山东大学 计算机科学与技术 学院

操作系统课程设计 课程实验报告

| | | |
|---|---|---|
| 学号：202200130193 | 姓名：张景岳 | 班级：22.6 班 |
| 实验题目：Nachos | | |
| 实验学时：很多 | 实验日期：2025.3.17-2025.5.8 | |
| 实验目的：操作系统课设 | | |
| 实验环境：Win11 WSL | | |
| 源程序清单：code/** | | |

编译及运行结果：

代码库 github 地址：SDU-YueQiu/Nachos: 山大操作系统课设

记录了实验过程的代码修改记录。

# 实验一

为防止混淆以下报告中用“主线程”指代代码 Initialize 函数中创建的名为“main”的线程，用 main()指代程序代码中的 main()函数。

思考：Nachos 如何判定我使用的系统？

Nachos 使用宏记录用户系统信息，在执行硬件/系统相关指令时候根据宏做操作：

```
 of liability and disclaimer of warranty provisions.
 */

#include "copyright.h"
#include "switch.h"

#ifdef HOST_i386

        .text
        .align  2

        .globl  ThreadRoot

/* void ThreadRoot( void )
**
** expects the following registers to be initialized:
**      eax     points to startup function (interrupt enable)
**      edx     contains inital argument to thread function
**      esi     points to thread function
**      edi     point to Thread::Finish()
*/
23 references
ThreadRoot:
        pushl   %ebp
        movl    %esp,%ebp
        pushl   InitialArg
        call    StartupPC
        call    InitialPC
        call    WhenDonePC

        # NOT REACHED
        movl    %ebp,%esp
        popl    %ebp
        ret
```

# 1.测试 Nachos 的基本内核

```
int main(int argc, char **argv)
    for (argc--, argv++; argc > 0; argc -= argCount, argv += argCount)
#endif // NETWORK
    }

    currentThread->Finish(); // NOTE: if the procedure "main"
                             // returns, then the program "nachos"
                             // will exit (as any other normal program
                             // would).  But there may be other
                             // threads on the ready list.  We switch
                             // to those threads by saying that the
                             // "main" thread is finished, preventing
                             // it from returning.
    return (0);              // Not reached...
}
```

```
yueqiu@YueQiu2003:~/nachos/code/threads$ ./nachos
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 130, idle 0, system 130, user 0
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
```

解释：`./nachos` 不带参数启动，`init` 初始化系统状态，创建主线程 `main`，之后进入 `ThreadTest()`

```
void ThreadTest()
{
    DEBUG('t', "Entering SimpleTest");

    Thread *t = new Thread("forked thread");

    t->Fork(SimpleThread, 1);
    SimpleThread(0);
}
```

ThreadTest 中，新建了一个线程，执行 SimpleThread(1)，而系统主线程执行 SimpleThread(0)

```
void SimpleThread(_int which)
{
    int num;

    for (num = 0; num < 5; num++)
    {
        printf("*** thread %d looped %d times\n", (int)which, num);
        currentThread->Yield();
    }
}
```

主线程运行 SimpleThread(0),循环输出信息、出让这两步，因为系统现在只有两个线程，出让后切换到 fork 的线程，执行 SimpleThread(1)，同上循环输出信息、出让这两步。都执行五次后下一个执行的是主线程 num=5 跳出循环，进而跳出 ThreadTest，回到 main()函数里，执行 currentThread->Finish()输出信息退出系统。

## 2. 测试 SynchTest()

```
yueqiu@YueQiu2003: ~/nachc
code  doc
yueqiu@YueQiu2003:~/nachos$ cd code/threads/
yueqiu@YueQiu2003:~/nachos/code/threads$ ./nachos
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
Direction [0], Car [0], Arriving...
Direction [0], Car [1], Arriving...
Direction [0], Car [2], Arriving...
Direction [0], Car [3], Arriving...
Direction [0], Car [4], Arriving...
Direction [0], Car [5], Arriving...
Direction [0], Car [6], Arriving...
Direction [0], Car [0], Crossing...
Direction [0], Car [1], Crossing...
Direction [0], Car [2], Crossing...
Direction [0], Car [0], Exiting...
Direction [0], Car [1], Exiting...
Direction [0], Car [2], Exiting...
Direction [1], Car [0], Arriving...
Direction [1], Car [1], Arriving...
Direction [1], Car [2], Arriving...
Direction [0], Car [3], Crossing...
```

## 3. 测试其他模块

```
yueqiu@YueQiu2003: ~/nachc

Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/threads$ cd ../filesys/
yueqiu@YueQiu2003:~/nachos/code/filesys$ ./nachos
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 1190, idle 1000, system 190, user 0
Disk I/O: reads 2, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

等

# 理解 1.系统启动与关机

启动时就是运行 main()函数，先 Initialize 系统，根据传入参数确定调试模式输出模式等，再初始化中断、线程调度器、定时器，然后创建主线程，再返回 main 运行。
关机也即主线程运行到 main()函数退出位置时，会先让主线程 finish，看看有无其他还需要运行的线程，没有的话就输出信息然后程序退出。

# 2.理解线程

```
currentThread = new Thread("main");
currentThread->setStatus(RUNNING);
```

如 1.所述，第一个线程“主线程”（main）是在系统初始化 Initialize 函数中创建的。
而其他线程是主线程运行过程中调用 fork 成员函数创建的。
线程调度，以线程测试中的出让为例，线程主动调用 Yield 函数出让，调度器寻找下一个就绪的线程执行，线程切换使用汇编指令直接切换寄存器等上下文实现。

# 回答问题 1.函数地址

如图，0x2f13 0x3141 0x2eec 和 0x4c4c
使用 gdb 直接 info address 得到

```
yueqiu@YueQiu2003: ~/nachc
yueqiu@YueQiu2003:~/nachos/code/threads$ gdb ./nachos
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.
html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./nachos...
(gdb) info address InterruptEnable
Symbol "InterruptEnable()" is a function at address 0x2f13.
(gdb) info address SimpleThread
Symbol "SimpleThread(int)" is a function at address 0x3141.
(gdb) info address ThreadFinish
Symbol "ThreadFinish()" is a function at address 0x2eec.
(gdb) info address ThreadRoot
Symbol "ThreadRoot" is at 0x4c4c in a file compiled without debugging.
(gdb)
```

## 2.对象地址

主线程地址在 0x56563ca0
fork 出来的线程地址在 0x56563d00

```
yueqiu@YueQiu2003: ~/nachc
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./nachos...
(gdb) info address InterruptEnable
Symbol "InterruptEnable()" is a function at address 0x2f13.
(gdb) info address SimpleThread
Symbol "SimpleThread(int)" is a function at address 0x3141.
(gdb) info address ThreadFinish
Symbol "ThreadFinish()" is a function at address 0x2eec.
(gdb) info address ThreadRoot
Symbol "ThreadRoot" is at 0x4c4c in a file compiled without debugging.
(gdb) b main.cc:92
Breakpoint 1 at 0x1421: file main.cc, line 92.
(gdb) r
Starting program: /home/yueqiu/nachos/code/threads/nachos
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1
".

