/progtest.h(新建)
userprog/progtest.cc
lab7-8/exception.cc
threads/list.h
threads/list.cc
threads/system.h
threads/system.cc
不需要修改但需要包含的文件:
userprog/syscall.h
userprog/bitmap.h
userprog/bitmap.cc
threads/main.cc

修改 threads/scheduler.h

```
#include"syscall.h" // 新增代码 包含SpaceId变量(其实就是int)

// The following class defines the scheduler/dispatcher abstraction --
// the data structures and operations needed to keep track of which
// thread is running, and which threads are ready but not running.

class Scheduler {
  public:
    Scheduler();      // Initialize list of ready threads
    ~Scheduler();     // De-allocate ready list

    void ReadyToRun(Thread* thread);  // Thread can be dispatched.
    Thread* FindNextToRun();    // Dequeue first thread on the ready
        // list, if any, and return thread.
    void Run(Thread* nextThread); // Cause nextThread to start running
    void Print();     // Print contents of ready list

    List* getWaitingList();    // 新增代码 返回等待进程队列
    List* getReadyList();      // 新增代码 返回就绪进程队列
    List* getTerminatedList(); // 新增代码 返回终止进程队列
    void deleteTerminatedThread(SpaceId spaceID);     // 新增代码 删除停止运行的进程
    void emptyList(List *list);                       // 新增代码 清空进程队列

  private:
    List *readyList;      // queue of threads that are ready to run,
      // but not running
    List *waitingList;      // 新增代码 等待队列
    List *terminatedList;   // 新增代码 终止队列
};

#endif // SCHEDULER_H
```

修改 threads/scheduler.cc

```
Scheduler::Scheduler()
{
    readyList = new List;
    terminatedList = new List;  // 新增代码 初始化终止队列
    waitingList = new List;     // 新增代码 初始化等待队列
}
```

```
152  // 新增代码3行 实现Scheduler::getTerminatedList函数
153  List *Scheduler::getTerminatedList(){
154      return terminatedList;
155  }
156
157  // 新增代码3行 实现Scheduler::getWaitingList函数
158  List *Scheduler::getWaitingList(){
159      return waitingList;
160  }
161
162  // 新增代码3行 实现Scheduler::getReadyList函数
163  List *Scheduler::getReadyList(){
164      return readyList;
165  }
166
167  // 新增代码10行 实现Scheduler::deleteTerminatedThread函数
168  void Scheduler::deleteTerminatedThread(int SpaceId){
169      int length = terminatedList->ListLength();
170      for(int i = 0; i < length; i++){
171          Thread *thread = (Thread *)terminatedList->GetItem(i);
172          if(thread->UserProgramId == SpaceId){
173              terminatedList->Remove(thread);
174              break;
175          }
176      }
177  }
178
179  // 新增代码5行 实现Scheduler::emptyList函数
180  void Scheduler::emptyList(List *list){
181      int length = list->ListLength();
182      for(int i = 0;i < length;i++)
183          list->Remove();
184  }
185
```

修改 threads/thread.h，全部修改如下，把前面的修改删掉吧

```
43  #ifdef USER_PROGRAM
44  #include "machine.h"
45  #include "addrspace.h"
46  #include "list.h" // 新增代码 为了包含List类
47  #endif
```

```
114      char name[50];// 修改代码 原为char* name;
```

```
120 #ifdef USER_PROGRAM
121 // A thread running a user program actually has *two* sets of CPU registers -
122 // one for its state while executing user code, one for its state
123 // while executing kernel code.
124
125     int userRegisters[NumTotalRegs];  // user-level CPU register state
126
127
128
129   public:
130     void SaveUserState();   // save user-level register state
131     void RestoreUserState();    // restore user-level register state
132
133     void Join(int SpaceId); // 新增代码 增加Join函数
134     void Terminated();      // 新增代码 增加Terminated函数
135     void setExitCode(int Code);    // 新增代码 设置进程的退出码
136     int getExitStatus();           // 新增代码 返回进程的退出码
137     SpaceId waitingProcessSpaceId;// 新增代码 等待进程的SpaceId
138     SpaceId UserProgramId;        // 新增代码 用户进程id
139     int waitProcessExitCode;      // 新增代码 等待进程的退出码
140     int exitCode;                 // 新增代码 进程的退出码
141
142     AddrSpace *space;     // User code this thread is running.
143 #endif
144 };
145
```

