Lab4 虚拟实习报告

姓名 姜慧强 学号 1801210840
日期 2019.05.10

目录

内容一：总体概述

本次实验室操作系统高级课程的第四次实验。在 Lab3 阅读同步机制，完成信号量、条件变量解决生产者-消费者基础上，主要完成了 Nachos 内存管理相关代码的实践。Nashocs 中对内存的管理，主要通过使用 PageTable，加上 TLB 缓存实现内存的高效读写操作。并在代码阅读的基础上，TLB 异常处理，TLB Miss 处理，基于 bitmap 的内存管理机制，对多线程的支持，pageTable Miss 的处理，对加载到内存过程的 Lazy loading 加载策略， 实现倒排 PageTable 等任务。在实践过程中，增强对操作系统对内存的管理管理的认识，加强了对 bitMap，移位操作等的训练。总的来说本次实验相较于前几次实验，难度有明显的提升。在实验过程中，也很好的提升了自己代码实践能力。

内容二：任务完成情况

任务完成列表（Y/N）

EXER1	EXER2	EXER3	EXER4	EXER5	EXER6	EXER7	C1	C2
Y	Y	Y	Y	Y	Y	Y	N	Y

具体 Exercise 的完成情况

一、TLB 异常处理

Exercise1

userprog/progtest.cc 是本次实验的入口函数，其通过 thread/main.cc 中 #ifdef USER_PROGRAM 模式进入，随之传递 filename 参数。该模式目前支持-x,-c 两种模式。

在 Userprog/progtest.cc 中,通过调用 fileSystem，AddressSpace 的代码，实现打开文件，在内存空间中建立相对应的地址空间，初始化 register，保存 PageTable 信息。随即执行相应的 file 中的代码。

整个 userprog/progtest.cc，充当 thread/threadtest.cc 一致的功能，可以算是入口函数，也可以说成是一个测试用例。

在该文件中，除了 StartPross 之外，还存在一个 ConsoleTest 函数，该函数用来进行在 terminal 中打印输入的变量值。在本项目中，未使用该部分代码。

userprog/addreSpace.cc 和.h 文件负责用户地址空间的管理，主要是转换大小端地址（SwapHeader），维护相应的 PageTable（AddrSpace），回收 PageTable(~AddrSpace),初始化寄存器（InitRegisters）,在交换上下文时保存地址空间信息(RestoreState).

其中 AddrSpace 中根据文件头中信息，计算文件存储到内存中所需空间，并按该空间大小做一系列的内存预分配，分配 PageTable。需要注意的是目前，PageTable 的虚拟地址等于物理地址，预示着可用的内存空间较小。

然后在 InitRegisters 函数中对寄存器进行了初始化。初始化普通寄存器为 0，当前指令寄存器 PCReg 为 4，下一指令寄存器为 4，并分配栈空间，numpages * PageSize – 16。

RestoreState 函数用于将 AddressSpace 中对 PageTable 的管理相关的参数序列化到 Machine 中。

Exception.cc 文件中定义了一系列对 exception 的处理过程。其中传入的是一个 Exception 类型 which。Execption 的具体参数通过 2 寄存器保存。在接下来的实践中，需要在该部分对 PageFaultException 进行处理。

在 syscall.h 文件中，定义了一系列 system call 的代码，包括 Halt，exit，exec，join，create，等等.

此外还实现了 BitMap 函数.

Machine run 定义在 machine/mipssim.cc 中。其中先设置把中断模式设置为用户模式。然后循环取指令（OneInstruction），执行指令（interrupt-）OneTick）。

在 OneInstruction 中，每次从 TLB 和 PageTable 中取 4 个字节的指令，machine->ReadMem(registers[PCReg], 4, &raw)。若取得到就继续，否则抛异常。