Breakpoint 1, main (argc=1, argv=0xffffcca4) at main.cc:92
92              ThreadTest();
(gdb) s
ThreadTest () at threadtest.cc:43
43          DEBUG('t', "Entering SimpleTest");
(gdb) n
45          Thread *t = new Thread("forked thread");
(gdb) n
47          t->Fork(SimpleThread, 1);
(gdb) print currentThread
$1 = (Thread *) 0x56563ca0
(gdb)
```

```
yueqiu@YueQiu2003: ~/nachc
(gdb) n
*** thread 0 looped 1 times
31              currentThread->Yield();
(gdb) n
*** thread 1 looped 1 times
28          for (num = 0; num < 5; num++)
(gdb) print num
$4 = 1
(gdb) print which
$5 = 0
(gdb) n
30              printf("*** thread %d looped %d times\n", (int)which, nu
m);
(gdb) n
*** thread 0 looped 2 times
31              currentThread->Yield();
(gdb) s
Thread::Yield (this=0x56563ca0) at thread.cc:179
179         IntStatus oldLevel = interrupt->SetLevel(IntOff);
(gdb) n
181         ASSERT(this == currentThread);
(gdb) n
183         DEBUG('t', "Yielding thread \"%s\"\n", getName());
(gdb) n
185         nextThread = scheduler->FindNextToRun();
(gdb) n
186         if (nextThread != NULL)
(gdb) print nextThread
$6 = (Thread *) 0x56563d00
(gdb)
```

# 3. 返回地址

0x56559cb2

首先，c 代码里的 SWITCH 实际链接的程序可能会执行两个汇编函数，SWITCH 和 ThreadRoot，最后一个 ret 是汇编 SWITCH 里切换完寄存器后那个 ret。因为 ThreadRoot 里设置完 PC 后就直接回到 PC 位置的 c 代码里了没有 ret 指令。

使用 gdb 调试汇编，直接内存地址敲断点，查看栈顶值，或者查看运行到下一条汇编指令时的地址值。这个位置是汇编函数 ThreadRoot 的地址，接下来要初始化切换后线程的 PC 然后直接执行对应代码。

```
yueqiu@YueQiu2003: ~/nachc
   0x56559cad <+83>:    mov    0x5655e054,%eax
   0x56559cb2 <+88>:    ret
   0x56559cb3 <+89>:    xchg   %ax,%ax
--Type <RET> for more, q to quit, c to continue without paging--c
   0x56559cb5 <+91>:    xchg   %ax,%ax
   0x56559cb7 <+93>:    xchg   %ax,%ax
   0x56559cb9 <+95>:    xchg   %ax,%ax
   0x56559cbb <+97>:    xchg   %ax,%ax
   0x56559cbd <+99>:    xchg   %ax,%ax
   0x56559cbf <+101>:   nop
End of assembler dump.
(gdb) b *0x56559cb2
Breakpoint 2 at 0x56559cb2
(gdb) si
0x56559c5f in SWITCH ()
(gdb) c
Continuing.

Breakpoint 2, 0x56559cb2 in SWITCH ()
(gdb) si
0x56559c4c in ThreadRoot ()
(gdb) disassemble
Dump of assembler code for function ThreadRoot:
=> 0x56559c4c <+0>:     push   %ebp
   0x56559c4d <+1>:     mov    %esp,%ebp
   0x56559c4f <+3>:     push   %edx
   0x56559c50 <+4>:     call   *%ecx
   0x56559c52 <+6>:     call   *%esi
   0x56559c54 <+8>:     call   *%edi
   0x56559c56 <+10>:    mov    %ebp,%esp
```

## 4. 返回地址

0x565569cc

这个地址是 scheduler.cc 中调用 SWITCH 的位置，继续执行下一行打印 DEBUG 日志。

```
yueqiu@YueQiu2003: ~/nachc
--Type <RET> for more, q to quit, c to continue without paging--c
   0x56559cb5 <+91>:    xchg   %ax,%ax
   0x56559cb7 <+93>:    xchg   %ax,%ax
   0x56559cb9 <+95>:    xchg   %ax,%ax
   0x56559cbb <+97>:    xchg   %ax,%ax
   0x56559cbd <+99>:    xchg   %ax,%ax
   0x56559cbf <+101>:   nop
End of assembler dump.
(gdb) b *0x56559cb2
Breakpoint 3 at 0x56559cb2
(gdb) c
Continuing.