修改 threads/thread.cc：

```
22  #include "list.h"// 新增代码 为了包含List类
```

```
36  Thread::Thread(char* threadName)
37  {
38      strcpy(name,threadName);// 修改代码 原为name = threadName;
39      stackTop = NULL;
40      stack = NULL;
41      status = JUST_CREATED;
42  #ifdef USER_PROGRAM
43      space = NULL;
44  #endif
45  }
```

```
162  // 修改代码33行 新Thread::Finish函数
163  int waitingThreadExitCode;
164  void Thread::Finish ()
165  {
166      (void) interrupt->SetLevel(IntOff);
167      ASSERT(this == currentThread);
168  #ifdef USER_PROGRAM
169      waitingThreadExitCode = currentThread->getExitStatus();
170      //joinee finised, wakeup the join user program
171      List *ReadyList = scheduler->getReadyList();
172      List *waitingList = scheduler->getWaitingList();
173      Thread *waitingThread;
174
175      // if joiner is sleeping and in waitinglist, joinee wait up joiner
176      // when joinee finish
177      int listLength = waitingList->ListLength();
178      for (int i = 1; i <= listLength; i++)
179      {
180          waitingThread = (Thread *)waitingList->GetItem(i);
181          if (currentThread->UserProgramId == waitingThread->waitingProcessSpaceId)
182          {
183              scheduler->ReadyToRun((Thread *)waitingThread);
184              waitingList->Remove(waitingThread);
185              break;
186          }
187      }
188      Terminated();
189  #else
190  DEBUG('t', "Finishing thread \"%s\"\n", getName());
191     threadToBeDestroyed = currentThread;
192     Sleep();     // invokes SWITCH
193     // not reached
194     #endif
195  }
```

```
435  // 新增代码39行 实现Thread::Join函数
436  #ifdef USER_PROGRAM
437  void Thread::Join(int SpaceId)  { //int join(SpaceId)
438      IntStatus oldLevel = interrupt->SetLevel(IntOff);   // 关中断
439      currentThread->waitingProcessSpaceId = SpaceId;     // 设置等待进程PID
440      //if joinee is still in not in terminated list?
441      // 判断等待的进程是否在已终止的队列中
442      Thread *thread;
443      List *terminatedList = scheduler->getTerminatedList();
444      List *waitingList = scheduler->getWaitingList();
445      bool interminatedList = FALSE;
446      int listLength = terminatedList->ListLength();  //length of ready queue
447      for (int i = 1; i <= listLength; i++)
448      {
449          thread = (Thread *)terminatedList->GetItem(i);
450          if (thread == NULL)
451              interminatedList = FALSE;   // joinee not finished
452          //joinee is still in Ready queue, not finished
453          if (thread->UserProgramId == SpaceId)
454          {
455              interminatedList = TRUE;  // joinee alreday finished
456              break;
457          }
458          interminatedList = FALSE;   // joinee not finished
459      }
460      //Joinee is not finished, still in not in
461      // terminated List(not at TERMINALTE),
462      if (!interminatedList)
463      // still in Nanchos system, maybe at READY or BLOCKED
464      {
465          waitingProcessSpaceId = SpaceId;
466          waitingList->Append((void *)this);   //blocked Joiner
467          currentThread->Sleep();
468      }
469      //joinee alreday finished, in terminated List, empty it, and return
470      currentThread->waitProcessExitCode = waitingThreadExitCode;
471      //delete terminated thread "Joinee"
472      scheduler->deleteTerminatedThread(SpaceId);
473      interrupt->SetLevel(oldLevel); // 开中断
474      }
475  #endif
476
```

```
// 新增代码21行 实现Thread::Terminated函数
#ifdef USER_PROGRAM
void Thread::Terminated()
{
    List *terminatedList = scheduler->getTerminatedList();

    Thread *nextThread;

    ASSERT(this == currentThread);  // a thread sleep by itsef
    ASSERT(interrupt->getLevel() == IntOff);
    status = TERNINATED;
    terminatedList->Append((void *)this);

    nextThread = scheduler->FindNextToRun();
    while(nextThread == NULL)
    {
        interrupt->Idle();
        nextThread = scheduler->FindNextToRun();
    }
    scheduler->Run(nextThread); // returns when we've been signalled
}
#endif

// 新增代码5行 得到退出代码
#ifdef USER_PROGRAM
int Thread::getExitStatus(){
    return exitCode;
}
#endif

// 新增代码5行 设置退出代码
#ifdef USER_PROGRAM
void Thread::setExitCode(int Code){
    exitCode = Code;
}
#endif
```

