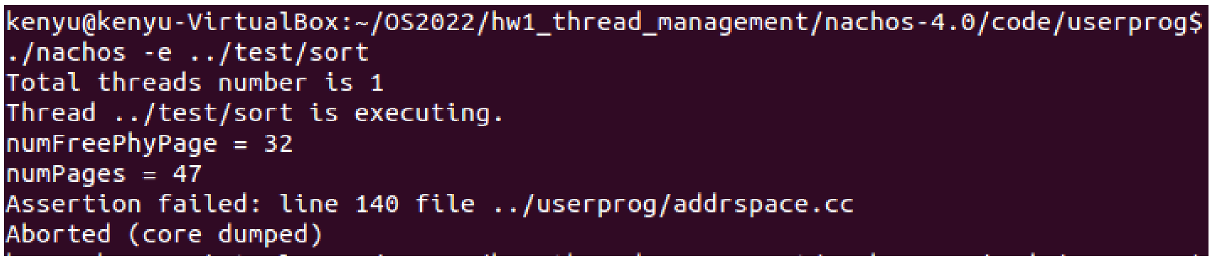
OS2022 HW3 - Memory Management

Student: 游家權(r10942152)

A. Motivation

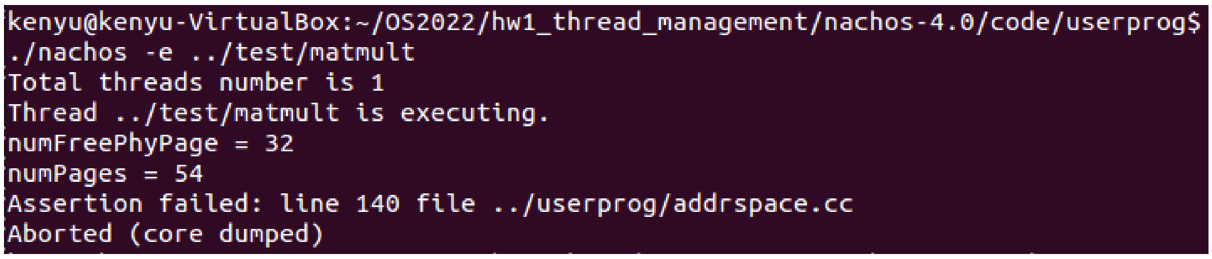
In this assignment, I need to implement virtual memory to execute two high-memory-consumed processes at the same time. Because we only have 32 physical pages for Nacho default setting(4096 MB memory in total), any process requiring more than 32 pages will not be able to execute.

As we can see below, when I execute a test/sort program, which requires 47 physical pages. The kernel is not able to load the process into memory and assertion check fails in addrspace.cc.



```
kenyu@kenyu-VirtualBox:~/OS2022/hw1_thread_management/nachos-4.0/code/userprog$
./nachos -e ../test/sort
Total threads number is 1
Thread ../test/sort is executing.
numFreePhyPage = 32
numPages = 47
Assertion failed: line 140 file ../userprog/addrspace.cc
Aborted (core dumped)
```

If I try to execute another process that requires 54 pages, the same thing will happen as shown below.



```
kenyu@kenyu-VirtualBox:~/OS2022/hw1_thread_management/nachos-4.0/code/userprog$
./nachos -e ../test/matmult
Total threads number is 1
Thread ../test/matmult is executing.
numFreePhyPage = 32
numPages = 54
Assertion failed: line 140 file ../userprog/addrspace.cc
Aborted (core dumped)
```

Therefore, the goal of this project is to implement virtual memory in Nachos, and allow the kernel to swap pages between memory and the second storage system. After this improvement, I expect the operating system should be able to execute these two programs concurrently.

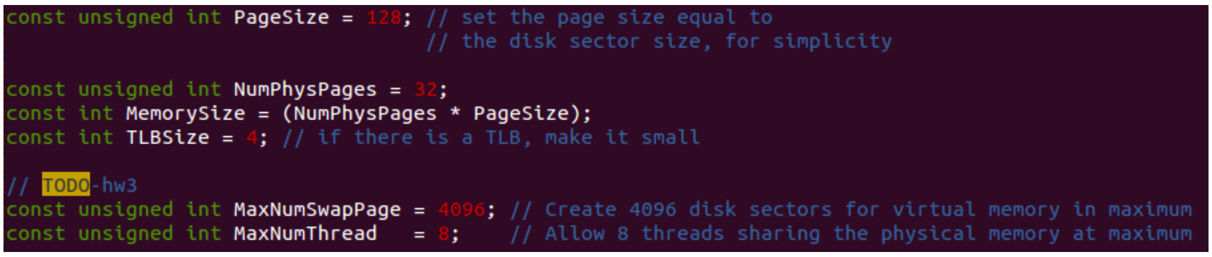
B. Implementation:

To implement virtual memory, I need to define a frame table for physical memory and virtual memory to keep track of their page's status. After that, I need to change the way of program loading to make use of virtual memory. Finally, I need to modify address translation, so that the kernel can deal with page faults when it happens and can swap pages between physical and virtual memory.

Therefore, I will explain my implementation in the following three step:

1. Define Frame Table, Swap Table, and Page Table

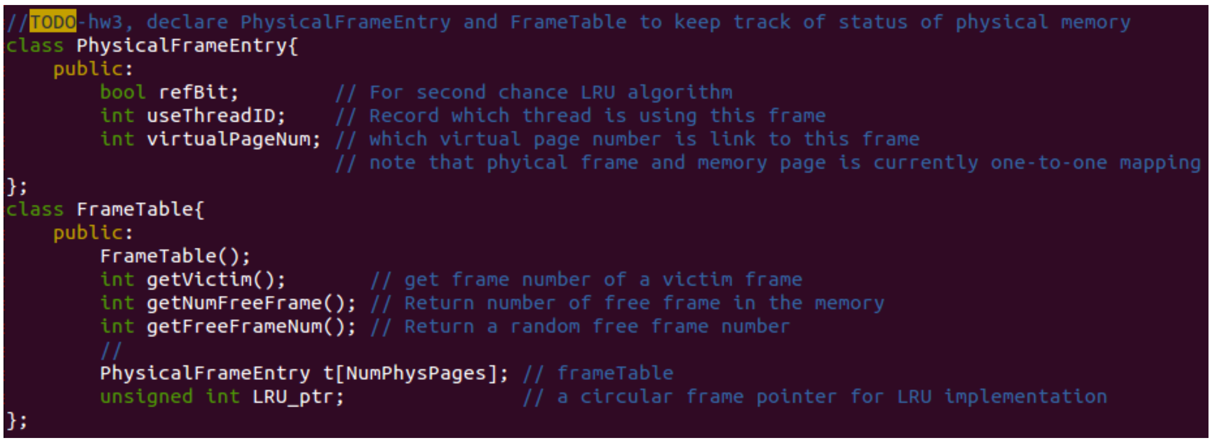
In machine/machine.h, I define the maximum disk sector that is used for virtual memory to be 4096, and the maximum threads to be 8.



```
const unsigned int PageSize = 128; // set the page size equal to
                                   // the disk sector size, for simplicity

const unsigned int NumPhysPages = 32;
const int MemorySize = (NumPhysPages * PageSize);
const int TLBSize = 4; // if there is a TLB, make it small

// TODO-hw3
const unsigned int MaxNumSwapPage = 4096; // Create 4096 disk sectors for virtual memory in maximum
const unsigned int MaxNumThread   = 8;    // Allow 8 threads sharing the physical memory at maximum
```