Breakpoint 3, 0x56559cb2 in SWITCH ()
(gdb) x $esp
0xffffcacc:     0x565569cc
(gdb) si
0x565569cc in Scheduler::Run (this=0x56563c80, nextThread=0x56563d00) at
 scheduler.cc:117
117         SWITCH(oldThread, nextThread);
(gdb) s
119         DEBUG('t', "Now in thread \"%s\"\n", currentThread->getName(
));
(gdb) print oldThread
$4 = (Thread *) 0x56563ca0
(gdb) print currentThread
$5 = (Thread *) 0x56563ca0
(gdb) print nextThread
$6 = (Thread *) 0x56563d00
(gdb)
```

## 额外问题 1.

直接在程序内打印地址，`printf %x` 等

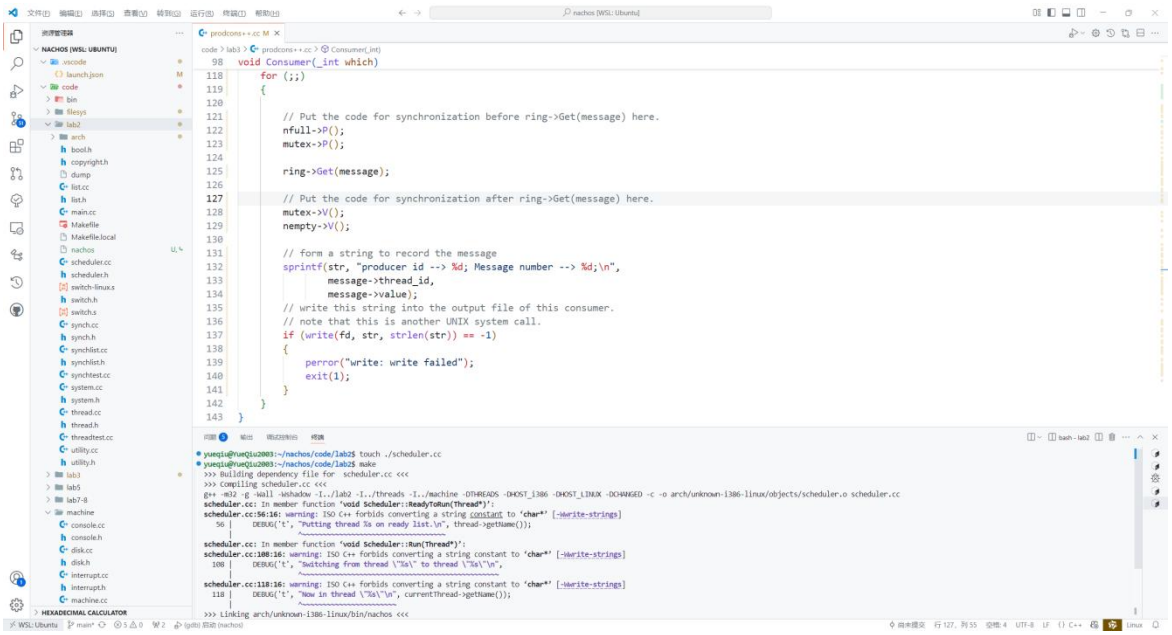
## 2.

上下文切换中不能被打断，这个操作必须是原子的。

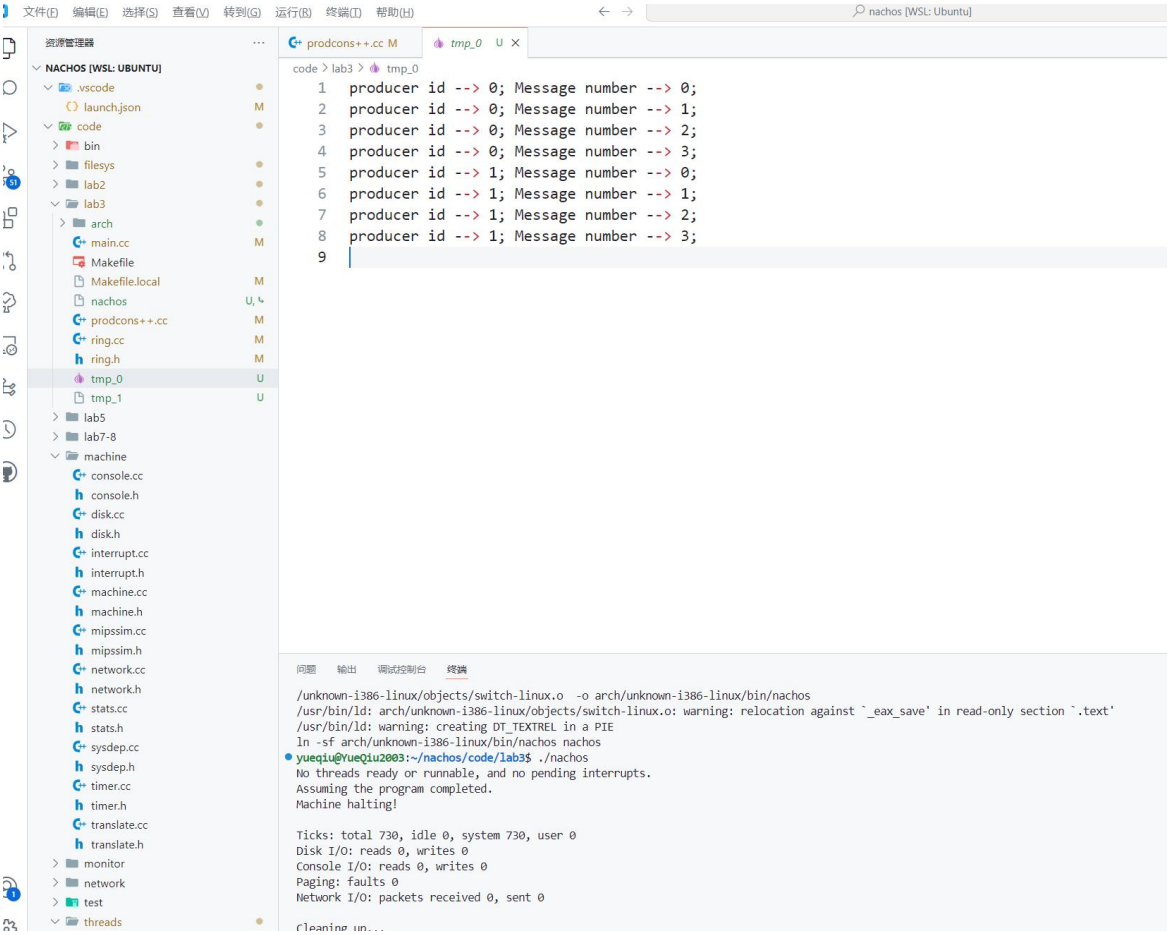
## 3.

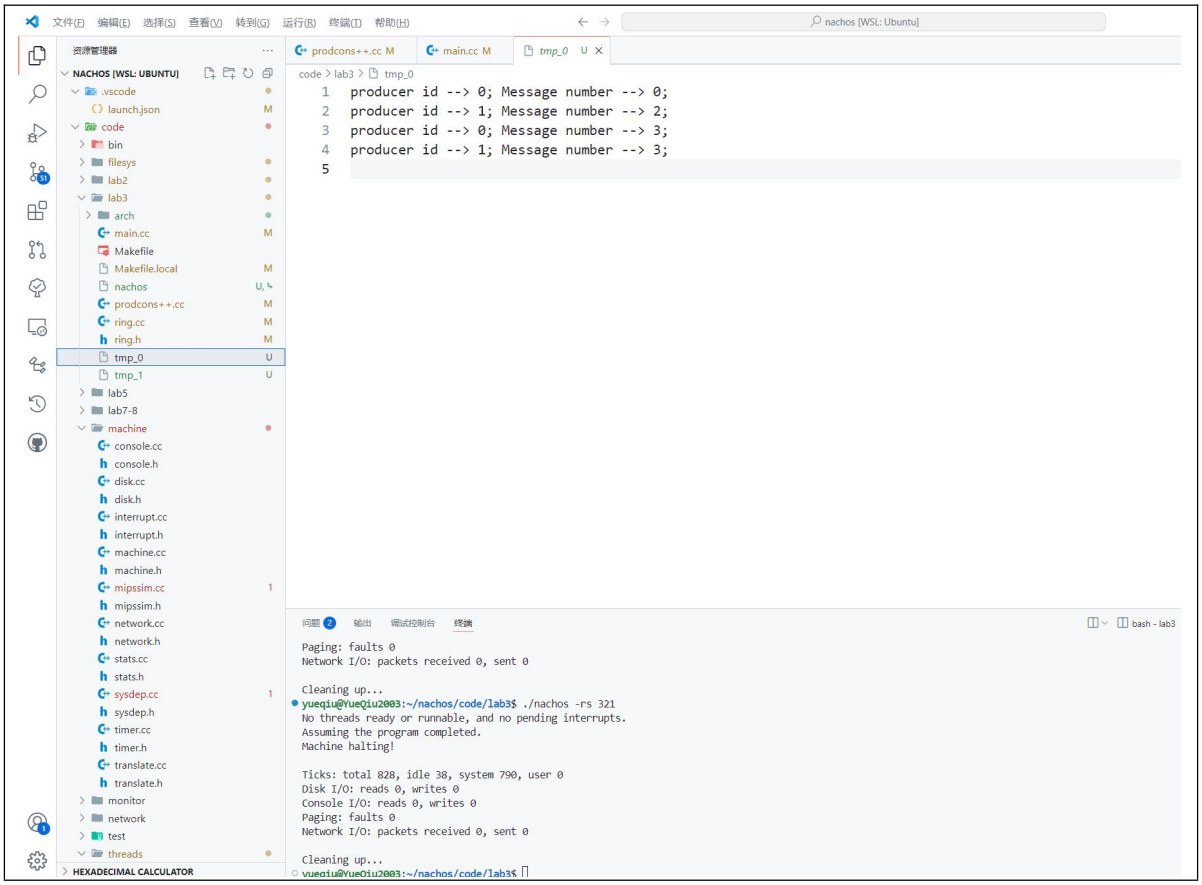
通过 `startupPC InitialPC WhenDonePC` 这三个函数来执行对应的代码。

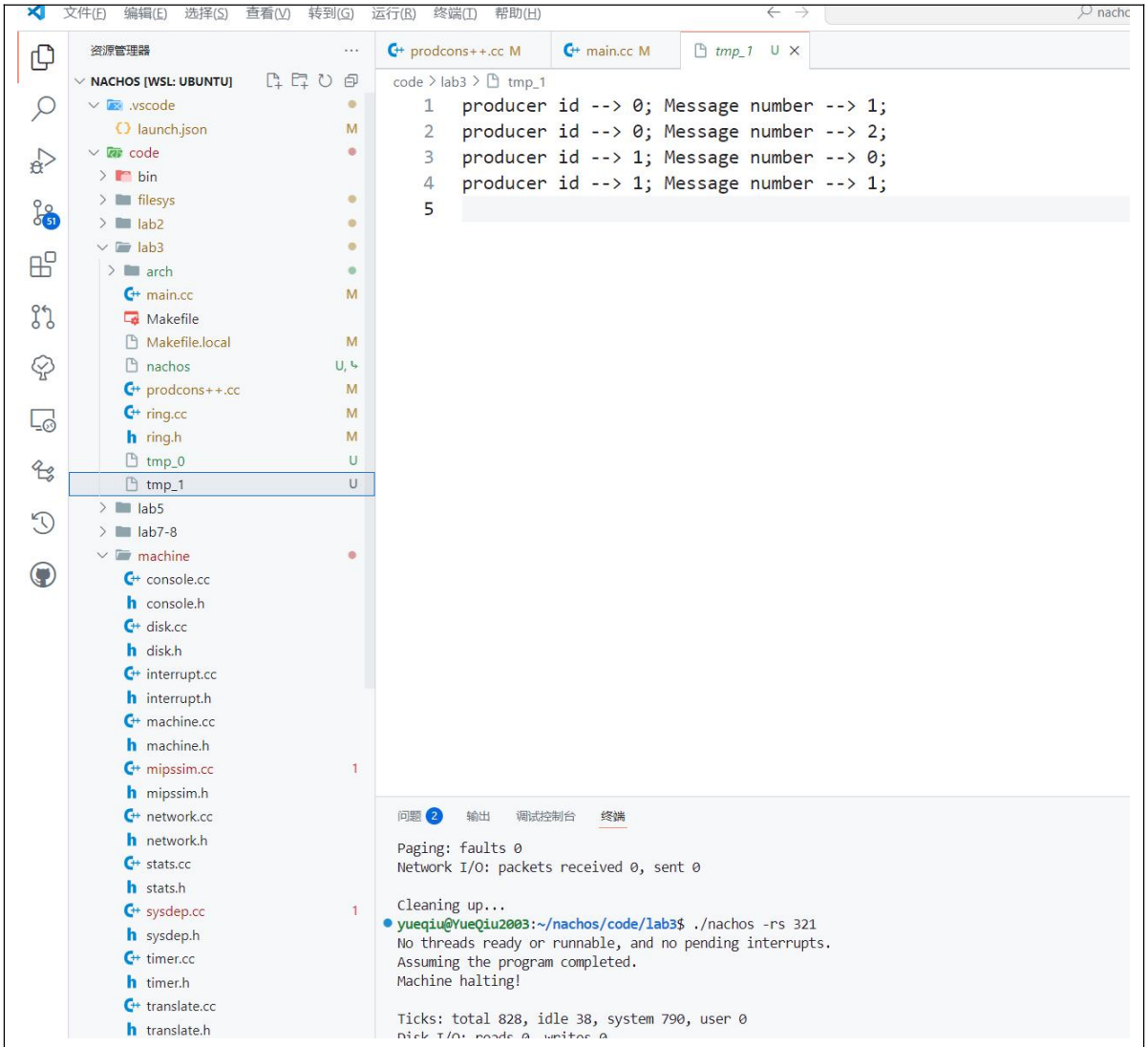
# 实验二


prodcons++.cc M
code > lab3 > prodcons++.cc > Consumer(_int)
void Consumer(_int which)
for (;;)
{
// Put the code for synchronization before ring->Get(message) here.
nfull->P();
mutex->P();
ring->Get(message);
// Put the code for synchronization after ring->Get(message) here.
mutex->V();
nempty->V();
// form a string to record the message
sprintf(str, "producer id --> %d; Message number --> %d;\n",
message->thread_id,
message->value);
// write this string into the output file of this consumer.
// note that this is another UNIX system call.
if (write(fd, str, strlen(str)) == -1)
{
perror("write: write failed");
exit(1);
}
}
}
yueqiu@YueQiu2003:~/nachos/code/lab2$ touch ./scheduler.cc
yueqiu@YueQiu2003:~/nachos/code/lab2$ make
>>> Building dependency file for scheduler.cc <<<
>>> Compiling scheduler.cc <<<
scheduler.cc: In member function 'void Scheduler::ReadyToRun(Thread*)':
scheduler.cc:56:16: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
DEBUG('t', "Putting thread %s on ready list.\n", thread->getName());
scheduler.cc: In member function 'void Scheduler::Run(Thread*)':
scheduler.cc:108:16: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
DEBUG('t', "Switching from thread \"%s\" to thread \"%s\"\n",
scheduler.cc:118:16: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
DEBUG('t', "Now in thread \"%s\"\n", currentThread->getName());
>>> Linking arch/unknown-i386-linux/bin/nachos <<<

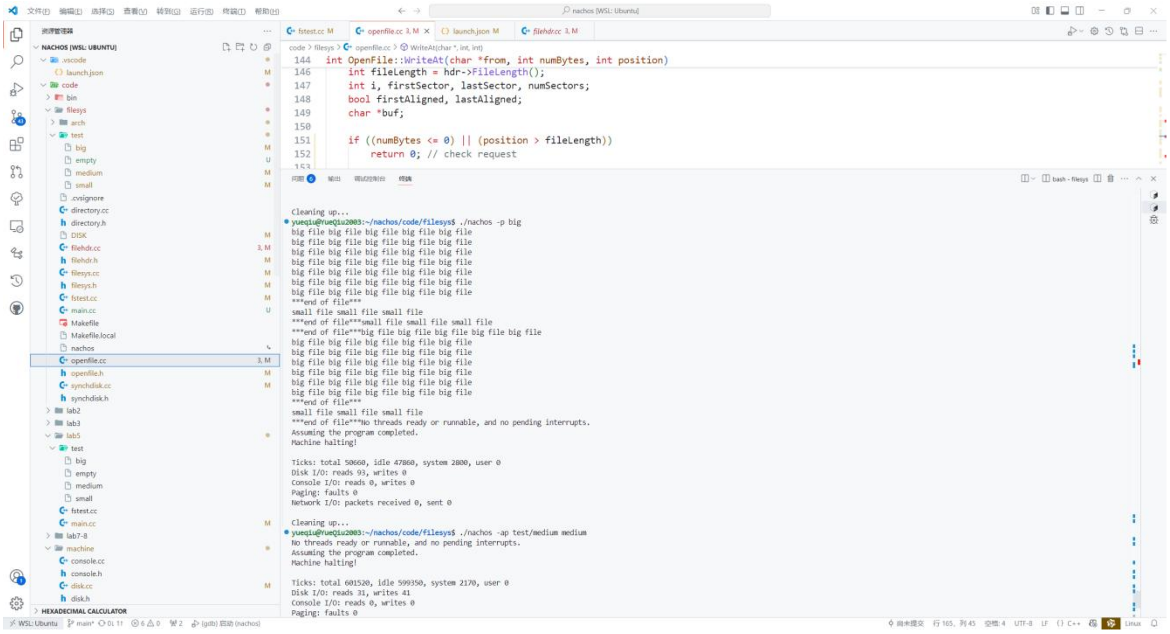
# 实验三


文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
nachos [WSL: Ubuntu]
prodcons++.cc M
tmp_0 U
code > lab3 > tmp_0
producer id --> 0; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 0; Message number --> 3;
producer id --> 1; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 1; Message number --> 2;
producer id --> 1; Message number --> 3;
问题 输出 调试控制台 终端
/unknown-i386-linux/objects/switch-linux.o -o arch/unknown-i386-linux/bin/nachos
