CIS 657 PA-4

System Calls and Virtual Memory System

***Instructor**: Professor Endadul Hoque*

***Students**:*

*Leah Luo (**NetId**: lluojr / **SUID**:* 326896495*)*

*Bo Li (**NetId**: bli158 /* **SUID**: 218717316*)*

***Date**: 04/19/2020*

Content

CIS 657
Leah Luo (SUID: 326896495, NetID: lluojr)
Bo Li (SUID: 218717316 , NetID: bli158)
PA-4
04/18/2020

List of Files Modified

1. **machine/ directory**
   a. timer.cc
   b. translate.cc
2. **threads/ directory**
   a. alarm.cc
   b. kernel.h & kernel.cc
   c. main.cc
3. **userprog/ directory**
   a. addrspace.cc
   b. exception.cc
   c. ksyscall.h

Task 1 Solution

1. **Implement Read and Write for Console**

a. int Read (char *buffer, int size, OpenFileId id)

(i) execption.cc

Explanation

Initialize variables buffer, size, and enterIn (type OpenFileId) from registers 4, 5, 6.
Call SysRead function in ksyscall.h, which returns the number of bytes actually read.
Store the returned value into register 2.
Modify return pointer. Specifically, set the previous program counter, set the program counter to next instruction, set the next program counter for branch execution.
Why PCReg+4: all instructions are 4 bytes wide.

```
case SC_Read:
{
    DEBUG(dbgSys, "read from buffer" << kernel->machine->ReadRegister(4)
        << kernel->machine->ReadRegister(5)<< kernel->machine->ReadRegister(6) << "\n");

    int buffer = (int)kernel->machine->ReadRegister(4);
    int size = (int)kernel->machine->ReadRegister(5);//register 5 save the address of next process
    OpenFileId enterIn = (OpenFileId)kernel->machine->ReadRegister(6);//The id of the file that will be

    // Result: bytes actually read
    int result = SysRead(buffer, size, enterIn);

    DEBUG(dbgSys, "SysRead returning with " << result << "\n");
    // Store the result into register 2
    kernel->machine->WriteRegister(2, result);

    /* Modify return point */
    {
        kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
        kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
        kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
    }

    return;
    ASSERTNOTREACHED();
}
break;
```

(ii) ksyscall.h

Explanation

Check if id is CONSOLEINPUT (id = 0) first, as stdin.

Create a counter to count bytes actually read (it does not always equal to the number of bytes required)

Read from stdin byte by byte into buffer

Return the number of bytes actually read

```
int SysRead(int buffer, int size, OpenFileId id)
{
    // Check if id is CONSOLEINPUT (OpenFileId id 0 is “stdin)
    if (id != 0) return -1;

    char ch;
    // Counter to count bytes actually read
    int num = 0;

    // Read size bytes from the open file into buffer,
    do {
        ch = kernel->synchConsoleIn->GetChar();
        kernel->machine->WriteMem(buffer+num, 1, (int)ch);
        num += 1;
    } while (num < size && ch != EOF);

    // Return bytes acutally read
    return num;
}
```

b. int Write(char *buffer, int size, OpenFileId id)

(i) execption.cc

Explanation

Initialize variables buffer, size, and enterOut (type OpenFileId) from registers 4, 5, 6.

Call SysWrite function in ksyscall.h, which returns the number of bytes successfully written.

Store the returned value into register 2.

Modify return pointer. Specifically, set the previous program counter, set the program counter to next instruction, set the next program counter for branch execution.

Why PCReg+4: all instructions are 4 bytes wide.

```
case SC_Write:
{
    DEBUG(dbgSys, "write to buffer" << kernel->machine->ReadRegister(4)
        << kernel->machine->ReadRegister(5)<< kernel->machine->ReadRegister(6) << "\n");

    int buffer = (int) kernel->machine->ReadRegister(4);
    int size = (int) kernel->machine->ReadRegister(5);
    OpenFileId enterOut = (OpenFileId)kernel->machine->ReadRegister(6); //The id of the file t

    // Result: bytes successfully written
    int result = SysWrite(buffer, size, enterOut);

    DEBUG(dbgSys, "Add returning with " << result << "\n");
    /* Prepare Result */
    kernel->machine->WriteRegister(2, result);

    /* Modify return point */
    {
        kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
        kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
        kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
    }
    return;
    ASSERTNOTREACHED();
}
break;
```

(ii) ksyscall.h

Explanation

Check if id is CONSOLEOUTPUT (id = 1) first, as stdout
Create a counter to count bytes successfully written (it does not always equal to the number of bytes required)
Write size bytes from buffer to the open file (byte by byte)
Return the number of bytes successfully written

```
int SysWrite(int buffer, int size, OpenFileId id ) {
    // Check if id is CONSOLEOUTPUT (OpenFileId id 1 is "stdout)
    if (id != 1) return -1;
    // Counter to count bytes successfully written
    int num = 0;

    while (num < size) {
        int current;
        // Write size bytes from buffer to the open file
        kernel->machine->ReadMem(buffer+num, 1, &current);
        // Perform console I/O
        kernel->synchConsoleOut->PutChar( (char) current );
        num += 1;
    }
    // Return bytes successfully written
    return num;
}
```

c. Testing Result

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Read
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 12] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 4]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 5]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 6]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 7]
>>hello
PASS!!  Reading is hello

status: 0
```

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Write
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 4]
PASS!!status: 0
^C
Cleaning up after signal 2
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$
```

2. Implement Fork system call

a. addrspace.cc

Explanation

Complete the copy constructor of the address space at addrspace.cc under userprog/ directory.
Copy numPages from copiedItem to current.
For the current, it's required to create new page table.
Copy all variables of page table from copiedItem to current page table one by one in the loop.

```
//your code goes here to add something into the copy constructor//
AddrSpace::AddrSpace(const AddrSpace& copiedItem) { // copy constructor     // Task 1
    if (copiedItem.numPages <= 0) return;   // Addr Space does not exist

    numPages = copiedItem.numPages;

    pageTable = new TranslationEntry[numPages];

    for (int i = 0; i < numPages; i++) {
        pageTable[i].virtualPage = copiedItem.pageTable[i].virtualPage;
        pageTable[i].physicalPage = copiedItem.pageTable[i].physicalPage;
        pageTable[i].valid = copiedItem.pageTable[i].valid;
        pageTable[i].readOnly = copiedItem.pageTable[i].readOnly;
        pageTable[i].use = copiedItem.pageTable[i].use;
        pageTable[i].dirty = copiedItem.pageTable[i].dirty;
    }
}
```

b. execption.cc

Explanation

Call the SysFork function in ksyscall.h, return the pid of the parent thread and store it into register 2.

```
case SC_SysFork:
{
    // Call SysFork function, return the pid of the parent thread
    ThreadId rc = SysFork();
    // Store the returned value into register 2
    kernel->machine->WriteRegister(2, (int)rc);    // Child Thread
    /* Modify return point */
    {
        kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg) );
        kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
        kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    }
    return;
    ASSERTNOTREACHED();
}
break;
```

c. ksyscall.h

Explanation

Create a new child thread, and invoke copy constructor to give it a new address space
Store the pid of the new child thread into register 2
Conduct a context-switch between child thread and parent thread. Save the state of parent thread by calling SaveUserstate() first
Since a new thread is created and executed, the pointer to next instruction should be modified. Get the pointer to the current register first, add 4 bytes to it (instruction is 4 bytes wide), and set to pointing to the current register by calling setUserRegister() function.
Call Fork function, invoke restoreFunction() which execute concurrently. restoreFunction() restore the state of the parent thread and run it.

```
ThreadId SysFork() {
    Thread *thread = new Thread("ThreadFork");
    thread->space = new AddrSpace(*(kernel->currentThread->space)); // Invoke Copy Constructor

    kernel->machine->WriteRegister(2, (int)0);    //Save child-thread pid to Reg 2
    thread->SaveUserState();     // Kernal -> User
    int* n = thread->getUserRegisters();        // GetUserReg

    // Add 4
    n[PCReg] = kernel->machine->ReadRegister(PCReg) + 4;
    n[NextPCReg] = kernel->machine->ReadRegister(NextPCReg) + 4;

    // put it back
    thread->setUserRegister(n);
    thread->Fork((VoidFunctionPtr)restoreFunction,(void*)0);  // User -> Kernal

    return (ThreadId)thread->pid;
}
```

```
void restoreFunction() {
    // Restore the state of thread and run it
    kernel->currentThread->RestoreUserState();
    kernel->currentThread->space->RestoreState();  // swap into machine
    kernel->machine->Run();
}
```

d. Testing Result

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Fork
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 4]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 5]
L[pid 2]: [virtualPage: 1] -> [physicalPage: 6]
cL[pid 1]: [virtualPage: 1] -> [physicalPage: 7]
hild processstatus: 0
parent processstatus: 0
^C
Cleaning up after signal 2
```

3. Implement Exec system call

a. execption.cc

Explanation

The total number of registers is 40, referring to NumTotalRegs in machine.h. The reason to set size as 40 is because it's showing error if we assigned it more than 40, also, if assign it less than 6, the error also occurs because "pass!!" has 6 bytes when executing Exec, since Exec calls write. Therefore, 40 is a bound size to be set as safety.

Read the size bytes from file into buffer.

Call SysExec in ksyscall.h and return the spaceId of the new thread, store it into register 2.