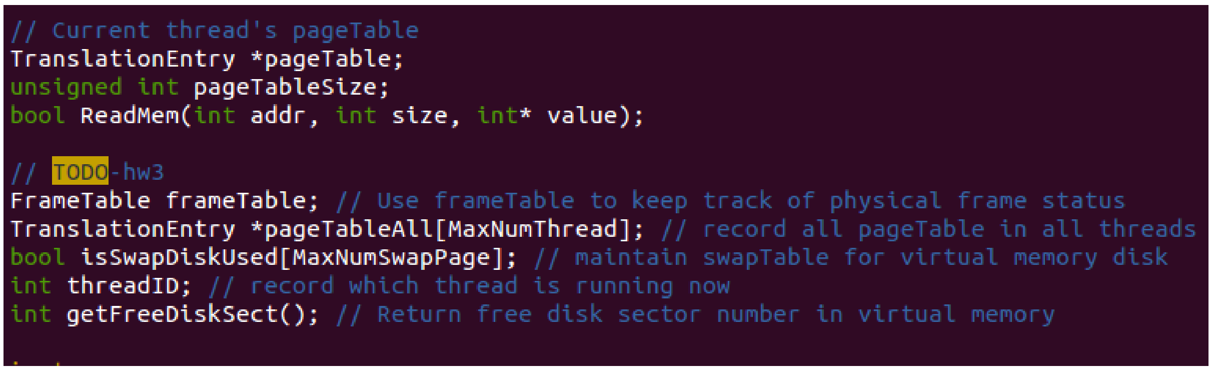
Later on, I created a FrameTable class to help me manage physical memory pages, while every entry is defined as a PhysicalFrameEntry class.

In PhysicalFrameEntry, refBit is used for the second chance LRU algorithm, which will be explained in detail later. useThreadID records which thread is using this frame rightnow, and virtualPageNum indicates the virtual address that points to this frame.



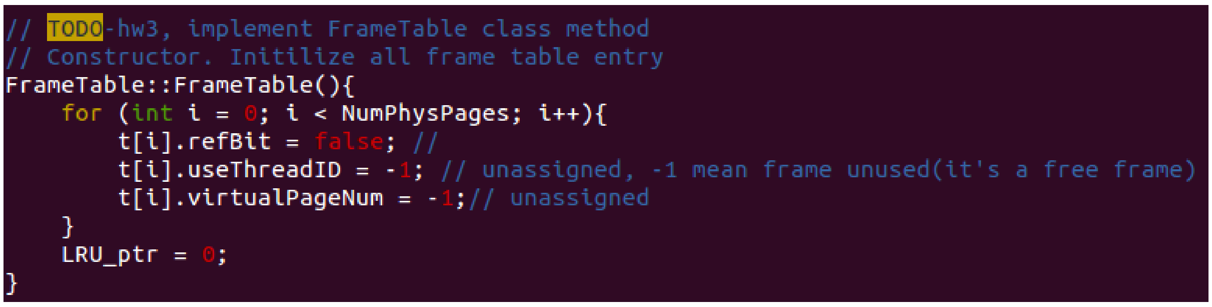
```
//TODO-hw3, declare PhysicalFrameEntry and FrameTable to keep track of status of physical memory
class PhysicalFrameEntry{
    public:
        bool refBit;        // For second chance LRU algorithm
        int useThreadID;    // Record which thread is using this frame
        int virtualPageNum; // which virtual page number is link to this frame
                            // note that phyical frame and memory page is currently one-to-one mapping
};
class FrameTable{
    public:
        FrameTable();
        int getVictim();       // get frame number of a victim frame
        int getNumFreeFrame(); // Return number of free frame in the memory
        int getFreeFrameNum(); // Return a random free frame number
        //
        PhysicalFrameEntry t[NumPhysPages]; // frameTable
        unsigned int LRU_ptr;               // a circular frame pointer for LRU implementation
};
```

I add those self-defined classes into Machine class as public objects so that every other classes in Nachos can access them.



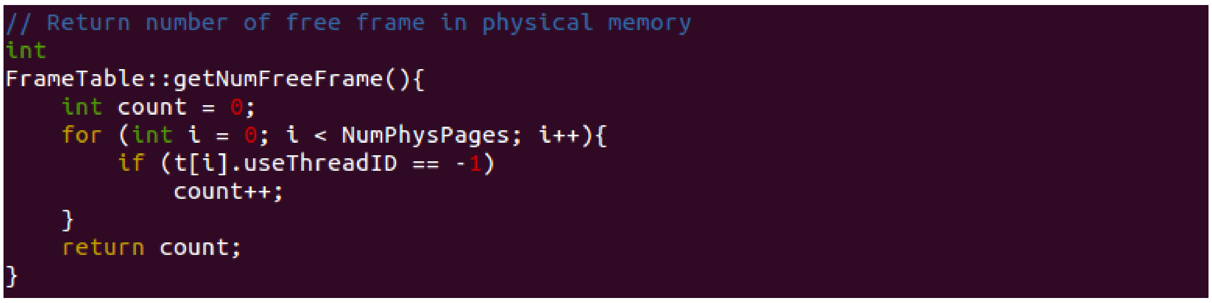
```
// Current thread's pageTable
TranslationEntry *pageTable;
unsigned int pageTableSize;
bool ReadMem(int addr, int size, int* value);

// TODO-hw3
FrameTable frameTable; // Use frameTable to keep track of physical frame status
TranslationEntry *pageTableAll[MaxNumThread]; // record all pageTable in all threads
bool isSwapDiskUsed[MaxNumSwapPage]; // maintain swapTable for virtual memory disk
int threadID; // record which thread is running now
int getFreeDiskSect(); // Return free disk sector number in virtual memory
```

In machine/machine.cc, I define the constructor of FrameTable and initialize every member variable.



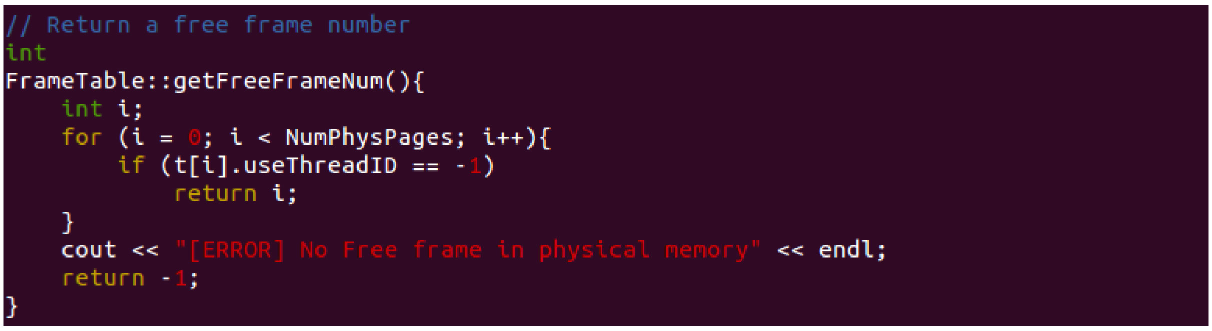
```
// TODO-hw3, implement FrameTable class method
// Constructor. Initilize all frame table entry
FrameTable::FrameTable(){
    for (int i = 0; i < NumPhysPages; i++){
        t[i].refBit = false; //
        t[i].useThreadID = -1; // unassigned, -1 mean frame unused(it's a free frame)
        t[i].virtualPageNum = -1;// unassigned
    }
    LRU_ptr = 0;
}
```

getNumFreeFrame calculates how many free pages are there in the physical memory. I implement this function by searching linearly in the memory.