/usr/bin/ld: arch/unknown-i386-linux/objects/switch-linux.o: warning: relocation against `_eax_save' in read-only section `.text'
/usr/bin/ld: warning: creating DT_TEXTREL in a PIE
ln -sf arch/unknown-i386-linux/bin/nachos nachos
yueqiu@YueQiu2003:~/nachos/code/lab3$ ./nachos
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
Ticks: total 730, idle 0, system 730, user 0
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...


文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
nachos [WSL: Ubuntu]
资源管理器
NACHOS [WSL: UBUNTU]
.vscode
launch.json
code
bin
filesys
lab2
lab3
arch
main.cc
Makefile
Makefile.local
nachos
prodcons++.cc
ring.cc
ring.h
tmp_0
tmp_1
lab5
lab7-8
machine
console.cc
console.h
disk.cc
disk.h
interrupt.cc
interrupt.h
machine.cc
machine.h
mipssim.cc
mipssim.h
network.cc
network.h
stats.cc
stats.h
sysdep.cc
sysdep.h
timer.cc
timer.h
translate.cc
translate.h
monitor
network
test
threads
HEXADECIMAL CALCULATOR
prodcons++.cc M
main.cc M
tmp_0 U
code > lab3 > tmp_0
producer id --> 0; Message number --> 0;
producer id --> 1; Message number --> 2;
producer id --> 0; Message number --> 3;
producer id --> 1; Message number --> 3;
问题 2 输出 调试控制台 终端
bash - lab3
Paging: faults 0
Network I/O: packets received 0, sent 0
Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/lab3$ ./nachos -rs 321
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
Ticks: total 828, idle 38, system 790, user 0
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...
yueqiu@YueQiu2003:~/nachos/code/lab3$

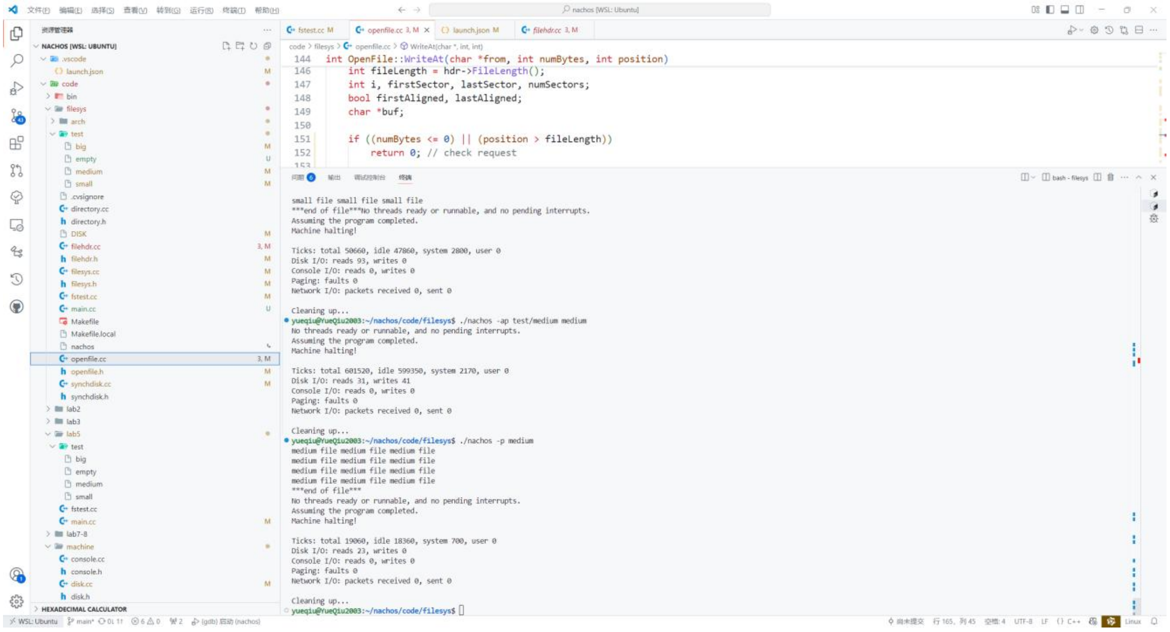

文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
资源管理器
NACHOS [WSL: UBUNTU]
.vscode
launch.json
code
bin
filesys
lab2
lab3
arch
main.cc
Makefile
Makefile.local
nachos
prodcons++.cc
ring.cc
ring.h
tmp_0
tmp_1
lab5
lab7-8
machine
console.cc
console.h
disk.cc
disk.h
interrupt.cc
interrupt.h
machine.cc
machine.h
mipssim.cc
mipssim.h
network.cc
network.h
stats.cc
stats.h
sysdep.cc
sysdep.h
timer.cc
timer.h
translate.cc
translate.h
prodcons++.cc M
main.cc M
tmp_1 U
code > lab3 > tmp_1
producer id --> 0; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 1; Message number --> 0;
producer id --> 1; Message number --> 1;
问题 2 输出 调试控制台 终端
Paging: faults 0
Network I/O: packets received 0, sent 0
Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/lab3$ ./nachos -rs 321
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
Ticks: total 828, idle 38, system 790, user 0

# 实验四、五



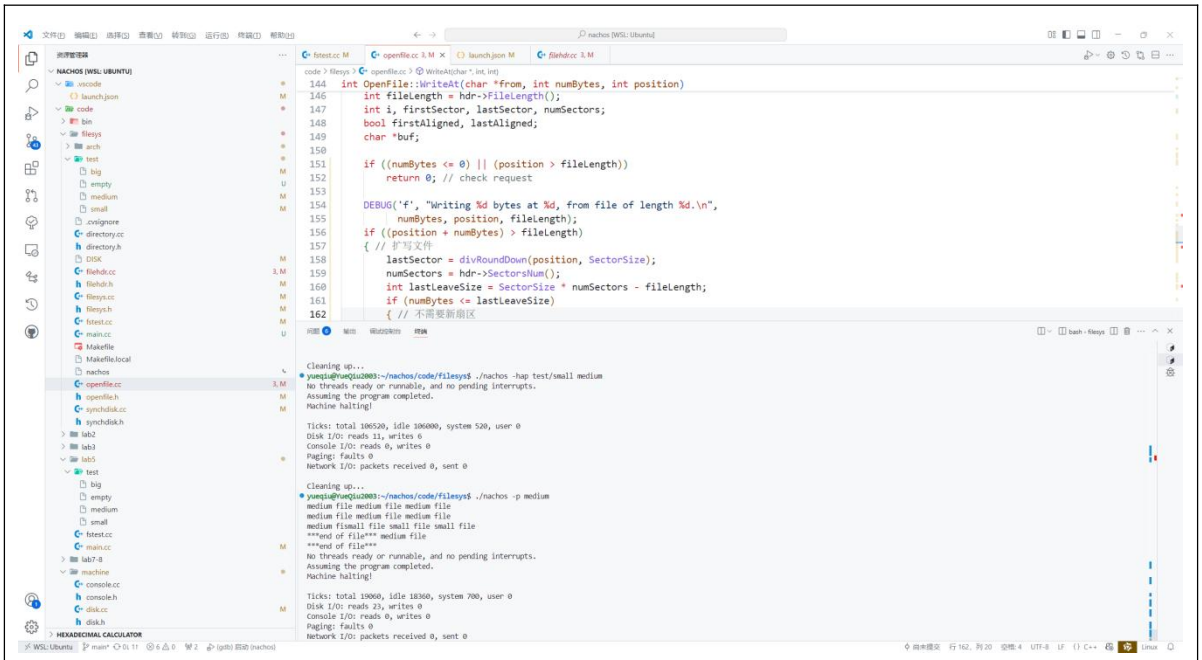
```cpp
int OpenFile::WriteAt(char *from, int numBytes, int position)
    int fileLength = hdr->FileLength();
    int i, firstSector, lastSector, numSectors;
    bool firstAligned, lastAligned;
    char *buf;