```
case SC_Exec:
{
    DEBUG(dbgSys, "Exec" << kernel->machine->ReadRegister(4) << "\n");

    int size = 40;

    int addr = (int)kernel->machine->ReadRegister(4);
    char buffer[size];

    //Read size bytes from file into buffer
    for(int i = 0; i < size; i++) {
        kernel->machine->ReadMem(addr++, 1, (int*)&buffer[i]);
    }

    SpaceId id = SysExec((char*) buffer);

    kernel->machine->WriteRegister(2, (int)id);

    {
        kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
        kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
        kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
    }
    return;
    ASSERTNOTREACHED();
}
break;
```

b. ksyscall.h

Explanation

Create a new thread for the new executable file and create a new address space for it
If the new thread can be loaded into memory, call Fork function then, which execute the ExecRun() function concurrently.
ExecRun() function executes the current thead.
Return the SpaceId of the new thread.
Note: Since spaceId is similar with threadId that it assigns to each thread uniquely, we could simply assign the value of threadId to spaceId.

```
SpaceId SysExec(char* buffer){
    Thread *thread = new Thread("ThreadExec");
    thread->space = new AddrSpace();

    if(thread->space->Load((char*)buffer)){
        thread->Fork((VoidFunctionPtr)ExecRun, (void*)0);
    }

    return (SpaceId)thread->pid;
}
```

```
void ExecRun() {
    kernel->currentThread->space->Execute();
}
```

c. Testing Result

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Exec
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
status: 0
L[pid 2]: [virtualPage: 12] -> [physicalPage: 0]
L[pid 2]: [virtualPage: 14] -> [physicalPage: 1]
L[pid 2]: [virtualPage: 23] -> [physicalPage: 2]
L[pid 2]: [virtualPage: 15] -> [physicalPage: 3]
L[pid 2]: [virtualPage: 13] -> [physicalPage: 4]
PASS!!status: 0
^C
Cleaning up after signal 2
```

4. Implement Exit system call

a. execption.cc

Explanation

Simply retrieve register 4 as status.
Call SysExit in ksyscall.h

```
case SC_Exit:
{
    DEBUG(dbgSys, "Exit " << kernel->machine->ReadRegister(4) << "\n");
    int status = (int)kernel->machine->ReadRegister(4);

    SysExit(status);

    return;
    ASSERTNOTREACHED();
}
break;
```

b. ksyscall.h

Explanation

Print out the status of the exited thread (this is for testing purpose for Task3)
Simply call Finish() function on the current thread, then we're done.

```
void SysExit(int status){
    cout << "status: " << status << endl;

    kernel->currentThread->Finish();
}
```

c. Testing Result

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Exit
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 4]
PASS!!status: 0
^C
Cleaning up after signal 2
```

Task 2 Solution

1. Implement multiprogramming

a. main.cc

Modify the main function, create a new list named "progList" to store all the names of program that user typed in terminal to execute.

```
int
main(int argc, char **argv)
{
    int i;
    char *debugArg = "";
    char* userProgName = NULL;        // defau

    List<char*>* progList = new List<char*>();

    bool threadTestFlag = false;
    bool consoleTestFlag = false;
    bool networkTestFlag = false;
    bool isRandom = false;
    bool useTLB = false;
```

Modify the main function, append all the string commands that followed by -x as parameters to the list progList.

```
else if (strcmp(argv[i], "-x") == 0) {
    ASSERT(i + 1 < argc);
    userProgName = argv[i + 1];

    progList->Append(userProgName);     // Task 2

    i++;
}
```

Replace the "RunUserProg(userProgName) to the following, as figure showed:
Loop through the list progList, create ListIterator<char*>* items to store command user typed (type: char*). Create a new thread for the new executable file, and call Fork function while calling the RunUserProg concurrently.
Doing this job is to invoke RunUserProg to execute each program via typed in command followed by -x

```
// finally, run an initial user program if requested to do so
// if (userProgName != NULL) {
//  RunUserProg(userProgName);
// }
```

```
if(!progList->IsEmpty()){
    ListIterator<char*>* items = new ListIterator<char*>(progList);

    while(!items->IsDone()){
        if (items->Item() != NULL) {
            Thread* thread = new Thread(items->Item());

            thread->Fork((VoidFunctionPtr)RunUserProg,(void*)items->Item());
            // thread->Fork((VoidFunctionPtr)RunUserProg,it->Item());
        }
        items->Next();
    }
}
```

b. Testing Result

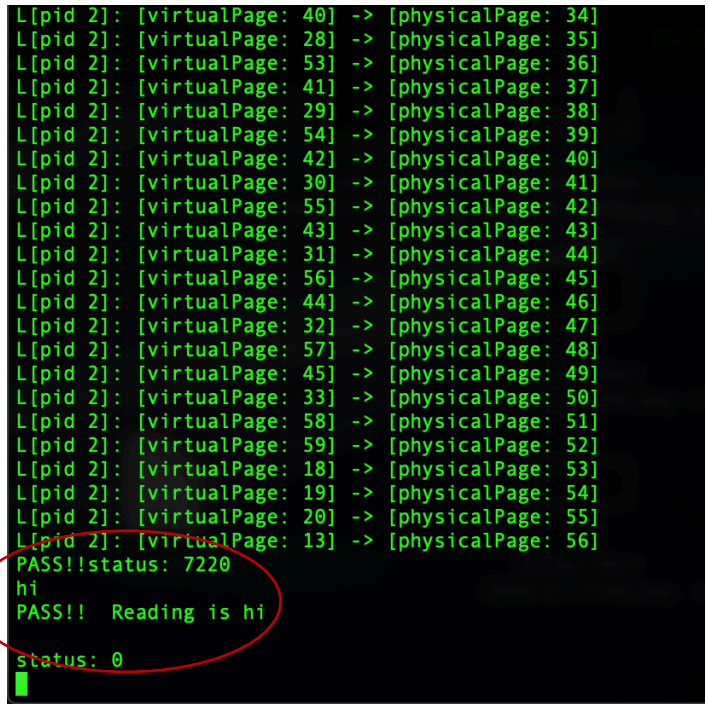
Note that we could only run some small programs for now, as we haven't handled the page fault exception yet. In other word, if the program requires a lot of memory, it might cause error.

(i) Test 1

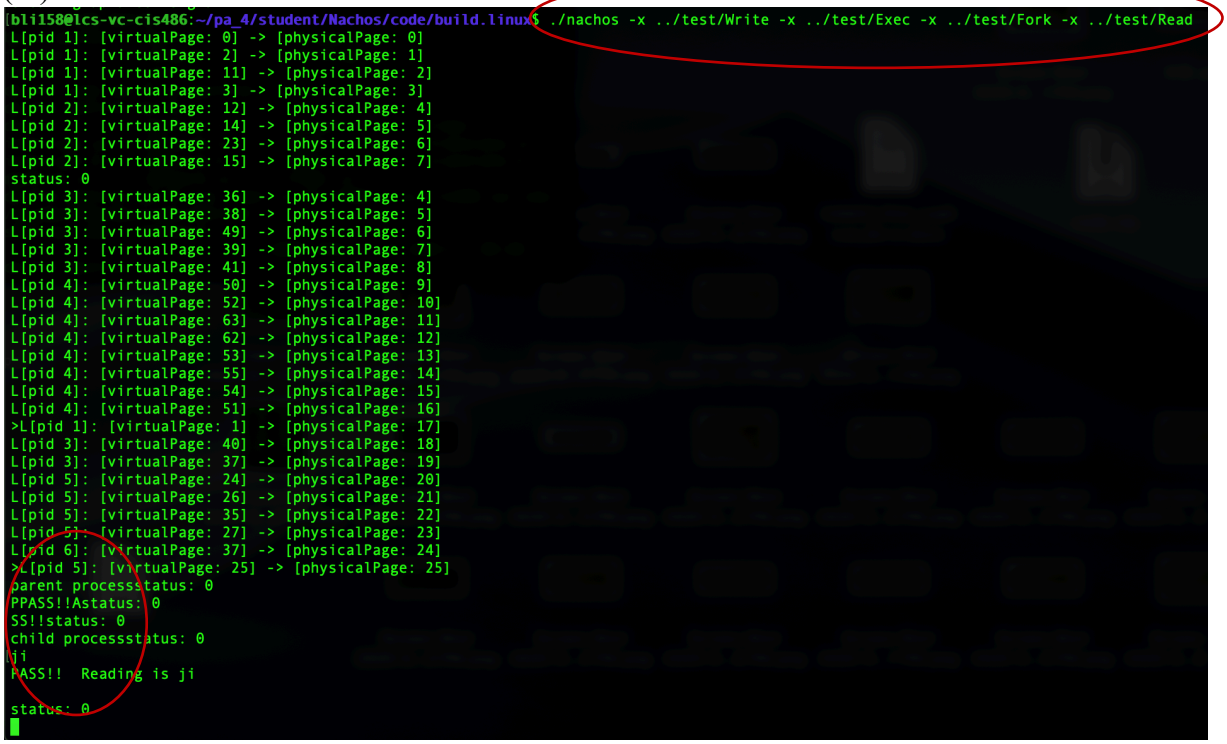
```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Write -x ../test/Read
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 4]
PL[pid 2]: [virtualPage: 12] -> [physicalPage: 5]
L[pid 2]: [virtualPage: 14] -> [physicalPage: 6]
L[pid 2]: [virtualPage: 25] -> [physicalPage: 7]
L[pid 2]: [virtualPage: 24] -> [physicalPage: 8]
L[pid 2]: [virtualPage: 15] -> [physicalPage: 9]
L[pid 2]: [virtualPage: 17] -> [physicalPage: 10]
L[pid 2]: [virtualPage: 16] -> [physicalPage: 11]
L[pid 2]: [virtualPage: 13] -> [physicalPage: 12]
ASS!!status: 0
>>hi
PASS!!  Reading is hi