修改 userprog/addrspace.h

```
13  #ifndef ADDRSPACE_H
14  #define ADDRSPACE_H
15
16  #include "copyright.h"
17  #include "filesys.h"
18
19  #define UserStackSize   1024  // increase this as necessary!
20
21  class AddrSpace {
22    public:
23      AddrSpace(OpenFile *executable);  // Create an address space,
24          // initializing it with the program
25          // stored in the file "executable"
26      ~AddrSpace();     // De-allocate an address space
27
28      void InitRegisters();   // Initialize user-level CPU registers,
29          // before jumping to user code
30
31      void SaveState();     // Save/restore address space-specific
32      void RestoreState();    // info on a context switch
33
34      void Print();       // 新增代码 输出程序的页表
35      int GetSpaceId();   // 新增代码 获取进程spaceId
36
37    private:
38      TranslationEntry *pageTable;  // Assume linear page table translation
39          // for now!
40      unsigned int numPages;    // Number of pages in the virtual
41          // address space
42      int spaceId;// 新增代码 声明pid
43  };
44
45  #endif // ADDRSPACE_H
```

修改 userprog/addrspace.cc

```
22   #include "bitmap.h"// 新增代码 为了包含BitMap类
```

```
121 // 新增代码121~200行 新AddrSpace
122 AddrSpace::AddrSpace(OpenFile *executable)
123 {
124     bool hasAvailablePid = false; // 标记是否能够找到pid
125     for(int i = 100; i < MAX_USERPROCESSES; i++){
126         if(ThreadMap[i] == false){ // 找到后将ThreadMap数组对应位置1
127             ThreadMap[i] = true;
128             spaceId = i;
129             AddrSpaces[spaceId] = this;
130             hasAvailablePid = true;
131             break;
132         }
133     }
134     if(!hasAvailablePid){
135         printf("Too many processes in Nachos!\n");
136         return;
137     }
138     // Init ProBitMap
139     if(ProBitmap == NULL)
140         ProBitmap = new BitMap(NumPhysPages); // 初始化ProBitmap
141 
142     NoffHeader noffH;
143     unsigned int i, size;
144 
145     executable->ReadAt((char *)&noffH, sizeof(noffH), 0);//把可执行文件的信息读入
146     if ((noffH.noffMagic != NOFFMAGIC) &&
147         (WordToHost(noffH.noffMagic) == NOFFMAGIC))
148         SwapHeader(&noffH);//如果是小端机器，转换为大端机器
149     ASSERT(noffH.noffMagic == NOFFMAGIC);
150 
151 // how big is address space?
152     size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
153             + UserStackSize;    // we need to increase the size
154                     // to leave room for the stack
155     numPages = divRoundUp(size, PageSize);
156     size = numPages * PageSize;
157 
158     ASSERT(numPages <= NumPhysPages);       // check we're not trying
159                     // to run anything too big --
160                     // at least until we have
161                     // virtual memory
162     DEBUG('a', "Initializing address space, num pages %d, size %d\n",
163                     numPages, size);
164 // first, set up the translation
165     pageTable = new TranslationEntry[numPages]; // 现在虚拟地址不等于物理地址
166     for (i = 0; i < numPages; i++) {
167         pageTable[i].virtualPage = i;
168         pageTable[i].physicalPage = ProBitmap->Find();  //找到空的页框
169         pageTable[i].valid = TRUE;
170         pageTable[i].use = FALSE;
171         pageTable[i].dirty = FALSE;
172         pageTable[i].readOnly = FALSE;  // if the code segment was entirely on
173                     // a separate page, we could set its
174                     // pages to be read-only
175         bzero(&(machine->mainMemory[pageTable[i].physicalPage * PageSize]), PageSize);
176     }
```

```
178   // zero out the entire address space, to zero the unitialized data segment
179   // and the stack segment
180       //bzero(machine->mainMemory, size);//将内存清零
181
182   // then, copy in the code and data segments into memory
183       // 将代码段和数据段读入内存中
184       if (noffH.code.size > 0) {
185           int Phynum = pageTable[noffH.code.virtualAddr / PageSize].physicalPage * PageSize;
186           int offset = noffH.code.virtualAddr % PageSize;
187           DEBUG('a', "Initializing code segment, at 0x%x, size %d\n",
188               Phynum + offset, noffH.code.size);
189           executable->ReadAt(&(machine->mainMemory[Phynum + offset]),
190               noffH.code.size, noffH.code.inFileAddr);
191       }
192       if (noffH.initData.size > 0) {
193           int Phynum = pageTable[noffH.initData.virtualAddr / PageSize].physicalPage * PageSize;
194           int offset = noffH.initData.virtualAddr % PageSize;
195           DEBUG('a', "Initializing data segment, at 0x%x, size %d\n",
196               Phynum + offset, noffH.initData.size);
197           executable->ReadAt(&(machine->mainMemory[Phynum + offset]),
198               noffH.initData.size, noffH.initData.inFileAddr);
199       }
200   }
```

```
210   AddrSpace::~AddrSpace()
211   {
212       // 新增代码4行
213       ThreadMap[spaceId] = 0; //false
214       for (int i = 0; i < numPages; i++) {
215           ProBitmap->Clear(pageTable[i].physicalPage);
216       }
217      delete [] pageTable;
218   }
```

```
280   // 新增代码10行
281   void AddrSpace::Print()
282   {
283       printf("page table dump:  %d pages  in total\n", numPages);
284       printf("============================\n");
285       printf("\tVirtPage, \tPhysPage\n");
286       for (int i=0; i < numPages;  i++) {
287       printf("\t %d, \t\t%d\n", pageTable[i].virtualPage, pageTable[i].physicalPage);
288       }
289       printf("==========================================\n\n");
290   }
291
292   // 新增代码3行
293   int AddrSpace::GetSpaceId(){
294       return spaceId;
295   }
```