然后根据取到的指令，做一个很大的 switch，找到相应的所需要的操作动作，对 register 进行相应的操作。

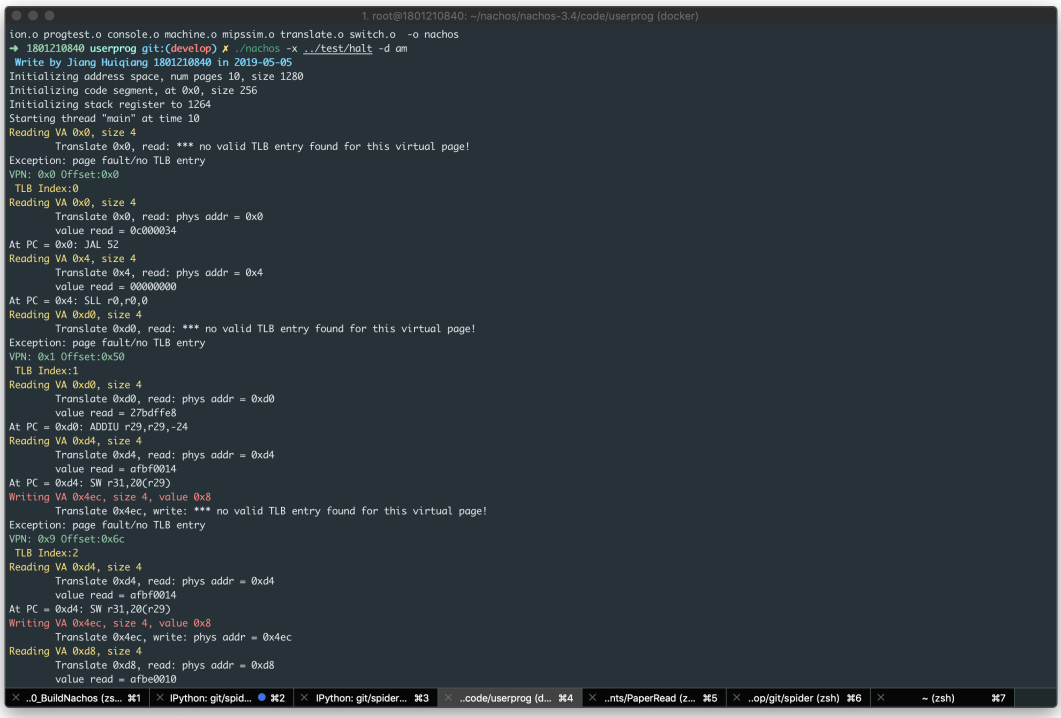
Exercise2

在做 TLB 之前，需要开启-DUSE_TLB.
PageFaultException 抛异常之后，有可能是 TLBMiss，也有可能是 PageTableMiss，这个时候我们需要根据存在 Register 里面的 BadVAddrReg，获得 vpn，根据 vpn 来判断是那种 PageFault。

然后对于 TLBMiss，从 Pagetable 中取出相应的 TranslationEntry 值。

```
void PageFaultHandler()
{
    int virtAddr, emptyTLBIndex = 0;

    virtAddr = machine->ReadRegister(BadVAddrReg);
    vpn = (unsigned)virtAddr / PageSize;
```

```
  offset = (unsigned)virtAddr % PageSize;
  DEBUG('a', "\033[92mVPN: 0x%x Offset:0x%x\033[0m\n", vpn, offset);

  if (machine->tlb == NULL)
  {
    PageTableFaultHandler(vpn);
  }
  else
  {
    while (emptyTLBIndex < TLBSize && machine->tlb[emptyTLBIndex].valid)
      ++emptyTLBIndex;


    if (machine->pageTable[vpn].valid)
    {
      printf("\033[92m TLB Index:%d \033[0m\n", emptyTLBIndex);
      machine->tlb[emptyTLBIndex] = machine->pageTable[vpn];
      ++tlbTime[emptyTLBIndex];
      ++TLBMiss;
      ++time;
    }
    else
    {
      PageTableFaultHandler(vpn);
    }
  }
}
```



Exercise3

Exercise3 在 TLB Miss 处理的基础上，对 TLB 换出策略做优化。在这里我实现了最少访问次数和 LRU 两种策略。

最少访问次数利用一个数组记录访问次数，然后但 TLB Miss 没有空闲的 TLB 的时候，遍历该数组取访问次数最小的一个换出。

```
int minTime = 0x3fffffff;
 for (int i = 0; i < TLBSize; ++i)
 {
 DEBUG('a', "\033[92m TLBTime: %d \033[0m\n", tlbTime[i]);
 if (tlbTime[i] < minTime)
 {
 minTime = tlbTime[i];
 emptyTLBIndex = i;
 }
```

LRU 则在 machine 中维护一个全局数组，当 TLB 命中和 Miss 均需要去 update 该数组内容。当命中了，将小于等于选中值的加一。

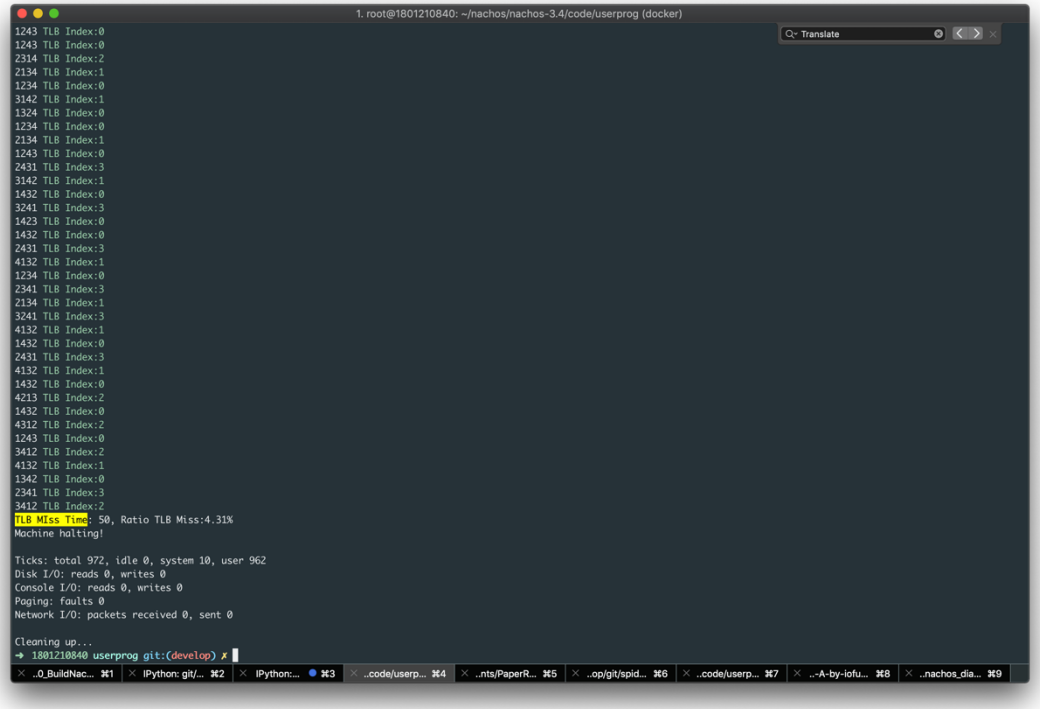
```
        int lruNum= LRUTLB[i];
        for(int j = 0;j < TLBSize;++j){
```