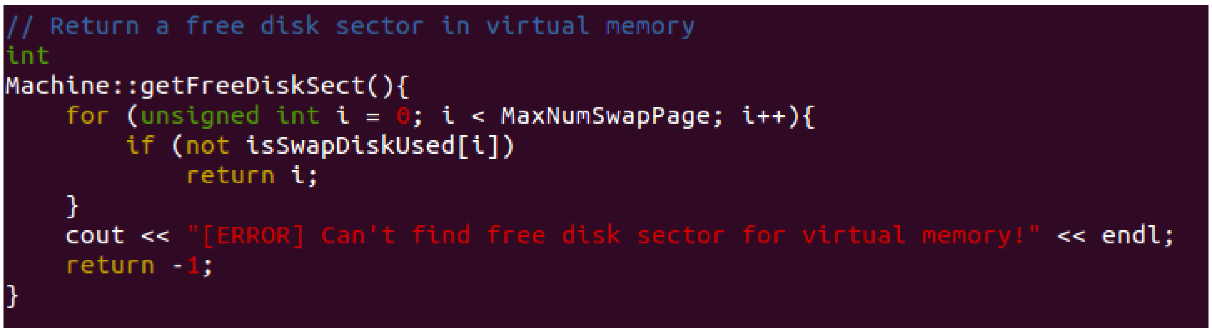
```
// Return number of free frame in physical memory
int
FrameTable::getNumFreeFrame(){
    int count = 0;
    for (int i = 0; i < NumPhysPages; i++){
        if (t[i].useThreadID == -1)
            count++;
    }
    return count;
}
```

getFreeFrameNum returns the frame number of a free frame.(not occupied by any process).



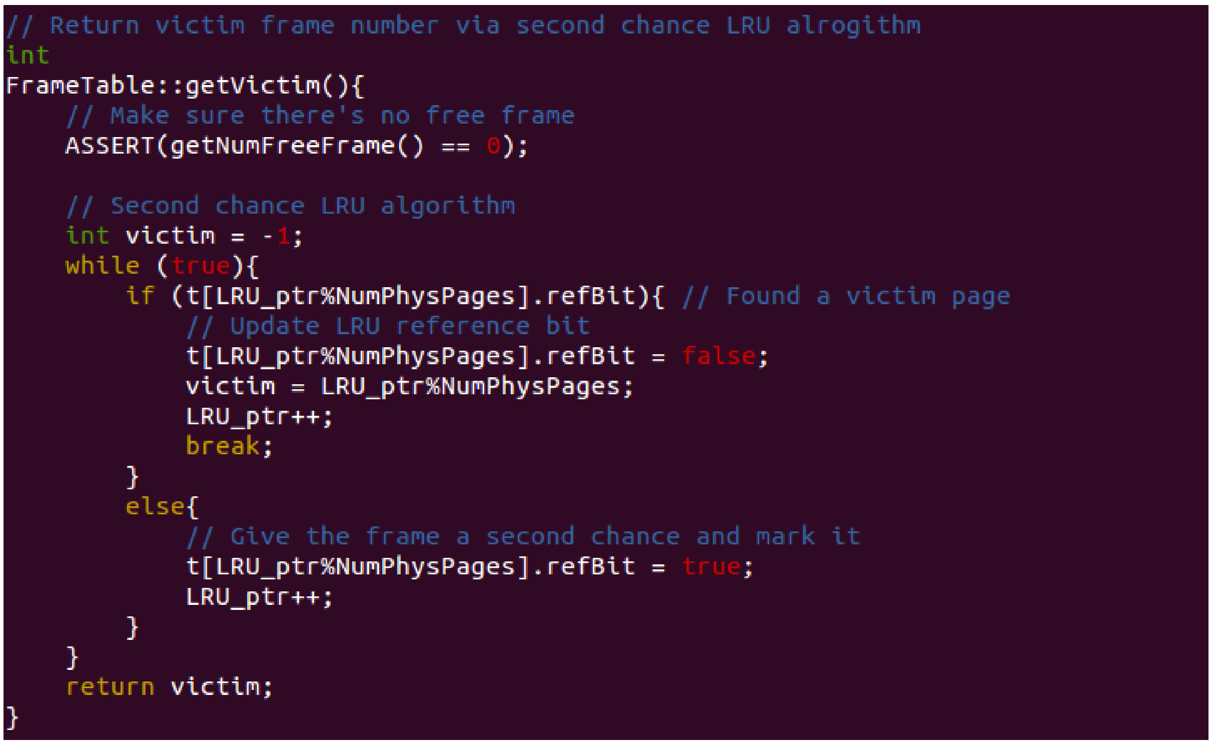
```
// Return a free frame number
int
FrameTable::getFreeFrameNum(){
    int i;
    for (i = 0; i < NumPhysPages; i++){
        if (t[i].useThreadID == -1)
            return i;
    }
    cout << "[ERROR] No Free frame in physical memory" << endl;
    return -1;
}
```

getFreeDiskSect returns the sector number of a free sector.



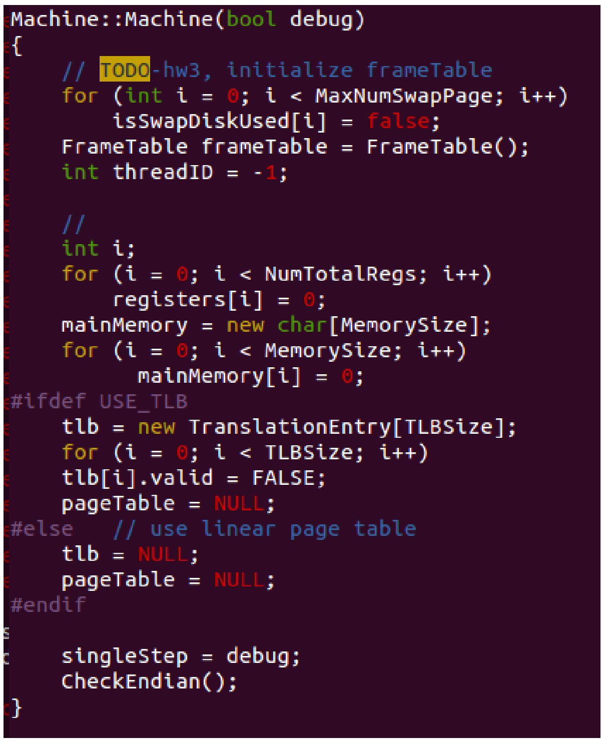
```
// Return a free disk sector in virtual memory
int
Machine::getFreeDiskSect(){
    for (unsigned int i = 0; i < MaxNumSwapPage; i++){
        if (not isSwapDiskUsed[i])
            return i;
    }
    cout << "[ERROR] Can't find free disk sector for virtual memory!" << endl;
    return -1;
}
```

In getVictime(), I need to find a victim frame and return its frame number. I implement the second chance algorithm to approximate LRU algorithm. The idea is to give the frame a second chance and switch on the reference bit when it first points by the circular pointer. After, if the circular pointer points to that victim, which has the reference point set to on, we will directly treat it as a victim frame and swap it. All in all, this algorithm finds victims in a Round-Robin fashion. It ensures fairness among all frames.



```
// Return victim frame number via second chance LRU alrogithm
int
FrameTable::getVictim(){
    // Make sure there's no free frame
    ASSERT(getNumFreeFrame() == 0);

    // Second chance LRU algorithm
    int victim = -1;
    while (true){
        if (t[LRU_ptr%NumPhysPages].refBit){ // Found a victim page
            // Update LRU reference bit
            t[LRU_ptr%NumPhysPages].refBit = false;
            victim = LRU_ptr%NumPhysPages;
            LRU_ptr++;
            break;
        }
        else{
            // Give the frame a second chance and mark it
            t[LRU_ptr%NumPhysPages].refBit = true;
            LRU_ptr++;
        }
    }
    return victim;
}
```