status: 0
```

(ii) Test 2

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Write -x ../test/matmult -x ../test/Read
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 4]
L[pid 2]: [virtualPage: 12] -> [physicalPage: 5]
L[pid 2]: [virtualPage: 14] -> [physicalPage: 6]
L[pid 2]: [virtualPage: 67] -> [physicalPage: 7]
L[pid 2]: [virtualPage: 15] -> [physicalPage: 8]
L[pid 2]: [virtualPage: 21] -> [physicalPage: 9]
L[pid 2]: [virtualPage: 16] -> [physicalPage: 10]
L[pid 2]: [virtualPage: 34] -> [physicalPage: 11]
L[pid 2]: [virtualPage: 17] -> [physicalPage: 12]
L[pid 3]: [virtualPage: 68] -> [physicalPage: 13]
L[pid 3]: [virtualPage: 70] -> [physicalPage: 14]
L[pid 3]: [virtualPage: 81] -> [physicalPage: 15]
L[pid 3]: [virtualPage: 80] -> [physicalPage: 16]
L[pid 3]: [virtualPage: 71] -> [physicalPage: 17]
L[pid 3]: [virtualPage: 73] -> [physicalPage: 18]
L[pid 3]: [virtualPage: 72] -> [physicalPage: 19]
L[pid 3]: [virtualPage: 69] -> [physicalPage: 20]
PL[pid 2]: [virtualPage: 46] -> [physicalPage: 21]
ASS!L[pid 2]: [virtualPage: 22] -> [physicalPage: 22]
!L[pid 2]: [virtualPage: 47] -> [physicalPage: 23]
status: 0
>>L[pid 2]: [virtualPage: 35] -> [physicalPage: 0]
L[pid 2]: [virtualPage: 23] -> [physicalPage: 1]
L[pid 2]: [virtualPage: 48] -> [physicalPage: 2]
L[pid 2]: [virtualPage: 36] -> [physicalPage: 3]
L[pid 2]: [virtualPage: 24] -> [physicalPage: 4]
L[pid 2]: [virtualPage: 49] -> [physicalPage: 24]
L[pid 2]: [virtualPage: 37] -> [physicalPage: 25]
L[pid 2]: [virtualPage: 25] -> [physicalPage: 26]
L[pid 2]: [virtualPage: 50] -> [physicalPage: 27]
L[pid 2]: [virtualPage: 38] -> [physicalPage: 28]
L[pid 2]: [virtualPage: 26] -> [physicalPage: 29]
L[pid 2]: [virtualPage: 51] -> [physicalPage: 30]
L[pid 2]: [virtualPage: 39] -> [physicalPage: 31]
L[pid 2]: [virtualPage: 27] -> [physicalPage: 32]
L[pid 2]: [virtualPage: 52] -> [physicalPage: 33]
L[pid 2]: [virtualPage: 40] -> [physicalPage: 34]
L[pid 2]: [virtualPage: 28] -> [physicalPage: 35]
L[pid 2]: [virtualPage: 53] -> [physicalPage: 36]
L[pid 2]: [virtualPage: 41] -> [physicalPage: 37]
L[pid 2]: [virtualPage: 29] -> [physicalPage: 38]
L[pid 2]: [virtualPage: 54] -> [physicalPage: 39]
L[pid 2]: [virtualPage: 42] -> [physicalPage: 40]
```



(iii) Test 3



2. Set quantum

Explanation: set variable named quantum -1 as default. If the command user typed includes “-Q” option, set the quantum as the number user typed after “-Q” and pass it to kernel. Otherwise, pass the default one (100) to kernel.

Note that all programs run in Round Robin.

a. kernel.cc

Modify the constructor of kernel, add the command user typed includes “-Q”, set the quantum as the number user typed after “-Q” then.

```
Kernel::Kernel(int argc, char **argv)
{
    // Task 2
    quantum = -1;
    // End task 2
```

```
// Task 2
else if (strcmp(argv[i], "-Q") == 0){
    quantum = atoi(argv[i+1]);
    // cout << "Kernel Quantum: " << quantum << endl;
}
```

Update the constructor of alarm (to add parameter “quantum”) in the Initialize function of kernel. (See alarm.cc)

```
void
Kernel::Initialize(bool israndom, bool usetlb)
{
    isRandom = israndom; // set if random pick page
    useTLB = usetlb;
    // We didn't explicitly allocate the current thread we are running in.
    // But if it ever tries to give up the CPU, we better have a Thread
    // object to save its state.
    currentThread = new Thread("main");
    currentThread->setStatus(RUNNING);

    stats = new Statistics();       // collect statistics
    interrupt = new Interrupt;      // start up interrupt handling
    scheduler = new Scheduler();    // initialize the ready queue

    // alarm = new Alarm(randomSlice);  // start up time slicing
    alarm = new Alarm(randomSlice, quantum);  // Task 2
```

b. alarm.cc

Update the constructor of timer in the constructor of alarm. (See timer.cc)

```
// Alarm::Alarm(bool doRandom)
// Task 2
Alarm::Alarm(bool doRandom, int quantum)
{
    // cout << "alarm: " << quantum << endl;
    timer = new Timer(doRandom, this, quantum);
}
```

c. timer.cc

Add parameter quantum to the constructor of timer.

```
// Timer::Timer(bool doRandom, CallBackObj *toCall)
// Task 2
Timer::Timer(bool doRandom, CallBackObj *toCall, int quantum)
{
    randomize = doRandom;
    callPeriodically = toCall;
    disable = FALSE;
    this->quantum = quantum;  // Taks 2
    // cout << "timer: " << this->quantum << endl;
    SetInterrupt();
}
```

Modify the function named “SetInterrupt” in timer.

If the quantum does not equal to -1, which means that user typed “-Q” and set the quantum himself, pass this new quantum to Schedule function in Interrupt. Otherwise, set the quantum as default (100) and pass it to Schedule function.

```
// Task 2
void
Timer::SetInterrupt()
{
    if (!disable) {
        int actualQ;
        // cout << "Set interrupt: " << quantum << endl;

        if (quantum == -1) {
            actualQ = TimerTicks;
        } else {
            actualQ = quantum;
        }

        if (randomize) {
            actualQ = 1 + (RandomNumber() % (TimerTicks * 2));
        }
        // schedule the next timer device interrupt
        kernel->interrupt->Schedule(this, actualQ, TimerInt);
    }
}
```

d. Testing Result

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Fork
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 4]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 5]
L[pid 2]: [virtualPage: 1] -> [physicalPage: 6]
cL[pid 1]: [virtualPage: 1] -> [physicalPage: 7]
hild processstatus: 0
parent processstatus: 0
^C
Cleaning up after signal 2
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Fork -Q 200
Kernel Quantum: 200
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 4]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 5]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 6]
pL[pid 2]: [virtualPage: 1] -> [physicalPage: 7]
arent processstatus: 0
child processstatus: 0
^C
Cleaning up after signal 2
```

Task 3 Solution

1. Swapfile

In order to implement demand paging, we need to create a swap file for all pages. All data, code and instructions should be stored in swap file first and be loaded into memory later when needed. And when a page is swapped out, its updated data should be stored back to swap file also. In other word, it's for context switch.

Note that kernel.h already defined the variable OpenFile *swapSpace. We could just simply create a new swap space file.

```
OpenFile *swapSpace;
```

```
#ifdef FILESYS_STUB
    fileSystem = new FileSystem();

    // Task 3
    if (fileSystem->Create("swapspace")) {
        swapSpace = fileSystem->Open("swapspace");
    }
    // End Task 3

#else
    fileSystem = new FileSystem(formatFlag);

    // Task 3
    if (fileSystem->Create("swapspace")) {
        swapSpace = fileSystem->Open("swapspace");
    }
    // End Task 3
```

2. Modify / Edit Functions related to Address Space (AddrSpace)

a. Modify the Load function in addrspace.cc

The Load() function loaded the data and other information directly to main memory after creating the page table before.

We modified it that the data is stored in swap space file after creating the page table. And we initialize the physical page number as "-1" of the pages.