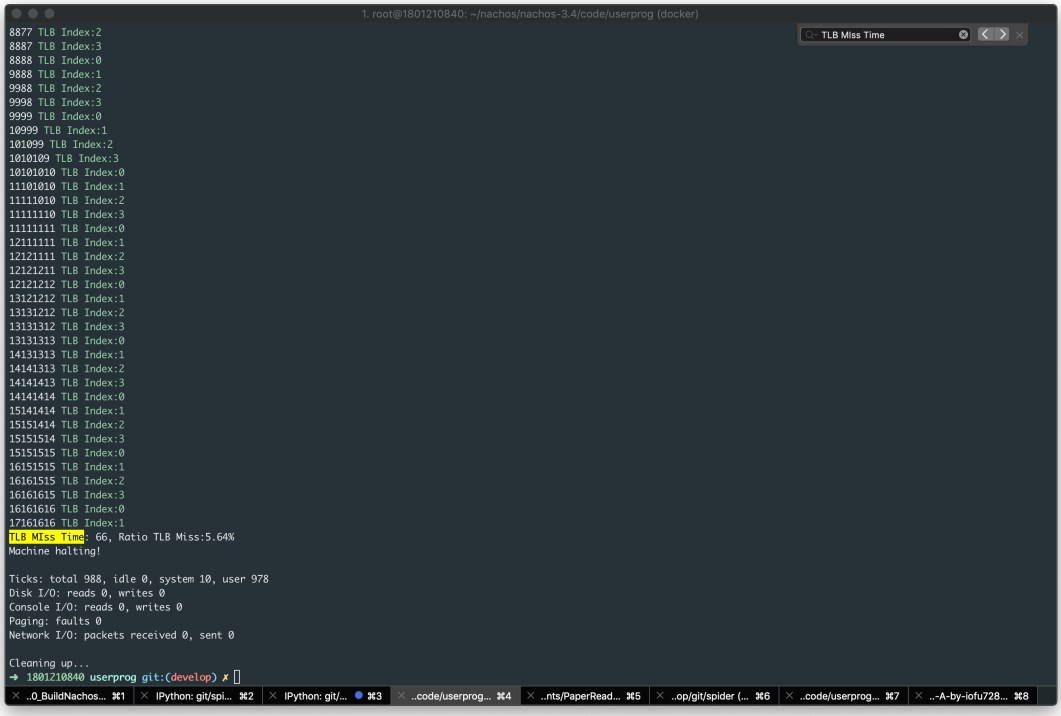
```
if (LRUTLB[j] <lruNum && LRUTLB[j] < TLBSize) ++LRUTLB[j];
     if(i==j) LRUTLB[j] =1;
}
```

当未命中时，对被替换的做相同操作

```
int maxTime = -1;
for (int i = 0; i < TLBSize; ++i)
if (machine->LRUTLB[i] > maxTime)
{
maxTime = machine->LRUTLB[i];
emptyTLBIndex = i;
}
int lruNUm = machine->LRUTLB[emptyTLBIndex];
for (int i = 0; i < TLBSize; ++i)
{
if (machine->LRUTLB[i] <= lruNUm && machine->LRUTLB[i] < TLBSize)
         ++machine->LRUTLB[i];
        if (emptyTLBIndex == i)
             machine->LRUTLB[i] = 1;
DEBUG('a', "\033[92m LRUTLB: %d \n", machine->LRUTLB[i]);
}
```