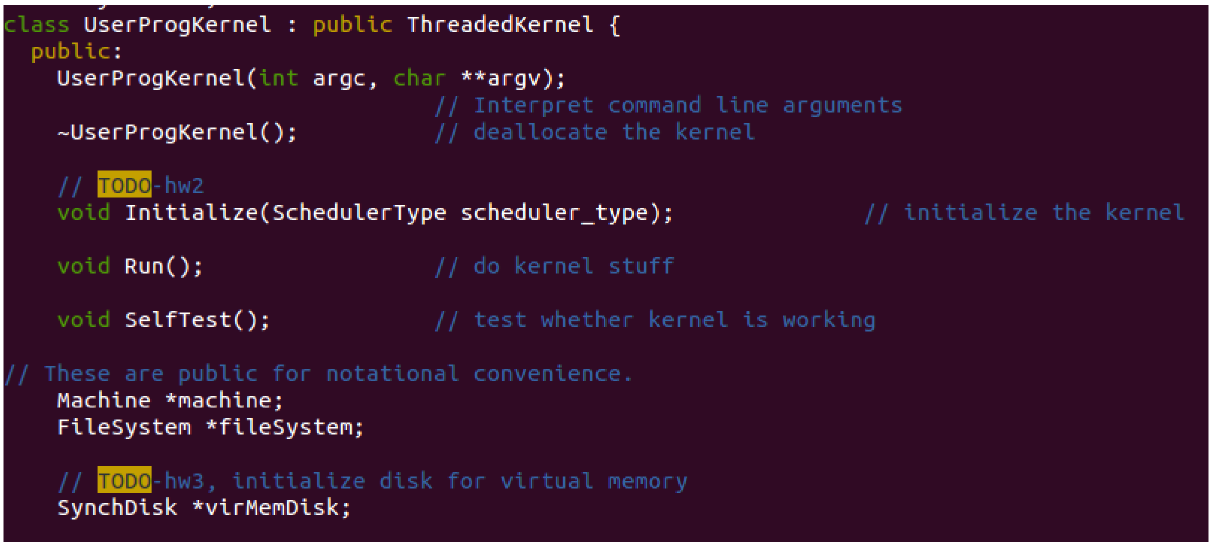
In machine/machine.cc, I need to modify the Machine constructor to initialize FrameTable and Swap Table.



```
Machine::Machine(bool debug)
{
    // TODO-hw3, initialize frameTable
    for (int i = 0; i < MaxNumSwapPage; i++)
        isSwapDiskUsed[i] = false;
    FrameTable frameTable = FrameTable();
    int threadID = -1;

    //
    int i;
    for (i = 0; i < NumTotalRegs; i++)
        registers[i] = 0;
    mainMemory = new char[MemorySize];
    for (i = 0; i < MemorySize; i++)
          mainMemory[i] = 0;
#ifdef USE_TLB
    tlb = new TranslationEntry[TLBSize];
    for (i = 0; i < TLBSize; i++)
    tlb[i].valid = FALSE;
    pageTable = NULL;
#else   // use linear page table
    tlb = NULL;
    pageTable = NULL;
#endif

    singleStep = debug;
    CheckEndian();
}
```

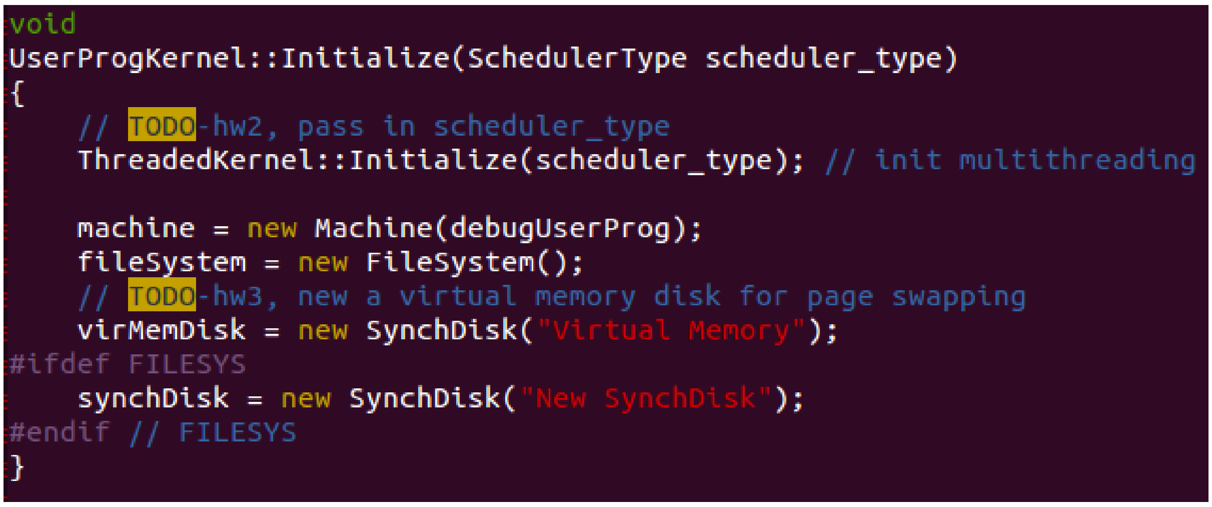
2. Enable Virtual Memory in Program Loading

In userprog/userkernel.h, we need to create a new disk dedicated for virtual memory; I use the class defined in disk.h and name it virMemDisk.



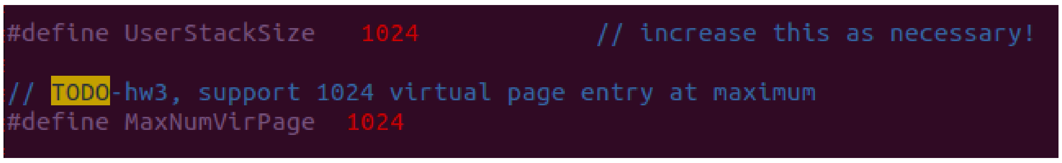
```
class UserProgKernel : public ThreadedKernel {
  public:
    UserProgKernel(int argc, char **argv);
                                // Interpret command line arguments
    ~UserProgKernel();          // deallocate the kernel

    // TODO-hw2
    void Initialize(SchedulerType scheduler_type);              // initialize the kernel

    void Run();                 // do kernel stuff

    void SelfTest();            // test whether kernel is working

// These are public for notational convenience.
    Machine *machine;
    FileSystem *fileSystem;

    // TODO-hw3, initialize disk for virtual memory
    SynchDisk *virMemDisk;
```

In userprog/userkernel.cc, I need to instanize virtual memory disk when the kernel starts.