新建文件 userprog/progtest.h

OS > nachos-3.4 > code > userprog > C progtest.h > ...

```h
// progtest.h
#ifndef PROGTEST_H
#define PROGTEST_H

// 声明函数
void StartProcess(int spaceId);
void StartProcess(char *filename);
void ConsoleTest(char *in, char *out);

#endif
```

修改 userprog/progtest.cc

```cpp
#include "progtest.h"  //新增代码 添加头文件
//----------------------------------------------------------------------
// StartProcess
//  Run a user program.  Open the executable, load it into
//  memory, and jump to it.
//----------------------------------------------------------------------

// 新增代码9行 重载StartProcess(char *filename)
void StartProcess(int spaceId)
{
    AddrSpace *space = AddrSpaces[spaceId]; // 分配地址空间
    space->InitRegisters(); // set the initial register values
    space->RestoreState();  // load page table register

    machine->Run();   // jump to the user progam
    ASSERT(FALSE);  // machine->Run never returns;
        // the address space exits by doing the syscall "exit"
}

void
StartProcess(char *filename)
{
    OpenFile *executable = fileSystem->Open(filename);
    AddrSpace *space;

    if (executable == NULL) {
    printf("Unable to open file %s\n", filename);
    return;
    }
    space = new AddrSpace(executable);
// 为应用程序filename分配内存空间并将其装入所分配的内存空间中，然后建立页表，并建立虚页与实页（帧）的映射关系
// space 就是该进程的标识，而不是用pid来当进程标识
    currentThread->space = space;// 将该进程映射到一个核心进程
    space->Print();//新增代码 输出该作业的页表信息
    delete executable;          // close file

    space->InitRegisters();     // set the initial register values
    // 初始化 CPU 的寄存器
    space->RestoreState();      // load page table register

    machine->Run();         // jump to the user progam
    // 从程序入口开始，完成取指令、译码、执行的过程，直到进程遇到Exit()语句或者异常才退出
    ASSERT(FALSE);          // machine->Run never returns;
                    // the address space exits
                    // by doing the syscall "exit"
}