得到的结果 LRU TLB Miss 率要低一点





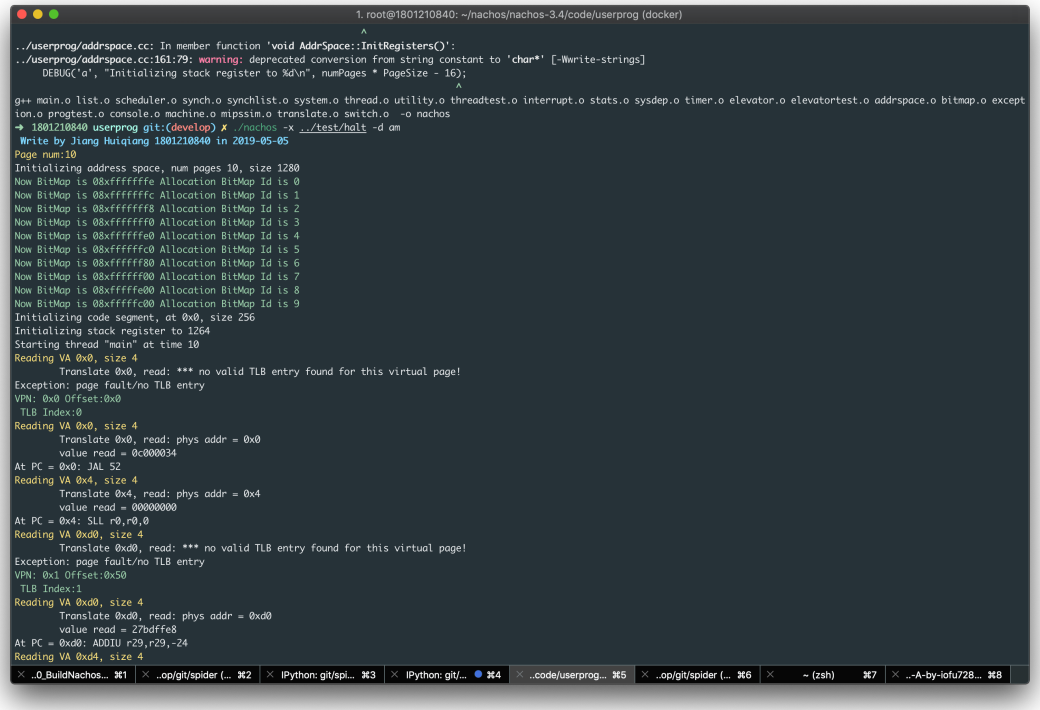
二、分页式内存管理

Exercise 4

实现一个 bitMap 来维护虚拟内存。bitMap 使用 unsigned int。PageSize 32 块，刚好和 unsigned int 一致。利用位运算来维护 bitMap

```
int Machine::AllocationMemory()
{
    int allocationPower = (bitMap & (~(bitMap - 1))), allocationId = 0;
    if (!allocationPower) return -1;
    while (allocationPower >> allocationId != 1)
        ++allocationId;
    bitMap = bitMap ^ allocationPower;
    DEBUG('a', "\033[92mNow BitMap is 08x%x Allocation BitMap Id is %d
\n\033[0m", bitMap, allocationId);
    return allocationId;
}
void Machine::DeallocationMemory(int index)
{
    bitMap = bitMap ^ (1 << index);
    DEBUG('a', "\033[92mNow BitMap is 08x%x \n\033[0m", bitMap);
}
```

```
void Machine::ClearMemory()
{
    bitMap = -1;
    DEBUG('a', "\033[92mNow BitMap is 08x%x \n\033[0m", bitMap);
}
```