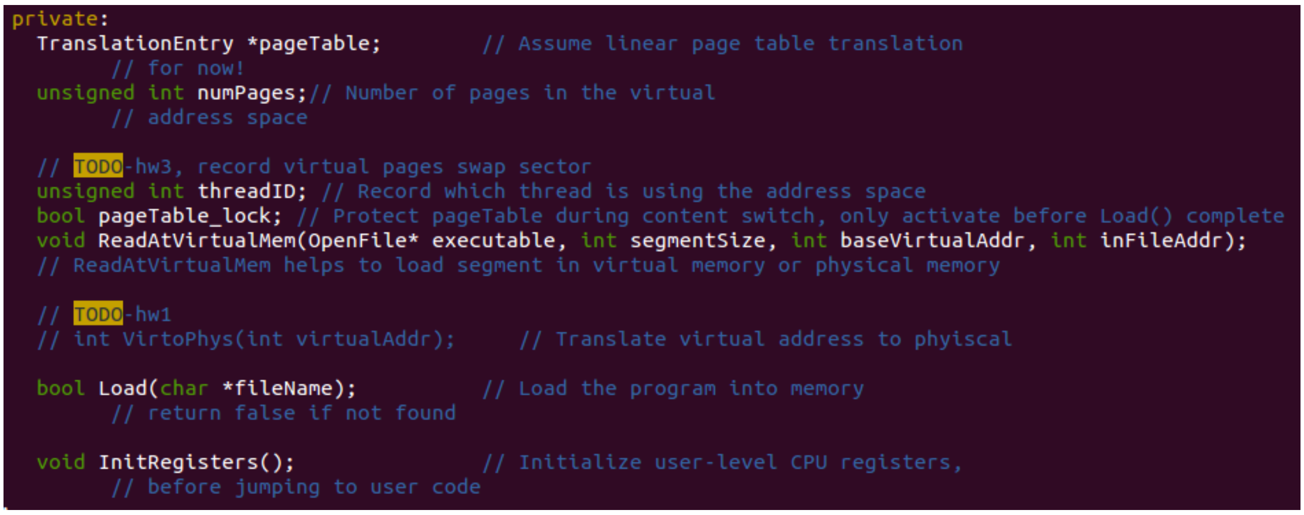
```
void
UserProgKernel::Initialize(SchedulerType scheduler_type)
{
    // TODO-hw2, pass in scheduler_type
    ThreadedKernel::Initialize(scheduler_type); // init multithreading

    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    // TODO-hw3, new a virtual memory disk for page swapping
    virMemDisk = new SynchDisk("Virtual Memory");
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}
```

In userprog/addrspace.h, I define virtual page entry to be 1024 at maximum



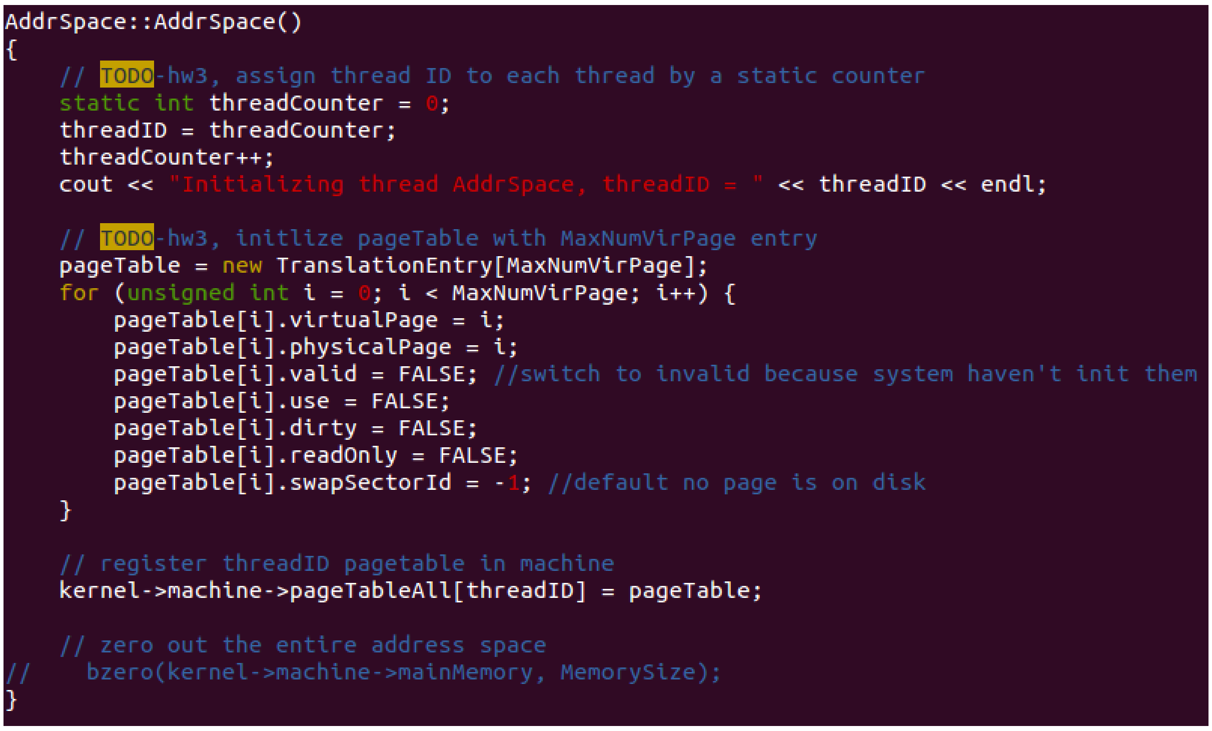
```
#define UserStackSize    1024            // increase this as necessary!

// TODO-hw3, support 1024 virtual page entry at maximum
#define MaxNumVirPage  1024
```

In userprog/addrspace.h, I need to define threadID to remember which thread uses this addrespace. pageTable_lock is used when a context switch happens. And ReadAtVirtualMem() is a function that loads program's segments into virtual memory.



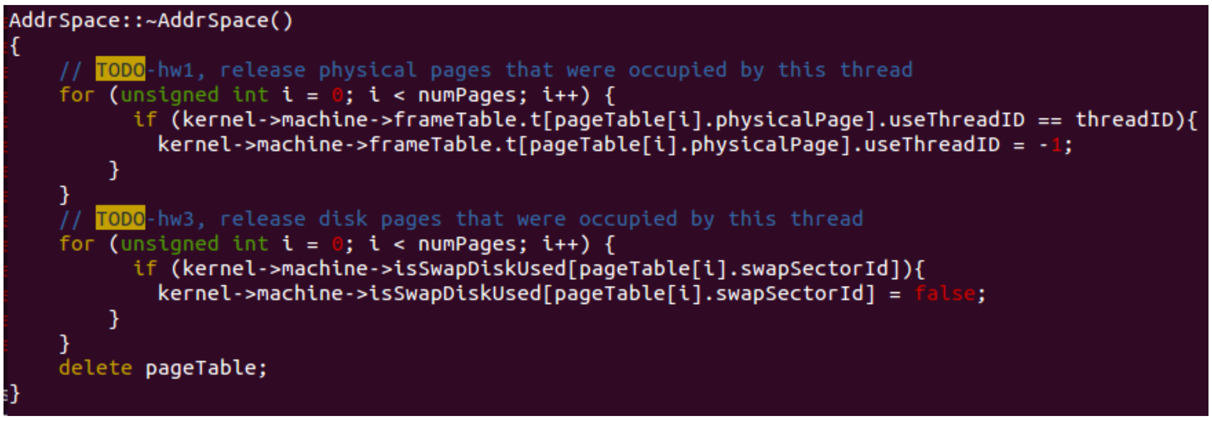
```
private:
  TranslationEntry *pageTable;        // Assume linear page table translation
        // for now!
  unsigned int numPages;// Number of pages in the virtual
        // address space

  // TODO-hw3, record virtual pages swap sector
  unsigned int threadID; // Record which thread is using the address space
  bool pageTable_lock; // Protect pageTable during content switch, only activate before Load() complete
  void ReadAtVirtualMem(OpenFile* executable, int segmentSize, int baseVirtualAddr, int inFileAddr);
  // ReadAtVirtualMem helps to load segment in virtual memory or physical memory

  // TODO-hw1
  // int VirtoPhys(int virtualAddr);     // Translate virtual address to phyiscal

  bool Load(char *fileName);          // Load the program into memory
        // return false if not found

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

In userprog/addrspace.cc, I add code below to initialize my data structure. Note that threadCount is a static variable; its purpose is to assign threadID. Aside from that, I instantiated page table entry here and set all entries to invalid.



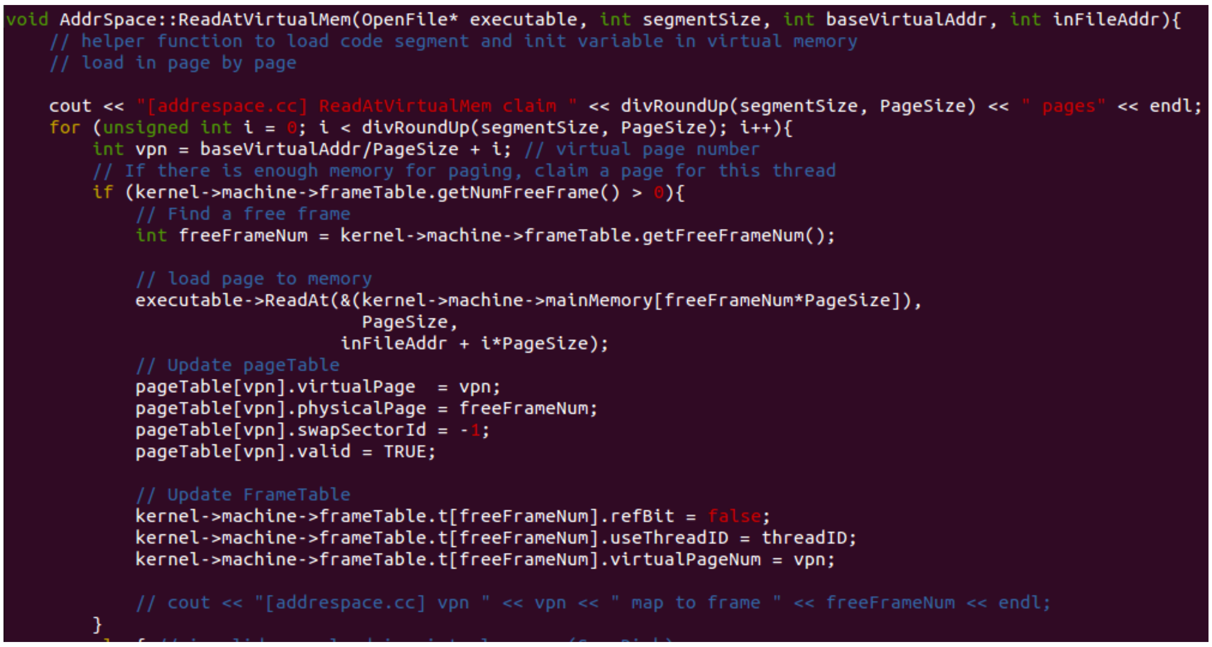
```
AddrSpace::AddrSpace()
{
    // TODO-hw3, assign thread ID to each thread by a static counter
    static int threadCounter = 0;
    threadID = threadCounter;
    threadCounter++;
    cout << "Initializing thread AddrSpace, threadID = " << threadID << endl;