    if ((numBytes <= 0) || (position > fileLength))
        return 0; // check request
```

```
Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/filesys$ ./nachos -p big
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
***end of file***
small file small file small file
***end of file***small file small file small file
***end of file***big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
big file big file big file big file big file
***end of file***
small file small file small file
***end of file***No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 50660, idle 47860, system 2800, user 0
Disk I/O: reads 93, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/filesys$ ./nachos -ap test/medium medium
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 601520, idle 599350, system 2170, user 0
Disk I/O: reads 31, writes 41
Console I/O: reads 0, writes 0
Paging: faults 0
```

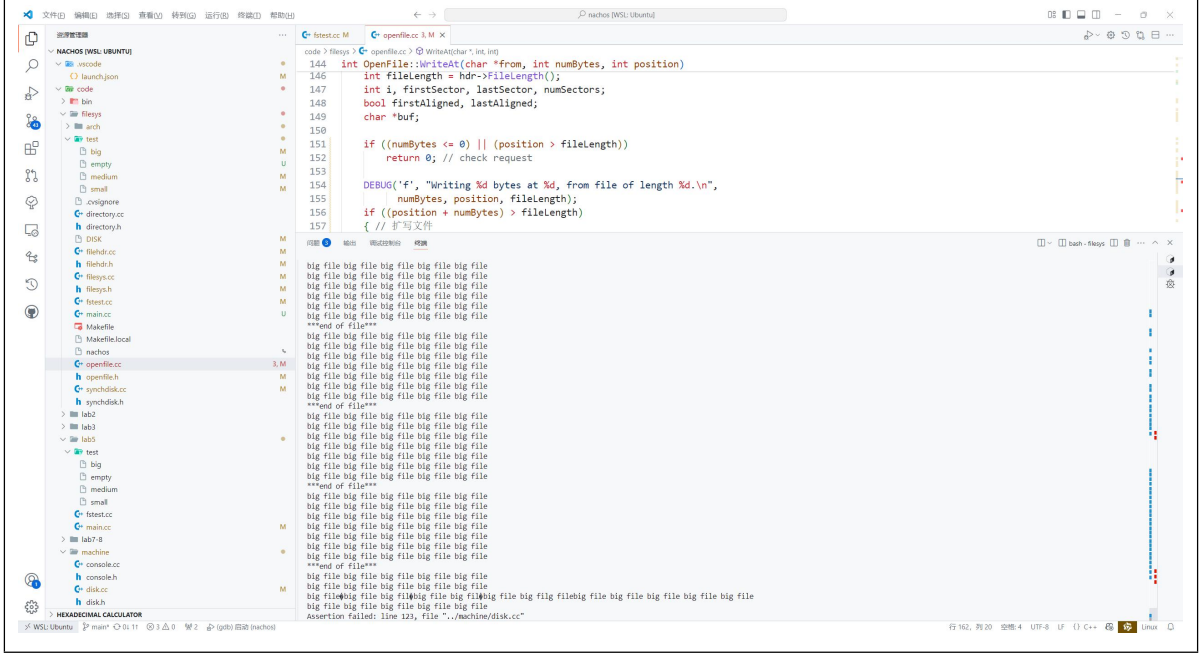


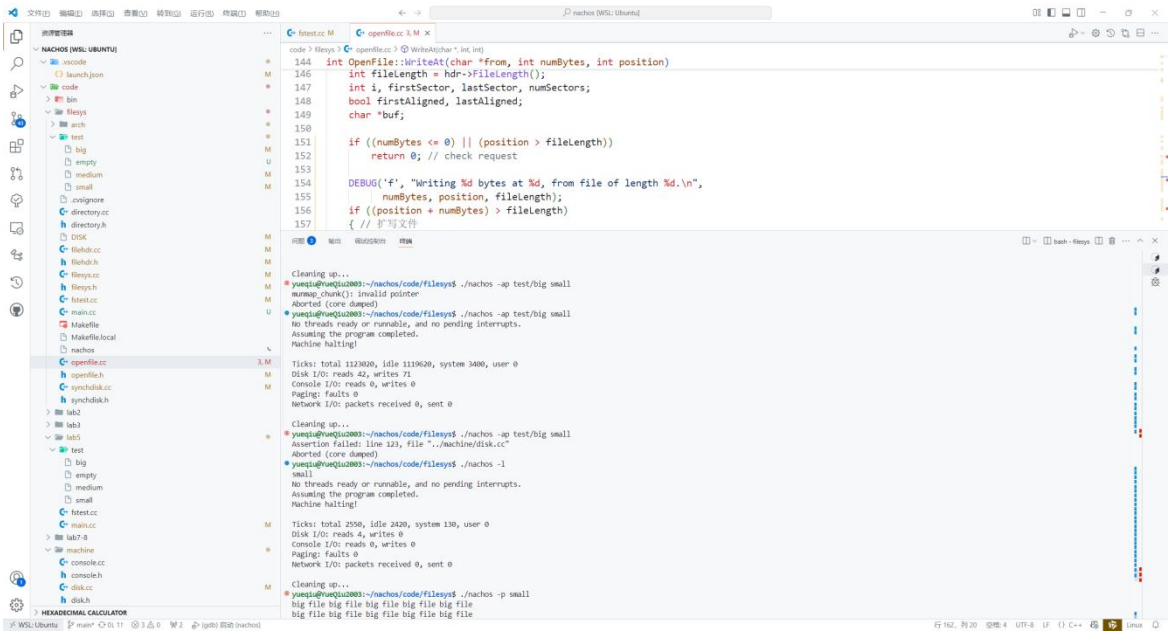
```cpp
int OpenFile::WriteAt(char *from, int numBytes, int position)
    int fileLength = hdr->FileLength();
    int i, firstSector, lastSector, numSectors;
    bool firstAligned, lastAligned;
    char *buf;