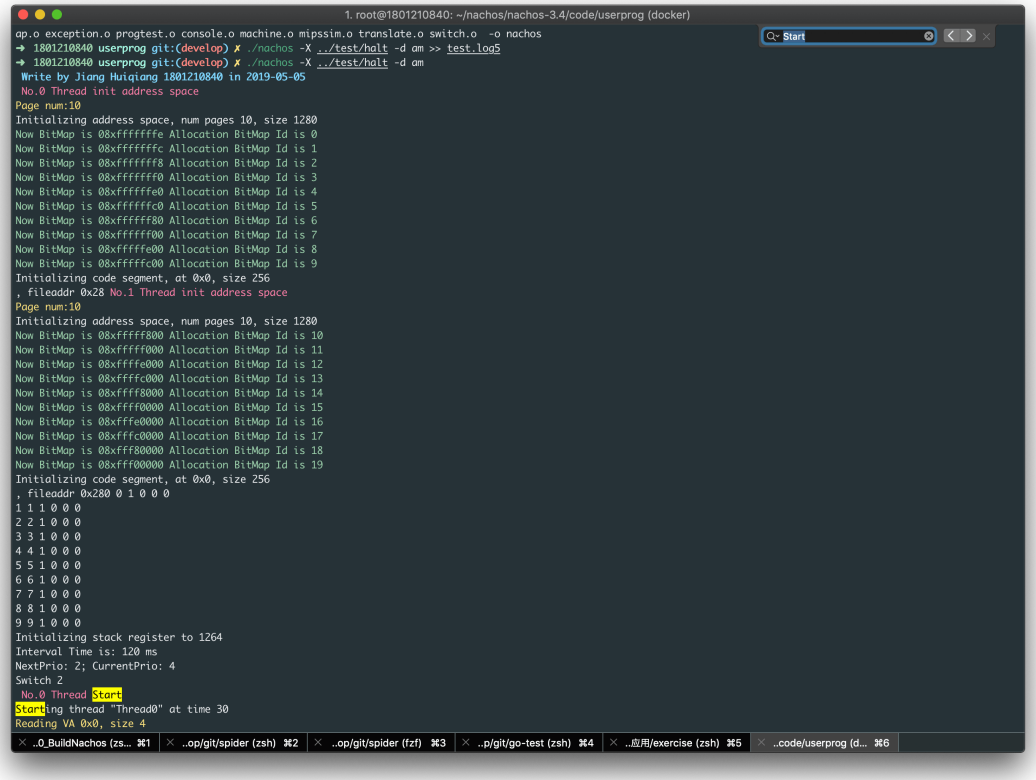
Exercise 5

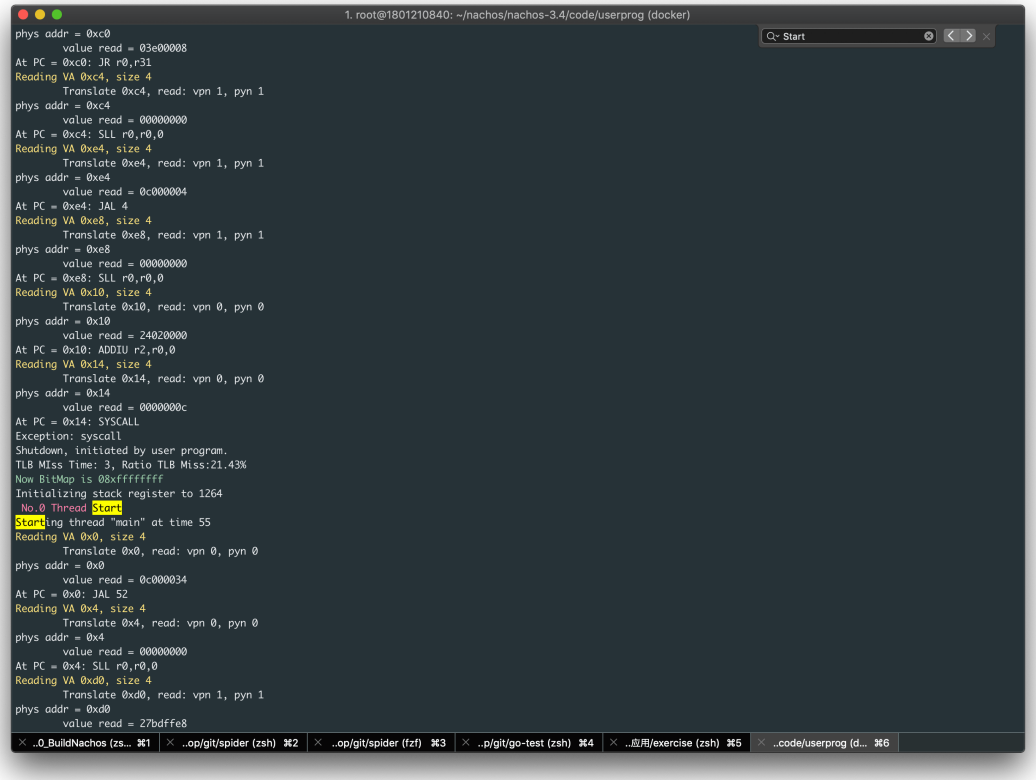
之前的 nachos 不支持多线程是因为一开始就初始化了 PageTable 使得，多个线程拥有相对独立的存储环境变得不可能。故通过维护多个 addressSpace 来实现多线程。