```
// Task 3
char * buf;
pageTable = new TranslationEntry[numPages];

for (int i = 0; i < numPages; i++) {

    pageTable[i].virtualPage = kernel->swapCounter++;
    pageTable[i].physicalPage = -1;
    pageTable[i].valid = FALSE;
    pageTable[i].use = FALSE;
    pageTable[i].dirty = FALSE;
    pageTable[i].readOnly = FALSE;

    buf = new char[PageSize];

    executable->ReadAt(buf,PageSize,noffH.code.inFileAddr+(i*PageSize));
    kernel->swapSpace->WriteAt(buf,PageSize,pageTable[i].virtualPage * PageSize);
    // cout << "ReadAt: " << noffH.code.inFileAddr+(i*PageSize) << endl;
    // cout << "WriteAt: " << pageTable[i].virtualPage * PageSize << endl;


}
```

b. Modify the destructor in addrspace.cc

In the destructor of AddrSpace (see AddrSpace.cc), we're checking each page in page table to see if PPN of current page is -1, we will clear the bitMap corresponding to the slot of page table in order that the next program could load into Physical Memory successfully while current program has been finished.

To do this part, the purpose is when a program finish executing, we need to free the space of the program in physical memory and could allow other programs to take these range of slots.

```
AddrSpace::~AddrSpace()
{

    for(int i =0; i<numPages; i++){ // Task 3
        if (pageTable[i].physicalPage != -1)  //phyPage != -1
            kernel->freeMap->Clear(pageTable[i].physicalPage);
    }


    delete pageTable;
}
```

3. Implement backing store for demand paging

The page will only be loaded into memory when needed and a page fault exception will be called if the page is not in physical memory. The detailed will be explained in the following section.

4. Implement page fault handler

a. kernel.h & kernel.cc

(i) kernel.h

Create LRU in kernel header file to operate LRU later

```
List<TranslationEntry *> *LRU;  // Task 3
```

(ii) kernel.cc

Initial swapCounter to be 0 in order that we will use it to the VPN of a page in the Load function (see AddrSpace.cc & Load())

Initialize LRU and create new list of it.

Initialize bitMap (freeMap) and create new to check whether the current address space of the Physical Memory could be allocated. All physical pages are marked as free or used in Bitmap to help us implement the page fault exception.

```
// Task 3
swapCounter = 0;
LRU = new List<TranslationEntry*>();
freeMap = new Bitmap(NumPhysPages);
// End Task 3
```

Remember to delete LRU and freeMap in the destructor of kernel, to free memory.

```
Kernel::~Kernel()
{
    delete stats;
    delete interrupt;
    delete scheduler;
    delete alarm;
    delete machine;
    delete synchConsoleIn;
    delete synchConsoleOut;
    delete synchDisk;
    delete fileSystem;
    delete postOfficeIn;
    delete postOfficeOut;

    delete LRU; // Task 3
    delete freeMap; // Task 3

    Exit(0);
}
```

b. translate.cc

We firstly check if the current entry from page table is in LRU list, if so, we remove the first one, and then append the exact same entry into the end of LRU list. This logic is used to Least Recently Used. Since the first element in the LRU is Least Recently Used, as long as we found the entry, we need to move it back to the end of LRU list.

```
// Task 3
if (kernel->LRU->IsInList(entry)) {
  kernel->LRU->Remove(entry);
}

kernel->LRU->Append(entry);
// Task 3 End
```

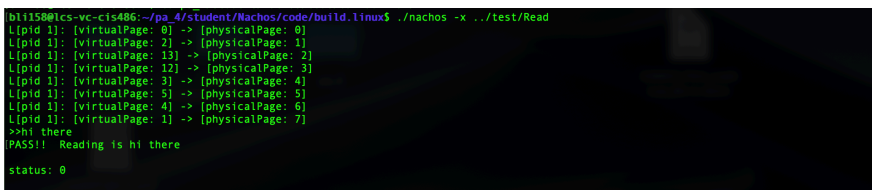
c. exception.cc

Add new case named “PageFaultException” in ExceptionHandler() function.
Increases the number of page faults ((int) numPageFault ++)
Get the address of the page causing page fault, compute the PPN then
Call the FindAndSet() function to find a free physical memory page and marked it as used if found.
If we could find a free page, we could simply load that new page into memory. Mark the valid bit of that new page as TRUE, load the data, code and instructions from the corresponding place of the swap space file into memory. Last but not least, append the page into the end of our LRU list.
If we couldn't find a free page in physical memory, which means that the physical memory is full, we’ll swap out the least used page first, and load the new one. According to our algorithm, the first entry in the LRU list will be the least used one (a page is moved or appended to the end of the list every time it’s accessed). We remove the first one from LRU list, mark the corresponding physical page as free, and store the updated data back to its swap space file. We’ll load the new page into memory then, load the data, code and instructions from swap space file, mark the free physical page as used, set the valid bit as TRUE in page table, and append the new entry to the end of LRU list.

```
case PageFaultException:{
    //Increse number of page faults
    kernel->stats->numPageFaults++;
    //Address of page fault
    int pageFaultAdd = (int)kernel->machine->ReadRegister(BadVAddrReg);
    // Compute the physical page number
    int pageFaultPPN = (int)pageFaultAdd / PageSize;
    // Try to get a free page in physical memory and set that it as used if found
    int PPN = kernel->freeMap->FindAndSet();
    int pid = kernel->currentThread->pid;
    //Get the page entry
    TranslationEntry* pageEntry  = kernel->currentThread->space->getPageEntry(pageFaultPPN);
    // The previous entry accessed
    TranslationEntry* preEntry;

    // If the physical memory isn't full, we could load a new page into it
    if (PPN != -1){
        //Set the corresponding ppn in page table as the number of that page we're gonna use
        pageEntry->physicalPage = PPN;
        //Set the valid bit as TRUE
        pageEntry->valid = TRUE;
        //Read the data&instruction from the corresponding swap file
        kernel->swapSpace->ReadAt(&kernel->machine->mainMemory[PPN*PageSize],
            PageSize,pageEntry->virtualPage * PageSize);
        //Append the page to the end of LRU list
        kernel->LRU->Append(pageEntry);
        //Print the output
        cout << "L[pid " << pid << "]: [virtualPage: " << pageEntry->virtualPage << "] -> [physicalPage: "
        << pageEntry->physicalPage << "]" << endl;

    }
    else {
        //According to our algorithm, the first page on the list is the least used one
        preEntry = kernel->LRU->RemoveFront();
        int physicalPN = preEntry->physicalPage;
        //Store the updated data back to swap file
        kernel->swapSpace->WriteAt(&kernel->machine->mainMemory[physicalPN * PageSize],
            PageSize,preEntry->virtualPage * PageSize);
        //Mark the corresponding physical page as free
        preEntry->physicalPage = -1;
        preEntry->valid = FALSE;

        //Load the new page into memory
        pageEntry->physicalPage = physicalPN;
        pageEntry->valid = TRUE;
        kernel->swapSpace->ReadAt(&kernel->machine->mainMemory[physicalPN * PageSize],
            PageSize,pageEntry->virtualPage * PageSize);
        //Append the new loaded page into the end of LRU list
        kernel->LRU->Append(pageEntry);

        cout << "S[pid " << pid << "]: [physicalPage: " << pageEntry->physicalPage
        << "] -> [virtualPage: " << pageEntry->virtualPage << "]" << endl;
    }
    return;
}
break;
```

Testing Result

1. Test 1
   - Read



   - Write

   - Exit

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Exit
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 4]
PASS!!status: 0
```

- Exec

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Exec
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
status: 0
L[pid 2]: [virtualPage: 12] -> [physicalPage: 0]
L[pid 2]: [virtualPage: 14] -> [physicalPage: 1]
L[pid 2]: [virtualPage: 23] -> [physicalPage: 2]
L[pid 2]: [virtualPage: 15] -> [physicalPage: 3]
L[pid 2]: [virtualPage: 13] -> [physicalPage: 4]
PASS!!status: 0
```

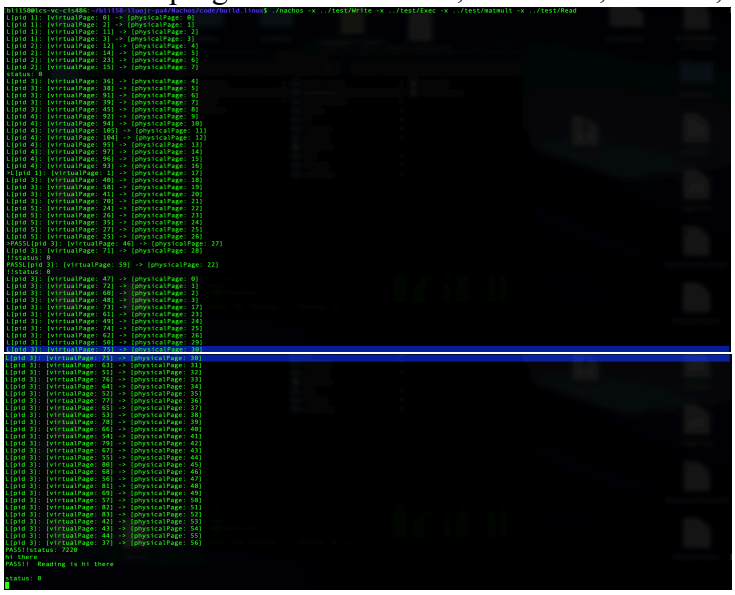
- Fork

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Fork
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 4]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 5]
L[pid 2]: [virtualPage: 1] -> [physicalPage: 6]
cL[pid 1]: [virtualPage: 1] -> [physicalPage: 7]
hild processstatus: 0
parent processstatus: 0
```