    if ((numBytes <= 0) || (position > fileLength))
        return 0; // check request
```

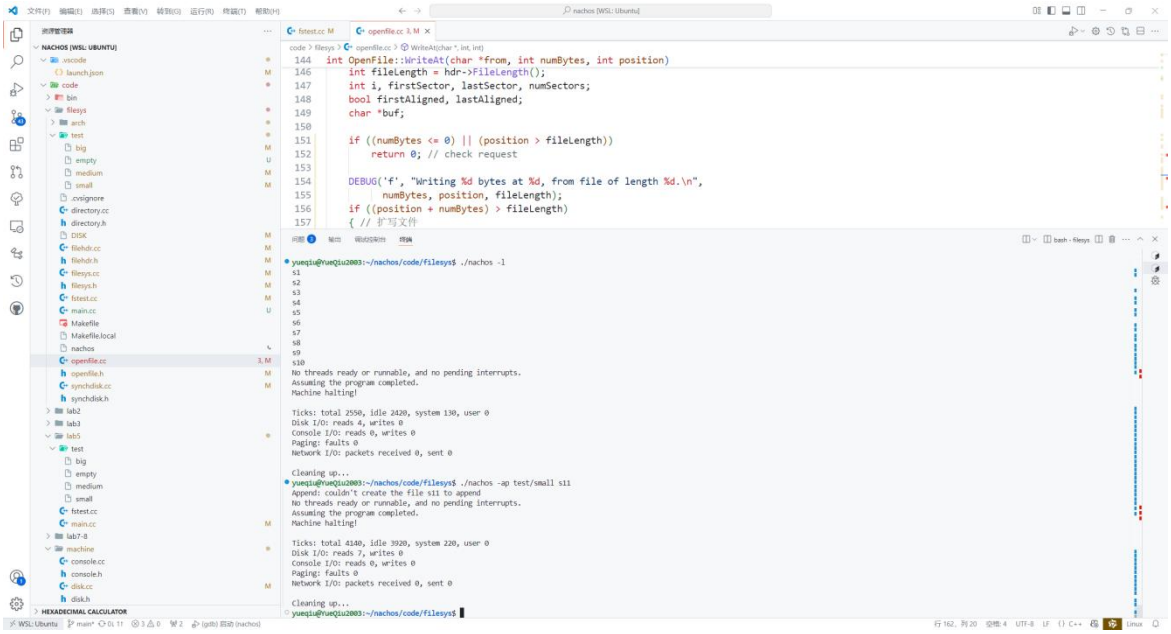
```
small file small file small file
***end of file***No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 50660, idle 47860, system 2800, user 0
Disk I/O: reads 93, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/filesys$ ./nachos -ap test/medium medium
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 601520, idle 599350, system 2170, user 0
Disk I/O: reads 31, writes 41
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/filesys$ ./nachos -p medium
medium file medium file medium file
medium file medium file medium file
medium file medium file medium file
medium file medium file medium file
***end of file***
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 19060, idle 18360, system 700, user 0
Disk I/O: reads 23, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
yueqiu@YueQiu2003:~/nachos/code/filesys$ 
```




# 第十五次文件系统暴毙

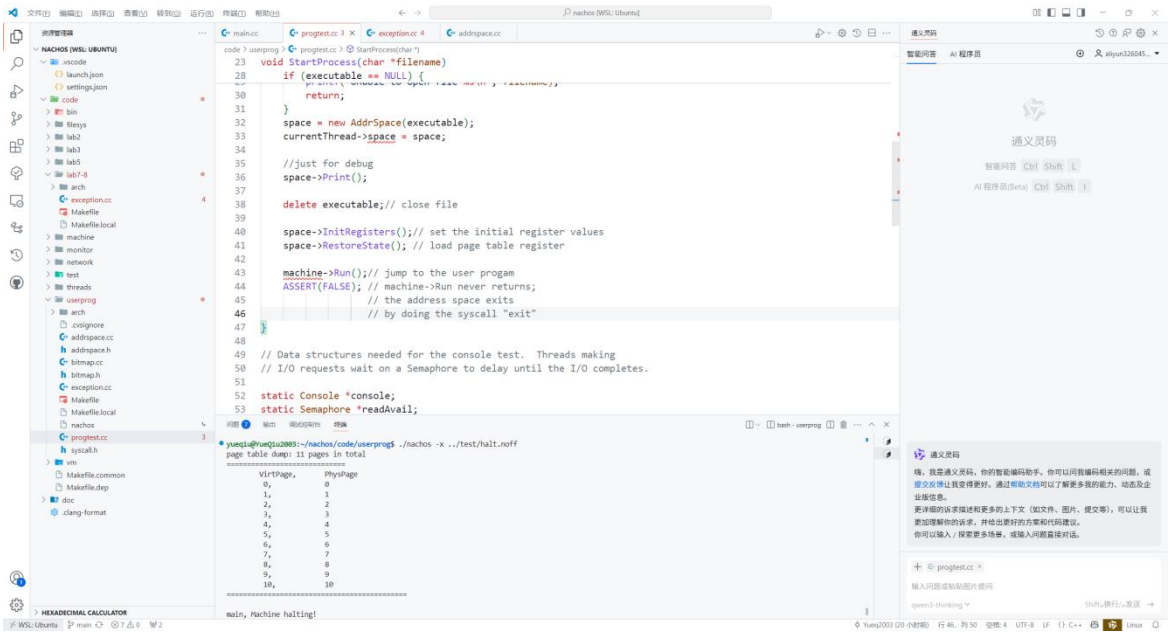

int OpenFile::WriteAt(char *from, int numBytes, int position)
int fileLength = hdr->FileLength();
int i, firstSector, lastSector, numSectors;
bool firstAligned, lastAligned;
char *buf;
if ((numBytes <= 0) || (position > fileLength))
return 0; // check request
DEBUG('f', "Writing %d bytes at %d, from file of length %d.\n",
numBytes, position, fileLength);
if ((position + numBytes) > fileLength)
{ // 扩写文件
big file big file big file big file big file big file
***end of file***
big file big file big file big file big file big file
***end of file***
big file big file big file big file big file big file
***end of file***
big file big file big file big file big file big file
***end of file***
big file big file big file big file big file big file
***end of file***
big file big file big file big file big file big file
big file big file big file big file big file big file
big file big file big file big file big file big file




# 实验六

2. 通过直接调用 start.s 内提供的系统调用函数来交互。
5. 先打开用户程序文件，根据用户程序创建内存空间，直接写入（模拟机的）主存，再把页表、寄存器交给模拟机，开始执行。




（3）用户程序是通过主线程运行的模拟机器来执行的。