在 progtest 中增加一个多线程模式，批量维护线程，Address，来实现多个线程同时在内存中。

```
void ForkThread(int num){
    printf("\033[95m No.%d Thread Start \033[0m\n", num);
    machine->Run();
}

void StartMultiProcess(char *filename, int threadNum){
    printf("\033[01;34m Write by Jiang Huiqiang 1801210840 in 2019-05-05
\033[0m\n");
    if (!threadNum) {
        printf("\033[95m ThreadNum should be a num \033[0m\n");
        return;
    }
    OpenFile *executable[threadNum] = {};
    AddrSpace *space[threadNum] = {};
    Thread *thread[threadNum] = {};
    char threadNameList[MaxThreadNum][20] = {};
    for (int i = 0; i < threadNum; ++i) executable[i] =
fileSystem->Open(filename);
    for (int i = 0; i < threadNum - 1; ++i) {
        char str[20];
        sprintf(str, "%d", i);
        strcat(threadNameList[i], "Thread");
        strcat(threadNameList[i], str);
        thread[i] = new Thread(threadNameList[i]);
    }

    if (executable[0] == NULL){
        printf("Unable to open file %s\n", filename);
        return;
    }
    for (int i = 0; i < threadNum; ++i) {
        printf("\033[95m No.%d Thread init address space \033[0m\n", i);
        space[i] = new AddrSpace(executable[i]);
    }

    currentThread->space = space[threadNum - 1];
```

```
    for (int i = 0; i < threadNum - 1; ++i){
        space[i]->printPageTable();
        space[i]->InitRegisters();
        space[i]->RestoreState();
        thread[i]->space = space[i];
        thread[i]->setPriority(2);
        thread[i]->Fork(ForkThread, (void *)i);
        currentThread->Yield();
    }
    for (int i = 0; i < threadNum; ++i) delete executable[i];
    space[0]->InitRegisters();
    space[0]->RestoreState();
    printf("\033[95m No.0 Thread Start \033[0m\n");

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


1. root@1801210840: ~/nachos/nachos-3.4/code/userprog (docker)
ap.o exception.o progtest.o console.o machine.o mipssim.o translate.o switch.o -o nachos
➜ 1801210840 userprog git:(develop) ✗ ./nachos -X ../test/halt -d am >> test.log5
➜ 1801210840 userprog git:(develop) ✗ ./nachos -X ../test/halt -d am
Write by Jiang Huiqiang 1801210840 in 2019-05-05
No.0 Thread init address space
Page num:10
Initializing address space, num pages 10, size 1280
Now BitMap is 08xfffffffe Allocation BitMap Id is 0
Now BitMap is 08xfffffffc Allocation BitMap Id is 1
Now BitMap is 08xfffffff8 Allocation BitMap Id is 2
Now BitMap is 08xfffffff0 Allocation BitMap Id is 3
Now BitMap is 08xffffffe0 Allocation BitMap Id is 4
Now BitMap is 08xffffffc0 Allocation BitMap Id is 5
Now BitMap is 08xffffff80 Allocation BitMap Id is 6
Now BitMap is 08xffffff00 Allocation BitMap Id is 7
Now BitMap is 08xfffffe00 Allocation BitMap Id is 8
Now BitMap is 08xfffffc00 Allocation BitMap Id is 9
Initializing code segment, at 0x0, size 256
, fileaddr 0x28 No.1 Thread init address space
Page num:10
Initializing address space, num pages 10, size 1280
Now BitMap is 08xfffff800 Allocation BitMap Id is 10
Now BitMap is 08xfffff000 Allocation BitMap Id is 11
Now BitMap is 08xffffe000 Allocation BitMap Id is 12
Now BitMap is 08xffffc000 Allocation BitMap Id is 13
Now BitMap is 08xffff8000 Allocation BitMap Id is 14
Now BitMap is 08xffff0000 Allocation BitMap Id is 15
Now BitMap is 08xfffe0000 Allocation BitMap Id is 16
Now BitMap is 08xfffc0000 Allocation BitMap Id is 17
Now BitMap is 08xfff80000 Allocation BitMap Id is 18
Now BitMap is 08xfff00000 Allocation BitMap Id is 19
Initializing code segment, at 0x0, size 256
, fileaddr 0x280 0 1 0 0 0
1 1 1 0 0 0
2 2 1 0 0 0
3 3 1 0 0 0
4 4 1 0 0 0
5 5 1 0 0 0
6 6 1 0 0 0
7 7 1 0 0 0
8 8 1 0 0 0
9 9 1 0 0 0
Initializing stack register to 1264
Interval Time is: 120 ms
NextPrio: 2; CurrentPrio: 4
Switch 2
No.0 Thread Start
Starting thread "Thread0" at time 30
Reading VA 0x0, size 4
Start
..0_BuildNachos (zs... ⌘1
..op/git/spider (zsh) ⌘2
..op/git/spider (fzf) ⌘3
..p/git/go-test (zsh) ⌘4
..应用/exercise (zsh) ⌘5
..code/userprog (d... ⌘6


1. root@1801210840: ~/nachos/nachos-3.4/code/userprog (docker)
phys addr = 0xc0
value read = 03e00008
At PC = 0xc0: JR r0,r31
Reading VA 0xc4, size 4
Translate 0xc4, read: vpn 1, pyn 1
phys addr = 0xc4
value read = 00000000
At PC = 0xc4: SLL r0,r0,0
Reading VA 0xe4, size 4
Translate 0xe4, read: vpn 1, pyn 1
phys addr = 0xe4
value read = 0c000004
At PC = 0xe4: JAL 4
Reading VA 0xe8, size 4
Translate 0xe8, read: vpn 1, pyn 1
phys addr = 0xe8
value read = 00000000
At PC = 0xe8: SLL r0,r0,0
Reading VA 0x10, size 4
Translate 0x10, read: vpn 0, pyn 0
phys addr = 0x10
value read = 24020000
At PC = 0x10: ADDIU r2,r0,0
Reading VA 0x14, size 4
Translate 0x14, read: vpn 0, pyn 0
phys addr = 0x14
value read = 0000000c
At PC = 0x14: SYSCALL
Exception: syscall
Shutdown, initiated by user program.
TLB MIss Time: 3, Ratio TLB Miss:21.43%
Now BitMap is 08xffffffff
Initializing stack register to 1264
No.0 Thread Start
Starting thread "main" at time 55
Reading VA 0x0, size 4
Translate 0x0, read: vpn 0, pyn 0
phys addr = 0x0
value read = 0c000034
At PC = 0x0: JAL 52
Reading VA 0x4, size 4
Translate 0x4, read: vpn 0, pyn 0
phys addr = 0x4
value read = 00000000
At PC = 0x4: SLL r0,r0,0
Reading VA 0xd0, size 4
Translate 0xd0, read: vpn 1, pyn 1
phys addr = 0xd0
value read = 27bdffe8
Start
..0_BuildNachos (zs... ⌘1
..op/git/spider (zsh) ⌘2
..op/git/spider (fzf) ⌘3
..p/git/go-test (zsh) ⌘4
..应用/exercise (zsh) ⌘5
..code/userprog (d... ⌘6

Exercise 6

之前我们已经对 TLB Miss 做了处理，如果 PageTable 也 Miss 掉了，就需要利用 fileSystem 从 disk 中 load 出来数据。