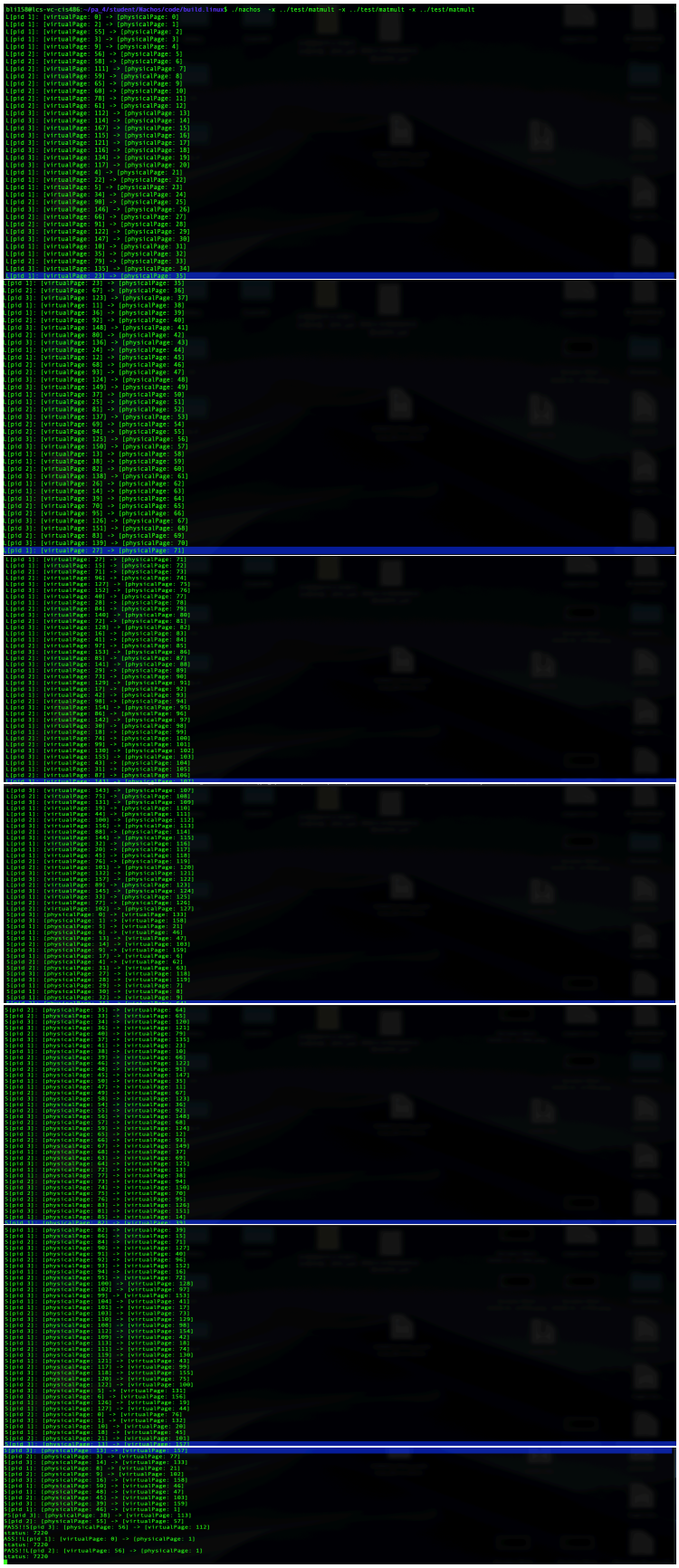
2. Test 2
   - Quantum

```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Fork
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 4]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 5]
L[pid 2]: [virtualPage: 1] -> [physicalPage: 6]
cL[pid 1]: [virtualPage: 1] -> [physicalPage: 7]
hild processstatus: 0
parent processstatus: 0
^C
Cleaning up after signal 2
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos -x ../test/Fork -Q 200
Kernel Quantum: 200
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 4]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 5]
L[pid 1]: [virtualPage: 1] -> [physicalPage: 6]
cL[pid 2]: [virtualPage: 1] -> [physicalPage: 7]
arent processstatus: 0
child processstatus: 0
^C
Cleaning up after signal 2
```

   - Multiprogram. Run “Write”, “matmul”, “Exec”, “Read”