```

修改 lab7-8/exception.cc

```
#include "copyright.h"
#include "system.h"
#include "syscall.h"
#include "openfile.h"// 新增代码 为了包含OpenFile类
#include"progtest.cc"// 新增代码 为了包含StartProcess函数
```

```
// 在此处完成系统调用，$4-$7（4到7号寄存器）传递函数的前四个参数给子程序，参数多于4个时，其余的利用堆栈进行传递
void
ExceptionHandler(ExceptionType which)
{
    int type = machine->ReadRegister(2);

    if ((which == SyscallException)) {
        switch(type){
            case SC_Halt:{
                printf("CurrentThreadId: %d Name: %s, Execute system call of Halt() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
                DEBUG('a', "Shutdown, initiated by user program.\n");
                interrupt->Halt();
                break;
            }
            case SC_Exec:{
                printf("CurrentThreadId: %d Name: %s, Execute system call of Exec() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
                char filename[50];
                int addr = machine->ReadRegister(4);
                int i = 0;
                do{
                    machine->ReadMem(addr+i,1,(int*)&filename[i]);
                }while(filename[i++]!='\0');//读出文件名
                OpenFile *executable = fileSystem->Open(filename);//打开可执行文件
                if(executable == NULL){
                    printf("Unable to open file %s\n",filename);
                    return;
                }
                AddrSpace* space = new AddrSpace(executable);//创建地址空间
                space->Print();
                delete executable;//关闭文件
                char* forkedThreadName = filename;
                Thread* thread = new Thread(forkedThreadName);//创建线程
                thread->Fork(StartProcess,space->GetSpaceId());//开始执行线程
                thread->space = space;
                thread->UserProgramId=space->GetSpaceId();
                machine->WriteRegister(2,space->GetSpaceId());//返回子进程的ID
                AdvancePC();
                break;
            }
            case SC_Join:{
                printf("CurrentThreadId: %d Name: %s, Execute system call of Join() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
                int SpaceID=machine->ReadRegister(4);//读取子进程的ID
                currentThread->Join(SpaceID);//等待子进程结束
                machine->WriteRegister(2,currentThread->waitProcessExitCode);//返回子进程的返回值
                AdvancePC();
                break;
            }
            case SC_Exit:{
                printf("CurrentThreadId: %d Name: %s, Execute system call of Exit() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
                int ExitStatus = machine->ReadRegister(4); // 读取返回值
                currentThread->setExitCode(ExitStatus); // 设置返回值
                if(ExitStatus == 99){
                    List *terminatedList=scheduler->getTerminatedList();
                    scheduler->emptyList(terminatedList);
                }
                delete currentThread->space; // 删除地址空间
                currentThread->Finish();
                AdvancePC();
                break;
```

```
        }
        case SC_Yield:{
            printf("CurrentThreadId: %d Name: %s, Execute system call of Yield() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
            currentThread->Yield();
            AdvancePC();
            break;
        }
        case SC_Create:{
            printf("CurrentThreadId: %d Name: %s, Execute system call of FILESYS_STUB_SC_Create() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
            int base = machine->ReadRegister(4);
            int value;
            int count = 0;
            char *FileName = new char[128];
            do{
                machine->ReadMem(base + count, 1 ,&value);
                FileName[count] = (char)value;
                count++;
            }while((char)value != '\0' && count < 128);
            int fileDescriptor = OpenForWrite(FileName);
            if(fileDescriptor == -1){
                printf("create file %s failed!\n",FileName);
            }
            else{