```
void PageTableFaultHandler(unsigned int vpn)
{

  DEBUG('a', "\033[91mPage Table Fault \033[0m\n");
  OpenFile *openfile = fileSystem->Open("virtual_memory");
  int pos = machine->AllocationMemory();
  if (pos == -1) {
    pos = 0;
    for (int i = 0; i < machine->pageTableSize; ++i){
      if (machine->pageTable[i].physicalPage == 0) {
        if (machine->pageTable[i].dirty == TRUE) {
          openfile->WriteAt(&(machine->mainMemory[pos * PageSize]),
            PageSize, machine->pageTable[i].virtualPage * PageSize);
          machine->pageTable[i].valid = FALSE;
          break;
        }
      }
    }

  }
  openfile->ReadAt(&(machine->mainMemory[pos * PageSize]), PageSize, vpn *
PageSize);
  machine->pageTable[vpn].valid = TRUE;
  machine->pageTable[vpn].physicalPage = pos;
  machine->pageTable[vpn].use = FALSE;
  machine->pageTable[vpn].dirty = FALSE;
  machine->pageTable[vpn].readOnly = FALSE;
  delete openfile;

  ASSERT(FALSE);
}
```

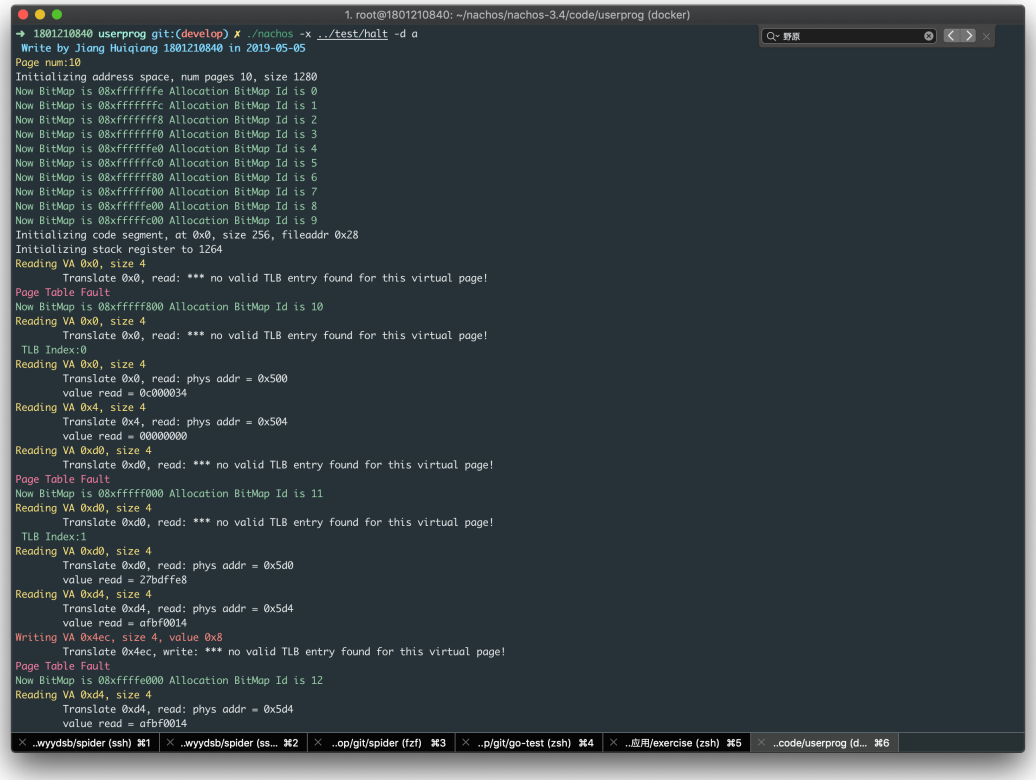
测试环节与 Exercise 共用

Exercise 7

在之前的 Nachos 版本中，其实一定不会出现 PageTable Miss 的，因为一开始初始化的时候，就把所有需要的 PageTable 都 load 到内存。为了实现 lazy

load 在一开始初始化的时候把需要加载的内存想 load 到虚拟内存中，当出现 PageTableFault 时，再去 bitMap 找空位换入。但 bitMap 中没有空位的时候需要选择一个换出，注意如果改 PageTabel 被修改，则需要 load 到 disk。