```
    Instruction *instr = new Instruction;// storage for decoded instru

    if (DebugIsEnabled('m'))
        printf("Starting thread \"%s\" at time %d\n",
                currentThread->getName(), stats->totalTicks);
    interrupt->setStatus(UserMode);

    currentThread->Print();

    for (;;) {
        OneInstruction(instr);
        interrupt->OneTick();
        if (singleStep && (runUntilTime <= stats->totalTicks))
            Debugger();
    }
}


//----------------------------------------------------------------------
// TypeToReg
//  Retrieve the register # referred to in an instruction.
```

```
问题 4   输出   调试控制台   终端

        5,          5
        6,          6
        7,          7
        8,          8
        9,          9
        10,         10
==========================================

main, Machine halting!

Ticks: total 37, idle 0, system 10, user 27
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
```

# 实验七、八

关于 pid，在 AddrSpace 类中加一个静态 set<SpaceId> pids 保存所有存活用户线程的 pid，space 释放时 erase 该 pid，分配时使用全局保存的 globalpid++遍历可用 pid。
修改可执行文件内存写入，现在是依次遍历分配内存页写入数据。

```
int reCodeSize = noffH.code.size;
int reIdataSize = noffH.initData.size;
int reUdataSize = noffH.uninitData.size + UserStackSize;
i = 0;
int currentSize = PageSize;
while (reCodeSize || reIdataSize || reUdataSize)
{
    int phyPos = pageTable[i].physicalPage * PageSize + PageSize - currentSize;
    if (reCodeSize)
    {
        int writedSize = noffH.code.size - reCodeSize;
        if (currentSize > reCodeSize)
        {
            executable->ReadAt(&(machine->mainMemory[phyPos]),
                               reCodeSize, noffH.code.inFileAddr + writedSize);
            currentSize -= reCodeSize;
            reCodeSize = 0;
        } else
        {
            executable->ReadAt(&(machine->mainMemory[phyPos]),
                               currentSize, noffH.code.inFileAddr + writedSize);
            reCodeSize -= currentSize;
            i++;
            currentSize = PageSize;
        }
    } else if (reIdataSize)
    {
        int writedSize = noffH.initData.size - reIdataSize;
```