3. Test 3
    - Run 3 times matmul

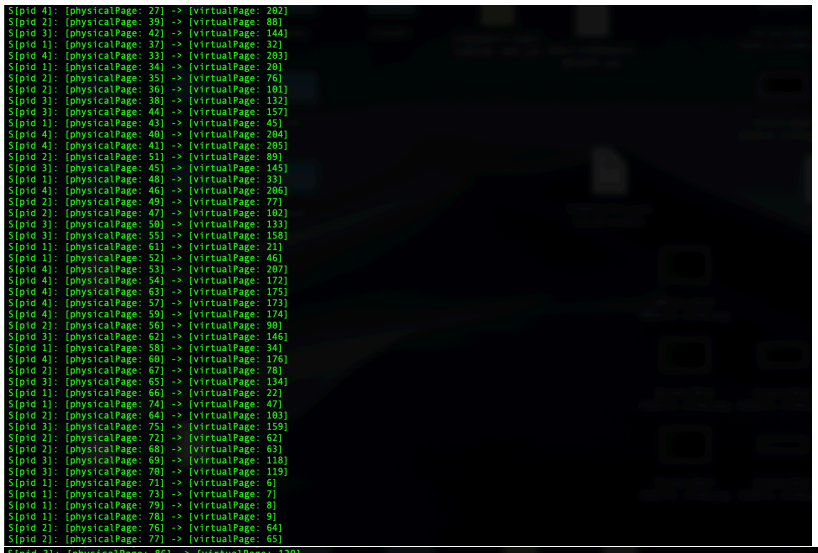


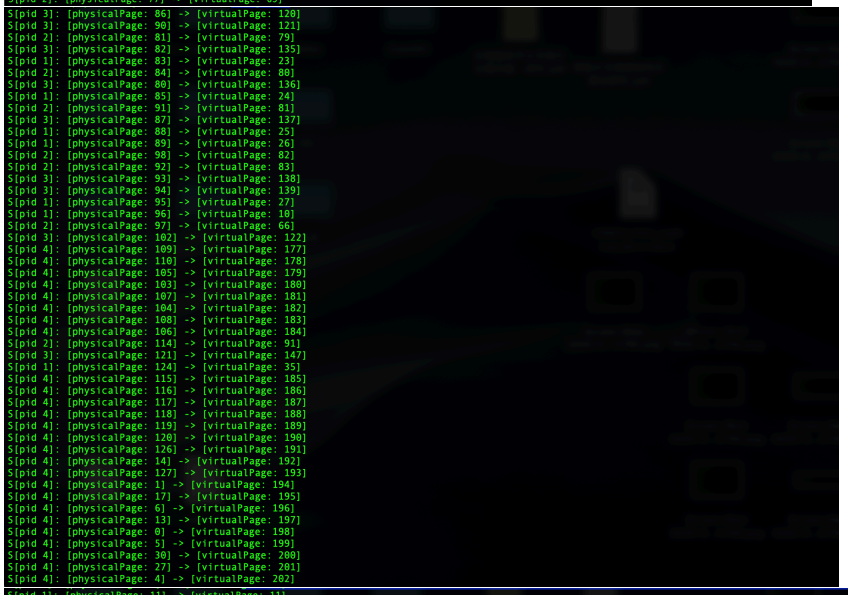
- Run 3 matmul and sort

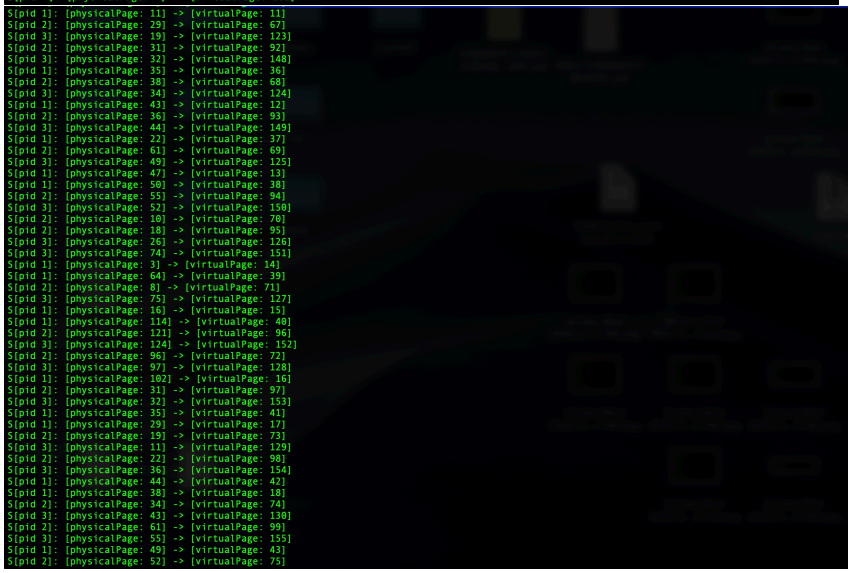
```
bli158@lcs-vc-cis486:~/pa_4/student/Nachos/code/build.linux$ ./nachos  -x ../test/matmult -x ../test/matmult -x ../test/matmult -x ../test/sort
L[pid 1]: [virtualPage: 0] -> [physicalPage: 0]
L[pid 1]: [virtualPage: 2] -> [physicalPage: 1]
L[pid 1]: [virtualPage: 55] -> [physicalPage: 2]
L[pid 1]: [virtualPage: 3] -> [physicalPage: 3]
L[pid 1]: [virtualPage: 9] -> [physicalPage: 4]
L[pid 2]: [virtualPage: 56] -> [physicalPage: 5]
L[pid 2]: [virtualPage: 58] -> [physicalPage: 6]
L[pid 2]: [virtualPage: 111] -> [physicalPage: 7]
L[pid 2]: [virtualPage: 59] -> [physicalPage: 8]
L[pid 2]: [virtualPage: 65] -> [physicalPage: 9]
L[pid 2]: [virtualPage: 60] -> [physicalPage: 10]
L[pid 2]: [virtualPage: 78] -> [physicalPage: 11]
L[pid 2]: [virtualPage: 61] -> [physicalPage: 12]
L[pid 3]: [virtualPage: 112] -> [physicalPage: 13]
L[pid 3]: [virtualPage: 114] -> [physicalPage: 14]
L[pid 3]: [virtualPage: 167] -> [physicalPage: 15]
L[pid 3]: [virtualPage: 115] -> [physicalPage: 16]
L[pid 3]: [virtualPage: 121] -> [physicalPage: 17]
L[pid 3]: [virtualPage: 116] -> [physicalPage: 18]
L[pid 3]: [virtualPage: 134] -> [physicalPage: 19]
L[pid 3]: [virtualPage: 117] -> [physicalPage: 20]
L[pid 4]: [virtualPage: 168] -> [physicalPage: 21]
L[pid 4]: [virtualPage: 170] -> [physicalPage: 22]
L[pid 4]: [virtualPage: 215] -> [physicalPage: 23]
L[pid 4]: [virtualPage: 171] -> [physicalPage: 24]
L[pid 4]: [virtualPage: 175] -> [physicalPage: 25]
L[pid 1]: [virtualPage: 4] -> [physicalPage: 26]
L[pid 1]: [virtualPage: 22] -> [physicalPage: 27]
L[pid 1]: [virtualPage: 5] -> [physicalPage: 28]
L[pid 1]: [virtualPage: 34] -> [physicalPage: 29]
L[pid 2]: [virtualPage: 90] -> [physicalPage: 30]
L[pid 3]: [virtualPage: 146] -> [physicalPage: 31]
L[pid 4]: [virtualPage: 176] -> [physicalPage: 32]
L[pid 2]: [virtualPage: 66] -> [physicalPage: 33]
L[pid 2]: [virtualPage: 91] -> [physicalPage: 34]
L[pid 3]: [virtualPage: 122] -> [physicalPage: 35]
L[pid 3]: [virtualPage: 147] -> [physicalPage: 36]
L[pid 1]: [virtualPage: 10] -> [physicalPage: 37]
L[pid 1]: [virtualPage: 35] -> [physicalPage: 38]
L[pid 4]: [virtualPage: 177] -> [physicalPage: 39]
L[pid 2]: [virtualPage: 79] -> [physicalPage: 40]
L[pid 3]: [virtualPage: 135] -> [physicalPage: 41]
L[pid 4]: [virtualPage: 178] -> [physicalPage: 42]
L[pid 1]: [virtualPage: 23] -> [physicalPage: 43]
L[pid 4]: [virtualPage: 179] -> [physicalPage: 44]
L[pid 2]: [virtualPage: 67] -> [physicalPage: 45]
L[pid 3]: [virtualPage: 123] -> [physicalPage: 46]
L[pid 1]: [virtualPage: 11] -> [physicalPage: 47]
```