                printf("create file %s succeed!,the file id is %d\n", FileName, fileDescriptor);
            }
            Close(fileDescriptor);
            AdvancePC();
            break;
        }
        case SC_Open:{
            printf("CurrentThreadId: %d Name: %s, Execute system call of FILESYS_STUB_SC_Open() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
            int base = machine->ReadRegister(4);
            int value;
            int count = 0;
            char *FileName = new char[128];
            do{
                machine->ReadMem(base + count, 1, &value);
                FileName[count] = (char)value;
                count++;
            }while(count < 128 && (char)value != '\0');
            int fileDescriptor = OpenForReadWrite(FileName,FALSE);
            if(fileDescriptor==-1){
                printf("Open file %s failed!\n",FileName);
            }
            else{
                printf("Open file %s succeed!, the file id is %d\n",FileName,fileDescriptor);
            }
            machine->WriteRegister(2,fileDescriptor);
            AdvancePC();
            break;
        }
```

```
        case SC_Write:{
            printf("CurrentThreadId: %d Name: %s, Execute system call of FILESYS_STUB_SC_Write() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
            int base = machine->ReadRegister(4);    // buffer
            int size = machine->ReadRegister(5);    // bytes written to file
            int fileId = machine->ReadRegister(6);  // fd
            int value;
            int count = 0;
            OpenFile *openfile = new OpenFile(fileId);
            ASSERT(openfile != NULL);
            char *buffer = new char[128];
            do{
                machine->ReadMem(base + count, 1, &value);
                buffer[count] = (char)value;
                count++;
            }while((char)value != '\0' && count < size);
            buffer[size] = '\0';
            int WritePostion;
            if (fileId == 1){
                WritePostion = 0;
            }
            else{
                WritePostion = openfile->Length();
            }
            int writtenBytes = openfile->WriteAt(buffer,size,WritePostion);
            if(writtenBytes == 0){
                printf("write file failed!\n");
            }
            else{
                printf("\"%s\" has wrote in file %d succeed!\n",buffer,fileId);
            }
            AdvancePC();
            break;
        }
        case SC_Read:{
            printf("CurrentThreadId: %d Name: %s, Execute system call of FILESYS_STUB_SC_Read() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
            int base = machine->ReadRegister(4);
            int size = machine->ReadRegister(5);
            int fileId = machine->ReadRegister(6);
            OpenFile *openfile = new OpenFile(fileId);
            char buffer[size];
            int readnum = 0;
            readnum = openfile->Read(buffer,size);
            for(int i = 0;i < size; i++){
                if(!machine->WriteMem(base,1,buffer[i]))
                    printf("This is something wrong.\n");
            }
            buffer[size]='\0';
            printf("read succeed!The content is \"%s\",the length is %d\n",buffer,size);
            machine->WriteRegister(2,readnum);
            AdvancePC();
            break;
        }
```

```
            case SC_Close:{
                printf("CurrentThreadId: %d Name: %s, Execute system call of FILESYS_STUB_SC_Close() \n",(currentThread->space)->GetSpaceId(),currentThread->getName());
                int fileId = machine->ReadRegister(4);
                Close(fileId);
                printf("File %d closed succeed!\n", fileId);
                AdvancePC();
                break;
            }
            default:{
                printf("Unexpected user mode exception %d %d\n", which, type);
                ASSERT(FALSE);
                break;
            }
        }
    } else {
    printf("Unexpected user mode exception %d %d\n", which, type);
    ASSERT(FALSE);
    }
}