    // TODO-hw3, initlize pageTable with MaxNumVirPage entry
    pageTable = new TranslationEntry[MaxNumVirPage];
    for (unsigned int i = 0; i < MaxNumVirPage; i++) {
        pageTable[i].virtualPage = i;
        pageTable[i].physicalPage = i;
        pageTable[i].valid = FALSE; //switch to invalid because system haven't init them
        pageTable[i].use = FALSE;
        pageTable[i].dirty = FALSE;
        pageTable[i].readOnly = FALSE;
        pageTable[i].swapSectorId = -1; //default no page is on disk
    }

    // register threadID pagetable in machine
    kernel->machine->pageTableAll[threadID] = pageTable;

    // zero out the entire address space
//    bzero(kernel->machine->mainMemory, MemorySize);
}
```

Because I will create content on disk rightnow, I need to release the claimed space when the process is terminated.



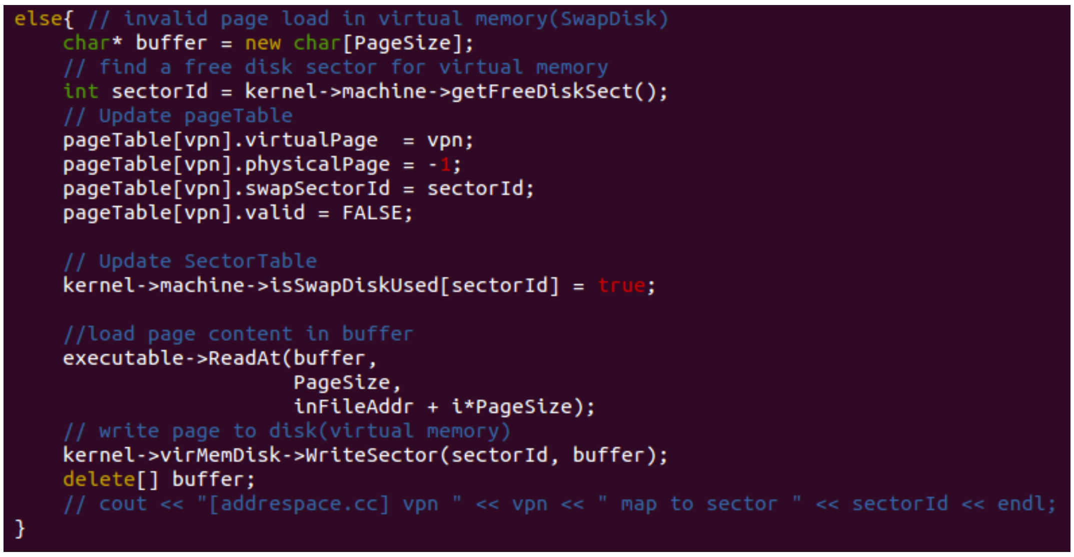
```
AddrSpace::~AddrSpace()
{
    // TODO-hw1, release physical pages that were occupied by this thread
    for (unsigned int i = 0; i < numPages; i++) {
          if (kernel->machine->frameTable.t[pageTable[i].physicalPage].useThreadID == threadID){
            kernel->machine->frameTable.t[pageTable[i].physicalPage].useThreadID = -1;
        }
    }
    // TODO-hw3, release disk pages that were occupied by this thread
    for (unsigned int i = 0; i < numPages; i++) {
          if (kernel->machine->isSwapDiskUsed[pageTable[i].swapSectorId]){
            kernel->machine->isSwapDiskUsed[pageTable[i].swapSectorId] = false;
        }
    }
    delete pageTable;
}
```

In ReadAtVirtualMem(), I implement a loading function that can utilize virtual memory. If there’s enough space in memory, just load in the program on a physical frame directly as shown below.



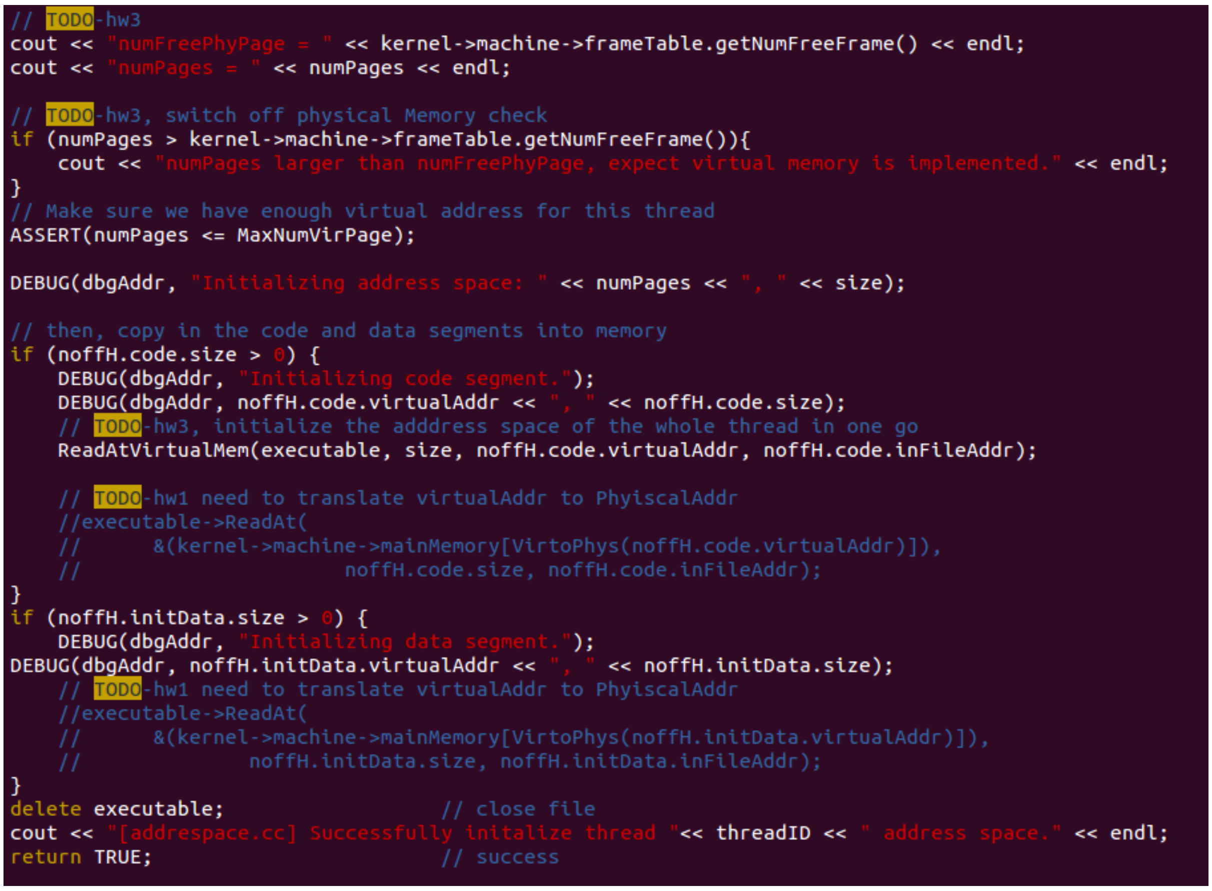
```
void AddrSpace::ReadAtVirtualMem(OpenFile* executable, int segmentSize, int baseVirtualAddr, int inFileAddr){
    // helper function to load code segment and init variable in virtual memory
    // load in page by page