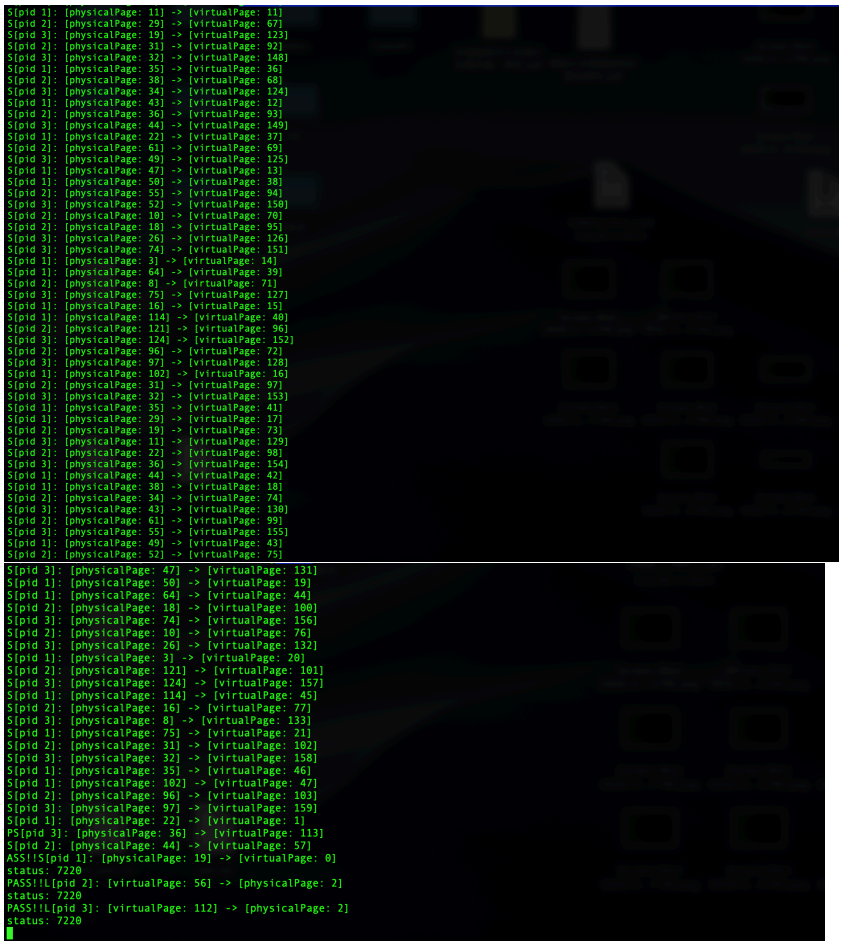
```
L[pid 2]: [virtualPage: 92] -> [physicalPage: 48]
L[pid 3]: [virtualPage: 148] -> [physicalPage: 49]
L[pid 1]: [virtualPage: 36] -> [physicalPage: 50]
L[pid 4]: [virtualPage: 180] -> [physicalPage: 51]
L[pid 2]: [virtualPage: 80] -> [physicalPage: 52]
L[pid 3]: [virtualPage: 136] -> [physicalPage: 53]
L[pid 1]: [virtualPage: 24] -> [physicalPage: 54]
L[pid 4]: [virtualPage: 181] -> [physicalPage: 55]
L[pid 1]: [virtualPage: 12] -> [physicalPage: 56]
L[pid 2]: [virtualPage: 68] -> [physicalPage: 57]
L[pid 2]: [virtualPage: 93] -> [physicalPage: 58]
L[pid 3]: [virtualPage: 124] -> [physicalPage: 59]
L[pid 3]: [virtualPage: 149] -> [physicalPage: 60]
L[pid 4]: [virtualPage: 182] -> [physicalPage: 61]
L[pid 1]: [virtualPage: 37] -> [physicalPage: 62]
L[pid 4]: [virtualPage: 183] -> [physicalPage: 63]
L[pid 1]: [virtualPage: 25] -> [physicalPage: 64]
L[pid 2]: [virtualPage: 81] -> [physicalPage: 65]
L[pid 3]: [virtualPage: 137] -> [physicalPage: 66]
L[pid 4]: [virtualPage: 184] -> [physicalPage: 67]
L[pid 2]: [virtualPage: 69] -> [physicalPage: 68]
L[pid 2]: [virtualPage: 94] -> [physicalPage: 69]
L[pid 3]: [virtualPage: 125] -> [physicalPage: 70]
L[pid 3]: [virtualPage: 150] -> [physicalPage: 71]
L[pid 1]: [virtualPage: 13] -> [physicalPage: 72]
L[pid 1]: [virtualPage: 38] -> [physicalPage: 73]
L[pid 4]: [virtualPage: 185] -> [physicalPage: 74]
L[pid 4]: [virtualPage: 186] -> [physicalPage: 75]
L[pid 2]: [virtualPage: 82] -> [physicalPage: 76]
L[pid 3]: [virtualPage: 138] -> [physicalPage: 77]
L[pid 1]: [virtualPage: 26] -> [physicalPage: 78]
L[pid 4]: [virtualPage: 187] -> [physicalPage: 79]
L[pid 1]: [virtualPage: 14] -> [physicalPage: 80]
L[pid 2]: [virtualPage: 70] -> [physicalPage: 81]
L[pid 2]: [virtualPage: 95] -> [physicalPage: 82]
L[pid 3]: [virtualPage: 126] -> [physicalPage: 83]
L[pid 3]: [virtualPage: 151] -> [physicalPage: 84]
L[pid 1]: [virtualPage: 39] -> [physicalPage: 85]
L[pid 4]: [virtualPage: 188] -> [physicalPage: 86]
L[pid 2]: [virtualPage: 83] -> [physicalPage: 87]
L[pid 3]: [virtualPage: 139] -> [physicalPage: 88]
L[pid 1]: [virtualPage: 27] -> [physicalPage: 89]
L[pid 4]: [virtualPage: 189] -> [physicalPage: 90]
L[pid 4]: [virtualPage: 190] -> [physicalPage: 91]
L[pid 1]: [virtualPage: 15] -> [physicalPage: 92]
L[pid 2]: [virtualPage: 71] -> [physicalPage: 93]
L[pid 2]: [virtualPage: 96] -> [physicalPage: 94]
L[pid 3]: [virtualPage: 127] -> [physicalPage: 95]
L[pid 3]: [virtualPage: 152] -> [physicalPage: 96]
```

```
L[pid 1]: [virtualPage: 40] -> [physicalPage: 97]
L[pid 4]: [virtualPage: 191] -> [physicalPage: 98]
L[pid 1]: [virtualPage: 28] -> [physicalPage: 99]
L[pid 2]: [virtualPage: 84] -> [physicalPage: 100]
L[pid 3]: [virtualPage: 140] -> [physicalPage: 101]
L[pid 4]: [virtualPage: 192] -> [physicalPage: 102]
L[pid 2]: [virtualPage: 72] -> [physicalPage: 103]
L[pid 3]: [virtualPage: 128] -> [physicalPage: 104]
L[pid 1]: [virtualPage: 16] -> [physicalPage: 105]
L[pid 1]: [virtualPage: 41] -> [physicalPage: 106]
L[pid 2]: [virtualPage: 97] -> [physicalPage: 107]
L[pid 3]: [virtualPage: 153] -> [physicalPage: 108]
L[pid 4]: [virtualPage: 193] -> [physicalPage: 109]
L[pid 4]: [virtualPage: 194] -> [physicalPage: 110]
L[pid 2]: [virtualPage: 85] -> [physicalPage: 111]
L[pid 3]: [virtualPage: 141] -> [physicalPage: 112]
L[pid 1]: [virtualPage: 29] -> [physicalPage: 113]
L[pid 4]: [virtualPage: 195] -> [physicalPage: 114]
L[pid 2]: [virtualPage: 73] -> [physicalPage: 115]
L[pid 3]: [virtualPage: 129] -> [physicalPage: 116]
L[pid 1]: [virtualPage: 17] -> [physicalPage: 117]
L[pid 1]: [virtualPage: 42] -> [physicalPage: 118]
L[pid 2]: [virtualPage: 98] -> [physicalPage: 119]
L[pid 3]: [virtualPage: 154] -> [physicalPage: 120]
L[pid 4]: [virtualPage: 196] -> [physicalPage: 121]
L[pid 2]: [virtualPage: 86] -> [physicalPage: 122]
L[pid 3]: [virtualPage: 142] -> [physicalPage: 123]
L[pid 4]: [virtualPage: 197] -> [physicalPage: 124]
L[pid 1]: [virtualPage: 30] -> [physicalPage: 125]
L[pid 4]: [virtualPage: 198] -> [physicalPage: 126]
L[pid 2]: [virtualPage: 74] -> [physicalPage: 127]
S[pid 2]: [physicalPage: 0] -> [virtualPage: 99]
S[pid 3]: [physicalPage: 1] -> [virtualPage: 130]
S[pid 3]: [physicalPage: 5] -> [virtualPage: 155]
S[pid 1]: [physicalPage: 6] -> [virtualPage: 18]
S[pid 1]: [physicalPage: 13] -> [virtualPage: 43]
S[pid 4]: [physicalPage: 14] -> [virtualPage: 199]
S[pid 1]: [physicalPage: 21] -> [virtualPage: 31]
S[pid 2]: [physicalPage: 25] -> [virtualPage: 87]
S[pid 3]: [physicalPage: 9] -> [virtualPage: 143]
S[pid 4]: [physicalPage: 17] -> [virtualPage: 200]
S[pid 2]: [physicalPage: 4] -> [virtualPage: 75]
S[pid 3]: [physicalPage: 29] -> [virtualPage: 131]
S[pid 4]: [physicalPage: 30] -> [virtualPage: 201]
S[pid 1]: [physicalPage: 31] -> [virtualPage: 19]
S[pid 1]: [physicalPage: 32] -> [virtualPage: 44]
S[pid 2]: [physicalPage: 11] -> [virtualPage: 100]
S[pid 3]: [physicalPage: 19] -> [virtualPage: 156]
```


S[pid 4]: [physicalPage: 27] -> [virtualPage: 202]
S[pid 2]: [physicalPage: 39] -> [virtualPage: 88]
S[pid 3]: [physicalPage: 42] -> [virtualPage: 144]
S[pid 1]: [physicalPage: 37] -> [virtualPage: 32]
S[pid 4]: [physicalPage: 33] -> [virtualPage: 203]
S[pid 1]: [physicalPage: 34] -> [virtualPage: 20]
S[pid 2]: [physicalPage: 35] -> [virtualPage: 76]
S[pid 2]: [physicalPage: 36] -> [virtualPage: 101]
S[pid 3]: [physicalPage: 38] -> [virtualPage: 132]
S[pid 3]: [physicalPage: 44] -> [virtualPage: 157]
S[pid 1]: [physicalPage: 43] -> [virtualPage: 45]
S[pid 4]: [physicalPage: 40] -> [virtualPage: 204]
S[pid 4]: [physicalPage: 41] -> [virtualPage: 205]
S[pid 2]: [physicalPage: 51] -> [virtualPage: 89]
S[pid 3]: [physicalPage: 45] -> [virtualPage: 145]
S[pid 1]: [physicalPage: 48] -> [virtualPage: 33]
S[pid 4]: [physicalPage: 46] -> [virtualPage: 206]
S[pid 2]: [physicalPage: 49] -> [virtualPage: 77]
S[pid 2]: [physicalPage: 47] -> [virtualPage: 102]
S[pid 3]: [physicalPage: 50] -> [virtualPage: 133]
S[pid 3]: [physicalPage: 55] -> [virtualPage: 158]
S[pid 1]: [physicalPage: 61] -> [virtualPage: 21]
S[pid 1]: [physicalPage: 52] -> [virtualPage: 46]
S[pid 4]: [physicalPage: 53] -> [virtualPage: 207]
S[pid 4]: [physicalPage: 54] -> [virtualPage: 172]
S[pid 4]: [physicalPage: 63] -> [virtualPage: 175]
S[pid 4]: [physicalPage: 57] -> [virtualPage: 173]
S[pid 4]: [physicalPage: 59] -> [virtualPage: 174]
S[pid 2]: [physicalPage: 56] -> [virtualPage: 90]
S[pid 3]: [physicalPage: 62] -> [virtualPage: 146]
S[pid 1]: [physicalPage: 58] -> [virtualPage: 34]
S[pid 4]: [physicalPage: 60] -> [virtualPage: 176]
S[pid 2]: [physicalPage: 67] -> [virtualPage: 78]
S[pid 3]: [physicalPage: 65] -> [virtualPage: 134]
S[pid 1]: [physicalPage: 66] -> [virtualPage: 22]
S[pid 1]: [physicalPage: 74] -> [virtualPage: 47]
S[pid 2]: [physicalPage: 64] -> [virtualPage: 103]
S[pid 3]: [physicalPage: 75] -> [virtualPage: 159]
S[pid 2]: [physicalPage: 72] -> [virtualPage: 62]
S[pid 2]: [physicalPage: 68] -> [virtualPage: 63]
S[pid 3]: [physicalPage: 69] -> [virtualPage: 118]
S[pid 3]: [physicalPage: 70] -> [virtualPage: 119]
S[pid 1]: [physicalPage: 71] -> [virtualPage: 6]
S[pid 1]: [physicalPage: 73] -> [virtualPage: 7]
S[pid 1]: [physicalPage: 79] -> [virtualPage: 8]
S[pid 1]: [physicalPage: 78] -> [virtualPage: 9]
S[pid 2]: [physicalPage: 76] -> [virtualPage: 64]
S[pid 2]: [physicalPage: 77] -> [virtualPage: 65]