```

修改 threads/list.h

```
class List {
  public:
    List();     // initialize the list
    ~List();      // de-allocate the list

    void Prepend(void *item);   // Put item at the beginning of the list
    void Append(void *item);  // Put item at the end of the list
    void *Remove();     // Take item off the front of the list

    void Mapcar(VoidFunctionPtr func);  // Apply "func" to every element
        // on the list
    bool IsEmpty();   // is the list empty?


    // Routines to put/get items on/off list in order (sorted by key)
    void SortedInsert(void *item, int sortKey); // Put item into list
    void *SortedRemove(int *keyPtr);      // Remove first item from list

    int ListLength();       // 新增代码 返回列表元素的个数
    void* GetItem(int i);   // 新增代码 返回第i个列表元素
    void Remove(void *item);// 新增代码 删除第i个列表元素

  private:
    ListElement *first;   // Head of the list, NULL if list is empty
    ListElement *last;    // Last element of list
    int num;                // 新增代码 列表元素的个数
};
```

修改 threads/list.cc

```
List::List()
{
    first = last = NULL;
    num = 0; // 新增代码 列表元素个数初始为0
}
```

```
77  void
78  List::Append(void *item)
79  {
80      ListElement *element = new ListElement(item, 0);
81
82      if (IsEmpty()) {        // list is empty
83      first = element;
84      last = element;
85      } else {            // else put it after last
86      last->next = element;
87      last = element;
88      }
89      num++;  // 新增代码 列表元素+1
90  }
```

```
104   void
105 List::Prepend(void *item)
106   {
107       ListElement *element = new ListElement(item, 0);
108
109       if (IsEmpty()) {        // list is empty
110       first = element;
111       last = element;
112       } else {            // else put it before first
113       element->next = first;
114       first = element;
115       }
116       num++;// 新增代码 列表元素+1
117   }
118
```

```
181  void
182 List::SortedInsert(void *item, int sortKey)
183  {
184      ListElement *element = new ListElement(item, sortKey);
185      ListElement *ptr;       // keep track
186
187      if (IsEmpty()) {    // if list is empty, put
188          first = element;
189          last = element;
190      } else if (sortKey < first->key) {
191          // item goes on front of list
192      element->next = first;
193      first = element;
194      } else {        // look for first elt in list bigger than item
195          for (ptr = first; ptr->next != NULL; ptr = ptr->next) {
196              if (sortKey < ptr->next->key) {
197      element->next = ptr->next;
198              ptr->next = element;
199      return;
200          }
201      }
202      last->next = element;       // item goes at end of list
203      last = element;
204      }
205      num++;// 新增代码 列表元素+1
206  }
```

```
void *
List::SortedRemove(int *keyPtr)
{
    ListElement *element = first;
    void *thing;

    if (IsEmpty())
    return NULL;

    thing = first->item;
    if (first == last) {    // list had one item, now has none
        first = NULL;
    last = NULL;
    } else {
        first = element->next;
    }
    if (keyPtr != NULL)
        *keyPtr = element->key;
    delete element;
    num--;// 新增代码 列表元素个数-1
    return thing;
}
```

```
// 新增代码 实现函数List::ListLength
int List::ListLength(){
    return num;
}

// 新增代码 实现函数List::GetItem
void* List::GetItem(int i){
    ListElement *ptr = first;
    for(int j=0;j<i;j++){
        ptr=ptr->next;
    }
    return ptr->item;
}

// 新增代码 实现函数List::Remove
void List::Remove(void *item){
    ListElement *ptr = first;
    ListElement *pre = NULL;
    while(ptr!=NULL){
        if(ptr->item==item){
            if(pre==NULL){
                first=ptr->next;
            }else{
                pre->next=ptr->next;
            }
            if(ptr==last){
                last=pre;
            }
            delete ptr;
            num--;
            return;
        }
        pre=ptr;
        ptr=ptr->next;
    }
}
```

修改 threads/system.h

```
17  #include "timer.h"
18
19  #include "addrspace.h"  // 新增代码 包含AddrSpace类
20  #include "bitmap.h"     // 新增代码 包含BitMap类
21  #define MAX_USERPROCESSES 256 // 新增代码 定义最大用户进程数量
22
23  // Initialization and cleanup routines
```

```
36  extern BitMap *ProBitmap; // 新增代码 for free frame
37  extern bool ThreadMap[MAX_USERPROCESSES];// 新增代码
38  extern AddrSpace* AddrSpaces[MAX_USERPROCESSES];// 新增代码
```

修改 threads/system.cc

```
20                      // for invoking context switches
21  #define MAX_USERPROCESSES 256              // 新增代码 定义最大进程用户数量
22  BitMap *ProBitmap;                         // 新增代码 管理空闲帧
23  bool ThreadMap[MAX_USERPROCESSES];         // 新增代码管理进程的spaceId
24  AddrSpace* AddrSpaces[MAX_USERPROCESSES];  // 新增代码 进程地址空间数组
25
```

检验代码

SpaceId 和 OpenFileId 其实都是 int
需要测试的功能：
Halt——void Halt();——停机
Exit——void Exit(int status);——Exit(0)是正常退出
　参数：status 是进程的退出码
Exec——SpaceId Exec(char *name);——执行文件
　参数：name 是文件名称
　返回值：产生的进程的 pid
Join——int Join(SpaceId id);——主动阻塞当前进程
　参数：id 是等待目标进程
　返回值：等待目标进程的退出码
Create——void Create(char *name);——创建文件
　参数：name 是文件名称
Open——OpenFileId Open(char *name);——打开文件
　参数：name 是文件名称
　返回值：打开文件的标识 id
Read——int Read(char *buffer, int size, OpenFileId id);
　参数：从文件 id 中读取 size 个字节到 buffer 中