    cout << "[addrespace.cc] ReadAtVirtualMem claim " << divRoundUp(segmentSize, PageSize) << " pages" << endl;
    for (unsigned int i = 0; i < divRoundUp(segmentSize, PageSize); i++){
        int vpn = baseVirtualAddr/PageSize + i; // virtual page number
        // If there is enough memory for paging, claim a page for this thread
        if (kernel->machine->frameTable.getNumFreeFrame() > 0){
            // Find a free frame
            int freeFrameNum = kernel->machine->frameTable.getFreeFrameNum();

            // load page to memory
            executable->ReadAt(&(kernel->machine->mainMemory[freeFrameNum*PageSize]),
                                 PageSize,
                               inFileAddr + i*PageSize);
            // Update pageTable
            pageTable[vpn].virtualPage  = vpn;
            pageTable[vpn].physicalPage = freeFrameNum;
            pageTable[vpn].swapSectorId = -1;
            pageTable[vpn].valid = TRUE;

            // Update FrameTable
            kernel->machine->frameTable.t[freeFrameNum].refBit = false;
            kernel->machine->frameTable.t[freeFrameNum].useThreadID = threadID;
            kernel->machine->frameTable.t[freeFrameNum].virtualPageNum = vpn;

            // cout << "[addrespace.cc] vpn " << vpn << " map to frame " << freeFrameNum << endl;
        }
```

However, if the memory is fully occupied, I need to allocate a disk sector to put the page into the disk sector as a virtual memory.



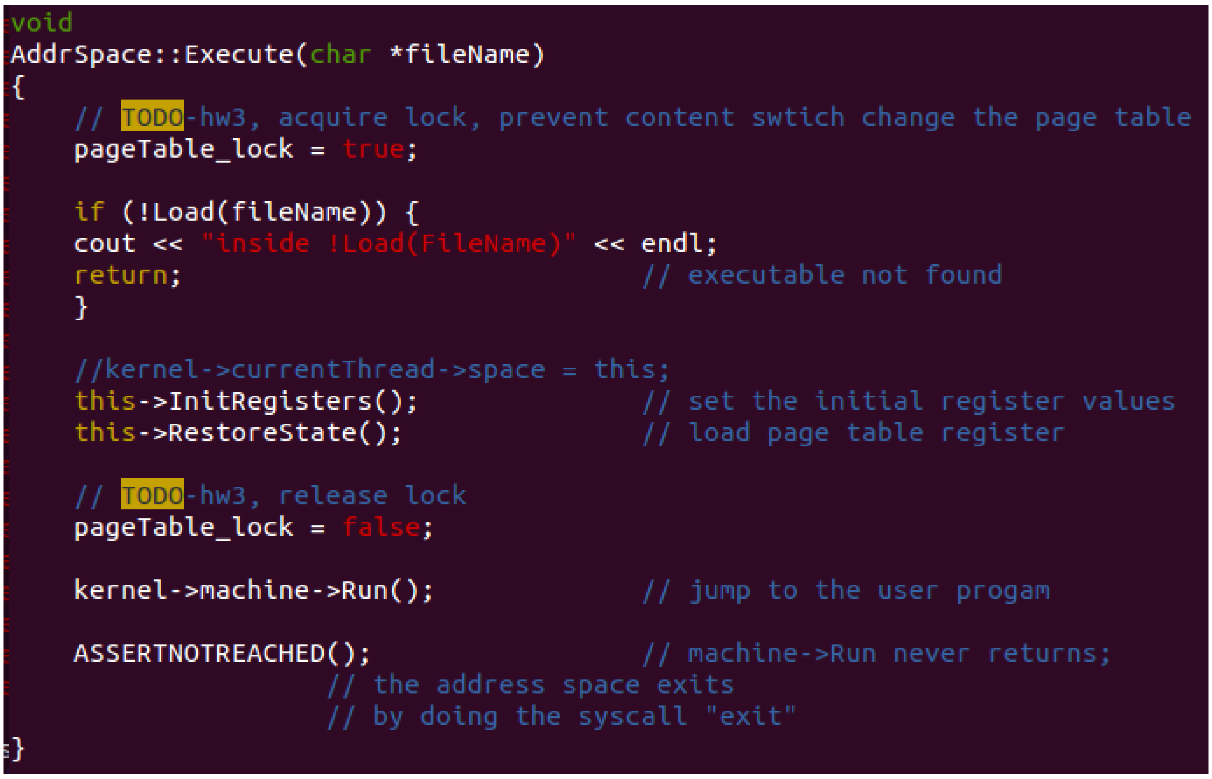
```
else{ // invalid page load in virtual memory(SwapDisk)
    char* buffer = new char[PageSize];
    // find a free disk sector for virtual memory
    int sectorId = kernel->machine->getFreeDiskSect();
    // Update pageTable
    pageTable[vpn].virtualPage  = vpn;
    pageTable[vpn].physicalPage = -1;
    pageTable[vpn].swapSectorId = sectorId;
    pageTable[vpn].valid = FALSE;

    // Update SectorTable
    kernel->machine->isSwapDiskUsed[sectorId] = true;