```
AddrSpace::AddrSpace(OpenFile *executable)
    while (reCodeSize || reIdataSize || reUdataSize)
        if (reCodeSize)
            } else
                reCodeSize -= currentSize;
                i++;
                currentSize = PageSize;
            }
        } else if (reIdataSize)
        {
            int writedSize = noffH.initData.size - reIdataSize;
            if (currentSize > reIdataSize)
            {
                executable->ReadAt(&(machine->mainMemory[phyPos]),
                                   reIdataSize, noffH.initData.inFileAddr + writedSize);
                currentSize -= reIdataSize;
                reIdataSize = 0;
            } else
            {
                executable->ReadAt(&(machine->mainMemory[phyPos]),
                                   currentSize, noffH.initData.inFileAddr + writedSize);
                reIdataSize -= currentSize;
                i++;
                currentSize = PageSize;
            }
        } else if (reUdataSize > 0)
        {
            bzero(&(machine->mainMemory[phyPos]), currentSize);
            i++;
            currentSize = PageSize;
            reUdataSize -= currentSize;
```

问题 39　输出　调试控制台　终端

实现 Exec()

于(R)　终端(T)　帮助(H)　nachos [WSL: Ubuntu]

progtest.cc M　mipssim.cc　exception.cc M　addrspace.cc 9+, M　halt.c M　exec.c 1, U

code > test > exec.c > main()

```
#include "syscall.h"

int main()
{
    SpaceId pid;
    Print("exec.c running 0");
    pid = Exec("../test/halt.noff");
    // Print("exec.c running 1");
    // Yield();
    Exit(0);
}
```

```
/* halt.c
 *  NOTE: for some reason, user programs with global data structures
 *  sometimes haven't worked in the Nachos environment.  So be careful
 *  out there!  One option is to allocate data structures as
 *  automatics within a procedure, but if you do this, you have to
 *  be careful to allocate a big enough stack to hold the automatics!
 */

#include "syscall.h"

int main()
{
    // char prompt[2];
    // prompt[0] = '-';
    // prompt[1] = '-';

    // Write(prompt, 1, "I will shut down!\n");
    Print("halt.c running 0");
    // Yield();
    Print("halt.c running 1");
    Print("halt.c running 2");
    Print("halt.c running 3");
    Print("halt.c running 4");
    Print("halt.c running 5");
    Print("halt.c running 6");
    Print("halt.c running 7");
    // Yield();
    Halt();
    /* not reached */
}
```

```
用户程序101输出:halt.c running 7
Machine halting!

Ticks: total 118, idle 0, system 30, user 88
```

(为调试输出了 space）

问题 20　输出　调试控制台　终端

```
/usr/bin/ld: warning: creating DT_TEXTREL in a PIE
ln -sf arch/unknown-i386-linux/bin/nachos nachos
yueqiu@YueQiu2003:~/nachos/code/userprog$ ./nachos -x ../test/exec.noff
space 100:page table dump: 11 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
===========================================

用户程序100输出:exec.c running 0
space 101:page table dump: 13 pages in total
============================
        VirtPage,       PhysPage
         0,             11
         1,             12
         2,             13
         3,             14
```

```
AddrSpace::AddrSpace(OpenFile *executable)
    while (reCodeSize || reIdataSize || reUdataSize)
        if (reCodeSize)
            if (currentSize > reCodeSize)
                currentSize -= reCodeSize;
                reCodeSize = 0;
            } else
            {
                executable->ReadAt(&(machine->mainMemory[phyPos]),
                                   currentSize, noffH.code.inFileAddr + writedSize);
                reCodeSize -= currentSize;
                i++;
                currentSize = PageSize;
            }
        } else if (reIdataSize)
        {
            int writedSize = noffH.initData.size - reIdataSize;
```

```
        2,          13
        3,          14
        4,          15
        5,          16
        6,          17
        7,          18
        8,          19
        9,          20
        10,         21
        11,         22
        12,         23
===========================

用户程序101输出:halt.c running 0
用户程序101输出:halt.c running 1
用户程序101输出:halt.c running 2
用户程序101输出:halt.c running 3
用户程序101输出:halt.c running 4
用户程序101输出:halt.c running 5
用户程序101输出:halt.c running 6
用户程序101输出:halt.c running 7
Machine halting!

Ticks: total 118, idle 0, system 30, user 88
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...
yueqiu@YueQiu2003:~/nachos/code/userprog$
```

```
问题 26    输出    调试控制台    终端

/usr/bin/ld: warning: creating DT_TEXTREL in a PIE
ln -sf arch/unknown-i386-linux/bin/nachos nachos
yueqiu@YueQiu2003:~/nachos/code/userprog$ ./nachos -x ../test/exec.noff
用户程序100输出:exec.c running 0
用户程序101输出:halt.c running 0
用户程序101输出:halt.c running 1
用户程序101输出:halt.c running 2
用户程序101输出:halt.c running 3
用户程序101输出:halt.c running 4
用户程序101输出:halt.c running 5
用户程序101输出:halt.c running 6
用户程序101输出:halt.c running 7
Machine halting!

Ticks: total 118, idle 0, system 30, user 88
Disk I/O: reads 0, writes 0
```

实现 Exit()和 Yield()

```
终端(T)  帮助(H)                                  nachos [WSL: Ubuntu]
progtest.cc 6, M   exception.cc 9+, M   addrspace.cc 9+, M   halt.c M ×   exec.c 1, U
code > test > halt.c > main()
/* halt.c
 *  out there!  One option is to allocate data structures as
 *  automatics within a procedure, but if you do this, you have to
 *  be careful to allocate a big enough stack to hold the automatics!
 */

#include "syscall.h"

int main()
{
    Print("halt.c running 0");
    Yield();
    Print("halt.c running 1");
    Yield();
    Print("halt.c running 2");
    Halt();
}
```

(R) 终端(T) 帮助(H) nachos [WSL: Ubuntu]

progtest.cc 6, M | exception.cc 9+, M | addrspace.cc 9+, M | halt.c M | exec.c 1, U

code > test > exec.c > main()

```c
#include "syscall.h"

int main()
{
    SpaceId pid;
    Print("exec.c running 0");
    pid = Exec("../test/halt.noff");
    Print("exec.c running 1");
    Yield();
    Exit(0);
}
```

```
/usr/bin/ld: warning: creating DT_TEXTREL in a PIE
ln -sf arch/unknown-i386-linux/bin/nachos nachos
yueqiu@YueQiu2003:~/nachos/code/userprog$ ./nachos -x ../test/exec.noff
用户程序100输出:exec.c running 0
用户程序101输出:halt.c running 0
用户程序100输出:exec.c running 1
用户程序101输出:halt.c running 1
用户程序100Exit with code 0
用户程序101输出:halt.c running 2
Machine halting!

Ticks: total 155, idle 0, system 70, user 85
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...
yueqiu@YueQiu2003:~/nachos/code/userprog$
```

问题及收获：

系统级编程 debug 比较麻烦，一是内核级涉及汇编有些代码需要汇编级 debug，比如 SWITCH()，汇编直接操作寄存器和内存，需要自己梳理逻辑。再就是用户级更麻烦，因为是虚拟机模拟的用户程序执行，根本没有 gdb，用户程序出问题只能抛异常再手动一个个看内存、寄存器。

收获比较多，搞懂了内存、磁盘等的管理，程序的执行、线程之间的关系等。