S[pid 3]: [physicalPage: 86] -> [virtualPage: 120]
S[pid 3]: [physicalPage: 90] -> [virtualPage: 121]
S[pid 2]: [physicalPage: 81] -> [virtualPage: 79]
S[pid 3]: [physicalPage: 82] -> [virtualPage: 135]
S[pid 1]: [physicalPage: 83] -> [virtualPage: 23]
S[pid 2]: [physicalPage: 84] -> [virtualPage: 80]
S[pid 3]: [physicalPage: 80] -> [virtualPage: 136]
S[pid 1]: [physicalPage: 85] -> [virtualPage: 24]
S[pid 2]: [physicalPage: 91] -> [virtualPage: 81]
S[pid 3]: [physicalPage: 87] -> [virtualPage: 137]
S[pid 1]: [physicalPage: 88] -> [virtualPage: 25]
S[pid 1]: [physicalPage: 89] -> [virtualPage: 26]
S[pid 2]: [physicalPage: 98] -> [virtualPage: 82]
S[pid 2]: [physicalPage: 92] -> [virtualPage: 83]
S[pid 3]: [physicalPage: 93] -> [virtualPage: 138]
S[pid 3]: [physicalPage: 94] -> [virtualPage: 139]
S[pid 1]: [physicalPage: 95] -> [virtualPage: 27]
S[pid 1]: [physicalPage: 96] -> [virtualPage: 10]
S[pid 2]: [physicalPage: 97] -> [virtualPage: 66]
S[pid 3]: [physicalPage: 102] -> [virtualPage: 122]
S[pid 4]: [physicalPage: 109] -> [virtualPage: 177]
S[pid 4]: [physicalPage: 110] -> [virtualPage: 178]
S[pid 4]: [physicalPage: 105] -> [virtualPage: 179]
S[pid 4]: [physicalPage: 103] -> [virtualPage: 180]
S[pid 4]: [physicalPage: 107] -> [virtualPage: 181]
S[pid 4]: [physicalPage: 104] -> [virtualPage: 182]
S[pid 4]: [physicalPage: 108] -> [virtualPage: 183]
S[pid 4]: [physicalPage: 106] -> [virtualPage: 184]
S[pid 2]: [physicalPage: 114] -> [virtualPage: 91]
S[pid 3]: [physicalPage: 121] -> [virtualPage: 147]
S[pid 1]: [physicalPage: 124] -> [virtualPage: 35]
S[pid 4]: [physicalPage: 115] -> [virtualPage: 185]
S[pid 4]: [physicalPage: 116] -> [virtualPage: 186]
S[pid 4]: [physicalPage: 117] -> [virtualPage: 187]
S[pid 4]: [physicalPage: 118] -> [virtualPage: 188]
S[pid 4]: [physicalPage: 119] -> [virtualPage: 189]
S[pid 4]: [physicalPage: 120] -> [virtualPage: 190]
S[pid 4]: [physicalPage: 126] -> [virtualPage: 191]
S[pid 4]: [physicalPage: 14] -> [virtualPage: 192]
S[pid 4]: [physicalPage: 127] -> [virtualPage: 193]
S[pid 4]: [physicalPage: 1] -> [virtualPage: 194]
S[pid 4]: [physicalPage: 17] -> [virtualPage: 195]
S[pid 4]: [physicalPage: 6] -> [virtualPage: 196]
S[pid 4]: [physicalPage: 13] -> [virtualPage: 197]
S[pid 4]: [physicalPage: 0] -> [virtualPage: 198]
S[pid 4]: [physicalPage: 5] -> [virtualPage: 199]
S[pid 4]: [physicalPage: 30] -> [virtualPage: 200]
S[pid 4]: [physicalPage: 27] -> [virtualPage: 201]
S[pid 4]: [physicalPage: 4] -> [virtualPage: 202]


S[pid 1]: [physicalPage: 11] -> [virtualPage: 11]
S[pid 2]: [physicalPage: 29] -> [virtualPage: 67]
S[pid 3]: [physicalPage: 19] -> [virtualPage: 123]
S[pid 2]: [physicalPage: 31] -> [virtualPage: 92]
S[pid 3]: [physicalPage: 32] -> [virtualPage: 140]
S[pid 1]: [physicalPage: 35] -> [virtualPage: 36]
S[pid 2]: [physicalPage: 38] -> [virtualPage: 68]
S[pid 3]: [physicalPage: 34] -> [virtualPage: 124]
S[pid 1]: [physicalPage: 43] -> [virtualPage: 12]
S[pid 2]: [physicalPage: 36] -> [virtualPage: 93]
S[pid 3]: [physicalPage: 44] -> [virtualPage: 149]
S[pid 1]: [physicalPage: 22] -> [virtualPage: 37]
S[pid 2]: [physicalPage: 61] -> [virtualPage: 69]
S[pid 3]: [physicalPage: 49] -> [virtualPage: 125]
S[pid 1]: [physicalPage: 47] -> [virtualPage: 13]
S[pid 1]: [physicalPage: 50] -> [virtualPage: 38]
S[pid 2]: [physicalPage: 55] -> [virtualPage: 94]
S[pid 3]: [physicalPage: 52] -> [virtualPage: 150]
S[pid 2]: [physicalPage: 10] -> [virtualPage: 70]
S[pid 2]: [physicalPage: 18] -> [virtualPage: 95]
S[pid 3]: [physicalPage: 26] -> [virtualPage: 126]
S[pid 3]: [physicalPage: 74] -> [virtualPage: 151]
S[pid 1]: [physicalPage: 3] -> [virtualPage: 14]
S[pid 1]: [physicalPage: 64] -> [virtualPage: 39]
S[pid 2]: [physicalPage: 8] -> [virtualPage: 71]
S[pid 3]: [physicalPage: 75] -> [virtualPage: 127]
S[pid 1]: [physicalPage: 16] -> [virtualPage: 15]
S[pid 1]: [physicalPage: 114] -> [virtualPage: 40]
S[pid 2]: [physicalPage: 121] -> [virtualPage: 96]
S[pid 3]: [physicalPage: 124] -> [virtualPage: 152]
S[pid 2]: [physicalPage: 96] -> [virtualPage: 72]
S[pid 3]: [physicalPage: 97] -> [virtualPage: 128]
S[pid 1]: [physicalPage: 102] -> [virtualPage: 16]
S[pid 2]: [physicalPage: 31] -> [virtualPage: 97]
S[pid 3]: [physicalPage: 32] -> [virtualPage: 153]
S[pid 1]: [physicalPage: 35] -> [virtualPage: 41]
S[pid 1]: [physicalPage: 29] -> [virtualPage: 17]
S[pid 2]: [physicalPage: 19] -> [virtualPage: 73]
S[pid 3]: [physicalPage: 11] -> [virtualPage: 129]
S[pid 2]: [physicalPage: 22] -> [virtualPage: 98]
S[pid 3]: [physicalPage: 36] -> [virtualPage: 154]
S[pid 1]: [physicalPage: 44] -> [virtualPage: 42]
S[pid 1]: [physicalPage: 38] -> [virtualPage: 18]
S[pid 2]: [physicalPage: 34] -> [virtualPage: 74]
S[pid 3]: [physicalPage: 43] -> [virtualPage: 130]
S[pid 2]: [physicalPage: 61] -> [virtualPage: 99]
S[pid 3]: [physicalPage: 55] -> [virtualPage: 155]
S[pid 1]: [physicalPage: 49] -> [virtualPage: 43]
S[pid 2]: [physicalPage: 52] -> [virtualPage: 75]

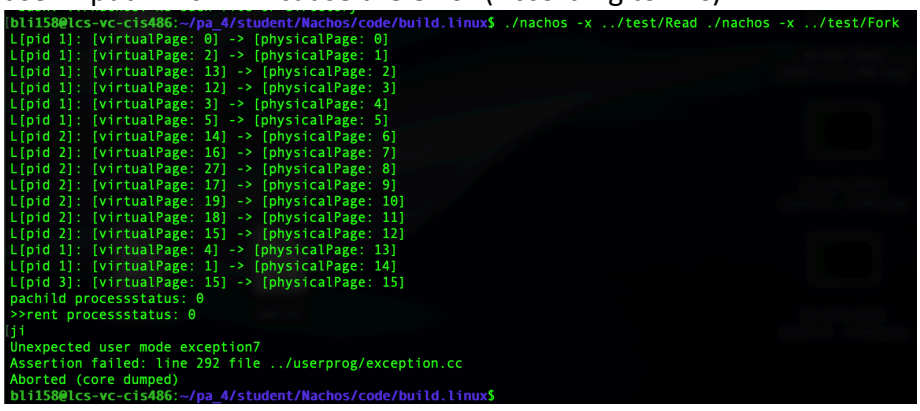


Remarks

1. ./nachos -x ../test/Read ./nachos -x ../test/Fork

We noticed that the command above will cause unexpected user mode error after we type for the Read program. Figure shows the error message.

Note that this is because when running multiple programs, Read has to be the last one to call, otherwise, Read program will simply assume the command string after it as the user input which will cause the error (According to TAs).



2. We were thinking to handle all memory leaks using delete all instances created by new, but it's complex to handle all of them. In some case, if we delete some instance, errors will take place. Since Nachos has a bunch of bugs regarding to memory leak, we did not handle them all comprehensively. (According to TAs).