返回值：实际读取数据个数，有时候文件不够长，或文件不可读取

Write——void Write(char *buffer, int size, OpenFileId id);——写文件

参数：向文件 id 中写入 size 个 buffer 中的字节数据

Close——void Close(OpenFileId id);——关闭文件

参数：文件 id

Yield——void Yield();——让出 CPU

总共设计两个文件：

1、test.c，该文件会测试所用系统调用

2、exit.c，辅助 test.c

设计 test/exit.c 文件

设计 test/test.c 文件：

```c
/*
 * Nachos系统调用测试程序
 * 测试功能：Halt, Exit, Exec, Join, Create, Open, Read, Write, Close, Yield
 */

#include "syscall.h"
#define TEST_FILE "testfile"    // 测试文件名
#define TEST_CONTENT "Hello Nachos File System!\n"  // 测试文件内容
#define BUF_SIZE 256            // 缓冲区大小
static int a[40];//新增代码，分配更大的地址空间
//--------------------------------------
// 测试函数1：文件操作（Create/Open/Write/Read/Close）
//--------------------------------------
void TestFileOps() {
    char buffer[BUF_SIZE];
    OpenFileId fd;
    int bytesRead;

    // 1. 创建并写入文件
    Create(TEST_FILE);                  // 创建文件
    fd = Open(TEST_FILE);               // 打开文件
    Write(TEST_CONTENT, sizeof(TEST_CONTENT)-1, fd); // 写入内容
    Close(fd);                          // 关闭文件

    // 2. 重新打开并验证内容
    fd = Open(TEST_FILE);
    bytesRead = Read(buffer, sizeof(TEST_CONTENT)-1, fd);
    Close(fd);
}

//--------------------------------------
// 测试函数2：进程控制（Exec/Join/Exit）
//--------------------------------------
void ChildProcess() {
    //Write("Child Process Running\n", 22, ConsoleOutput);
    Exit(0);  // 子进程正常退出
}

void TestProcessControl() {
    SpaceId pid;
    int exitStatus;

    // 启动子进程
    pid = Exec("../test/exit.noff");  // 假设test1是另一个测试程序
    exitStatus = Join(pid);

}

//--------------------------------------
// 测试函数3：Fork/Yield协作
//--------------------------------------
void ForkedFunction() {
    int i;
    for (i = 0; i < 3; i++) {
        //Write("Forked Thread Yield\n", 20, ConsoleOutput);
        Yield();  // 主动让出CPU
    }
    Exit(0);
}

void TestForkYield() {
    int i;
    //Fork(ForkedFunction);
    for (i = 0; i < 3; i++) {
        //Write("Main Thread Yield\n", 18, ConsoleOutput);
        Yield();
    }
}
 //--------------------------------------
 // 主测试程序
 //--------------------------------------
int main() {
    // 执行所有测试
    TestFileOps();       // 文件操作测试
    TestProcessControl();// 进程控制测试
    //TestForkYield();     // Fork/Yield测试
    // 最终停机
    Halt();
}
```

运行结果如下：

```
zhang@zhang:~/OS/nachos-3.4/code/lab7-8$ ./nachos -x ../test/test.noff
page table dump:  16 pages  in total
============================
        VirtPage,       PhysPage
         0,             0
         1,             1
         2,             2
         3,             3
         4,             4
         5,             5
         6,             6
         7,             7
         8,             8
         9,             9
         10,            10
         11,            11
         12,            12
         13,            13
         14,            14
         15,            15
========================================

CurrentThreadId: 100 Name: main, Execute system call of FILESYS_STUB_SC_Create()
create file testfile succeed!,the file id is 3
CurrentThreadId: 100 Name: main, Execute system call of FILESYS_STUB_SC_Open()
Open file testfile succeed!, the file id is 3
CurrentThreadId: 100 Name: main, Execute system call of FILESYS_STUB_SC_Write()
"Hello Nachos File System!
" has wrote in file 3 succeed!
CurrentThreadId: 100 Name: main, Execute system call of FILESYS_STUB_SC_Close()
File 3 closed succeed!
CurrentThreadId: 100 Name: main, Execute system call of FILESYS_STUB_SC_Open()
Open file testfile succeed!, the file id is 3
CurrentThreadId: 100 Name: main, Execute system call of FILESYS_STUB_SC_Read()
read succeed!The content is "Hello Nachos File System!
",the length is 26
CurrentThreadId: 100 Name: main, Execute system call of FILESYS_STUB_SC_Close()
File 3 closed succeed!
CurrentThreadId: 100 Name: main, Execute system call of Exec()
page table dump:  10 pages  in total
============================
```

```
============================
        VirtPage,       PhysPage
         0,             16
         1,             17
         2,             18
         3,             19
         4,             20
         5,             21
         6,             22
         7,             23
         8,             24
         9,             25
========================================

CurrentThreadId: 100 Name: main, Execute system call of Join()
CurrentThreadId: 101 Name: ../test/exit.noff, Execute system call of Exit()
CurrentThreadId: 100 Name: main, Execute system call of Halt()
Machine halting!

Ticks: total 159, idle 0, system 40, user 119
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
zhang@zhang:~/OS/nachos-3.4/code/lab7-8$
```