```
    fileSystem->Create("virtual_memory", size);
    OpenFile *openfile = fileSystem->Open("virtual_memory");
    if (openfile == NULL) ASSERT(false);
    if (noffH.code.size > 0){
        DEBUG('a', "Initializing code segment, at 0x%x, size %d, fileaddr
0x%x\n",
              noffH.code.virtualAddr, noffH.code.size,
noffH.code.inFileAddr);
        char char_stream[numPages * PageSize];
        executable->ReadAt((char_stream), noffH.code.size,
noffH.code.inFileAddr);
        openfile->WriteAt((char_stream), noffH.code.size,
noffH.code.virtualAddr);
        // executable->ReadAt(&(machine->mainMemory[noffH.code.virtualAddr]),
        //                 noffH.code.size, noffH.code.inFileAddr);
    }
    if (noffH.initData.size > 0){
        char char_stream[numPages * PageSize];
        executable->ReadAt((char_stream), noffH.initData.size,
noffH.initData.inFileAddr);
        openfile->WriteAt((char_stream), noffH.initData.size,
noffH.initData.virtualAddr * PageSize);
        DEBUG('a', "Initializing data segment, at 0x%x, size %d, fileaddr
0x%x\n",
              noffH.initData.virtualAddr, noffH.initData.size,
noffH.initData.inFileAddr);
    }
    delete openfile;
```

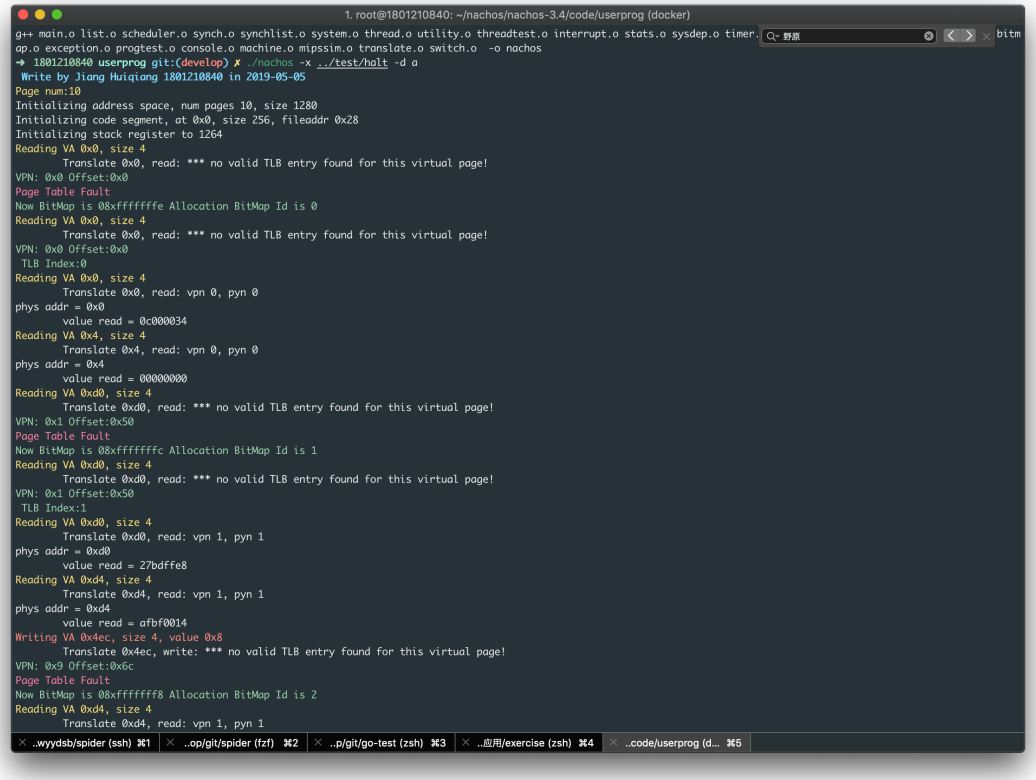


Challenge 2

需要实现倒排 PageTable，把原本虚拟内存 ID 作为的索引改成物理内存 ID 作为索引。直观的想法就是改原来的 PageTable 项改一下上下界，索引值。

```
        pageTable = new TranslationEntry[NumPhysPages];
        for (i = 0; i < NumPhysPages; i++) {
            pageTable[i].virtualPage = i; // for now, virtual page # = phys page
    #s
            pageTable[i].physicalPage = i;
            pageTable[i].valid = FALSE;
            pageTable[i].use = FALSE;
            pageTable[i].dirty = FALSE;
            pageTable[i].readOnly = FALSE; // if the code segment was entirely on
                                           // a separate page, we could set its
                                           // pages to be read-only
        }
      openfile->ReadAt(&(machine->mainMemory[pos * PageSize]), PageSize, vpn *
    PageSize);
      machine->pageTable[pos].valid = TRUE;
      machine->pageTable[pos].virtualPage = vpn;
      machine->pageTable[pos].use = FALSE;
```

```
machine->pageTable[pos].dirty = FALSE;
machine->pageTable[pos].readOnly = FALSE;
```



内容三：遇到的困难以及解决方法

在 Exercise5 中遇到过出现 PageTableFault 的情况，我们知道之前的版本是一开始就把文件内容 load 到内存中，正常情况是不可能出现 PageTableFault。通过 Debug，感觉像是文件内容读错了，导致指令取错了。因为维护了两个 addressSpace 怀疑是不是两个 AddressSpace 维护的 PageTable 出了问题。把 PageTable 打出来，发现并没有问题，对比汇编代码，感觉是第二个 space 读取的文件信息出了问题。所以决定对调一下 Thread 执行顺序，让 Addressspace 依次读取到文件，解决了问题。

内容四：收获及感想

这次的作业的难度比前面三个 lab 有所增加，做起来还是比较吃力的，尤其是 Exercise5 的实现，还是很有意思的。通过这次实践，强化了自己对内存管理，对文件系统的理解。

内容五：对课程的意见和建议

感觉自己这个 lab 时间脱得有点久，有点对不住老师，hhh.

内容六：参考文献

[1] Stevens, W. R. (2002). UNIX 环境高级编程：英文版. 机械工业出版社.