    //load page content in buffer
    executable->ReadAt(buffer,
                       PageSize,
                       inFileAddr + i*PageSize);
    // write page to disk(virtual memory)
    kernel->virMemDisk->WriteSector(sectorId, buffer);
    delete[] buffer;
    // cout << "[addrespace.cc] vpn " << vpn << " map to sector " << sectorId << endl;
}
```

In userprog/addrspace.cc, the code below shows the modification to load the program into virtual memory. Basically, I decide to load the code segment, initData, and uninitData all together in one go.



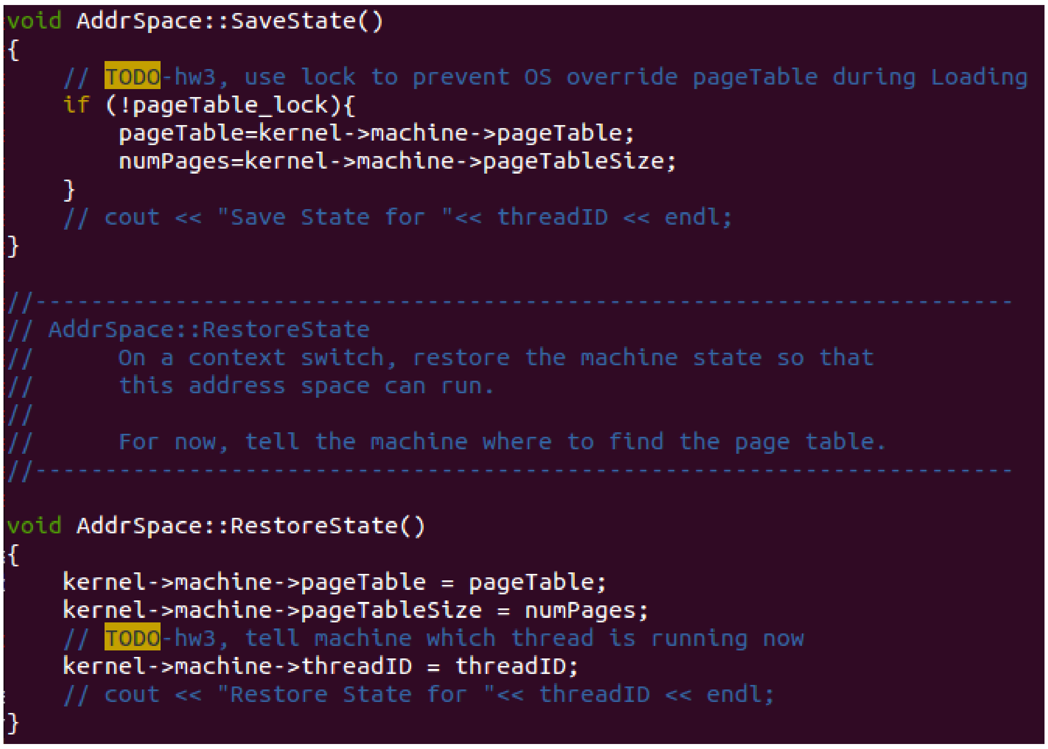
```
// TODO-hw3
cout << "numFreePhyPage = " << kernel->machine->frameTable.getNumFreeFrame() << endl;
cout << "numPages = " << numPages << endl;

// TODO-hw3, switch off physical Memory check
if (numPages > kernel->machine->frameTable.getNumFreeFrame()){
    cout << "numPages larger than numFreePhyPage, expect virtual memory is implemented." << endl;
}
// Make sure we have enough virtual address for this thread
ASSERT(numPages <= MaxNumVirPage);

DEBUG(dbgAddr, "Initializing address space: " << numPages << ", " << size);

// then, copy in the code and data segments into memory
if (noffH.code.size > 0) {
    DEBUG(dbgAddr, "Initializing code segment.");
    DEBUG(dbgAddr, noffH.code.virtualAddr << ", " << noffH.code.size);
    // TODO-hw3, initialize the adddress space of the whole thread in one go
    ReadAtVirtualMem(executable, size, noffH.code.virtualAddr, noffH.code.inFileAddr);

    // TODO-hw1 need to translate virtualAddr to PhyiscalAddr
    //executable->ReadAt(
    //      &(kernel->machine->mainMemory[VirtoPhys(noffH.code.virtualAddr)]),
    //                      noffH.code.size, noffH.code.inFileAddr);
}
if (noffH.initData.size > 0) {
    DEBUG(dbgAddr, "Initializing data segment.");
DEBUG(dbgAddr, noffH.initData.virtualAddr << ", " << noffH.initData.size);
    // TODO-hw1 need to translate virtualAddr to PhyiscalAddr
    //executable->ReadAt(
    //      &(kernel->machine->mainMemory[VirtoPhys(noffH.initData.virtualAddr)]),
    //              noffH.initData.size, noffH.initData.inFileAddr);
}
delete executable;                  // close file
cout << "[addrespace.cc] Successfully initalize thread "<< threadID << " address space." << endl;
return TRUE;                        // success
```

In userprog/addrspace.cc, I need to ensure the context switch won't change the page table that hasn't finished loading. Therefore, I use a simple lock to protect the Load() function.



```
void
AddrSpace::Execute(char *fileName)
{
    // TODO-hw3, acquire lock, prevent content swtich change the page table
    pageTable_lock = true;

    if (!Load(fileName)) {
    cout << "inside !Load(FileName)" << endl;
    return;                             // executable not found
    }

    //kernel->currentThread->space = this;
    this->InitRegisters();              // set the initial register values
    this->RestoreState();               // load page table register

    // TODO-hw3, release lock
    pageTable_lock = false;

    kernel->machine->Run();             // jump to the user progam

    ASSERTNOTREACHED();                 // machine->Run never returns;
                    // the address space exits
                    // by doing the syscall "exit"
}
```

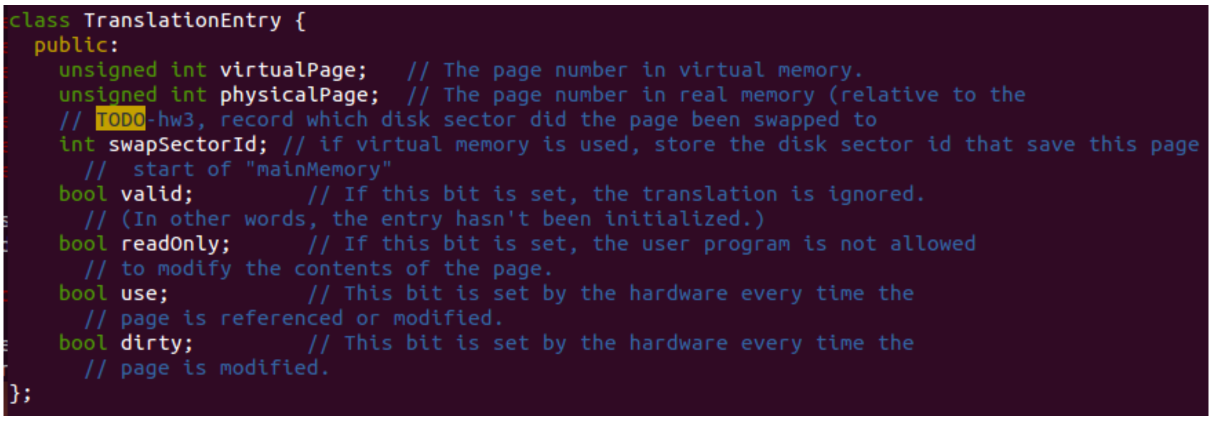
Aside from that, I also need to add code in SaveSate() and RestateState() to implement the page table lock.



```
void AddrSpace::SaveState()
{
    // TODO-hw3, use lock to prevent OS override pageTable during Loading
    if (!pageTable_lock){
        pageTable=kernel->machine->pageTable;
        numPages=kernel->machine->pageTableSize;
    }
    // cout << "Save State for "<< threadID << endl;
}

//----------------------------------------------------------------------
// AddrSpace::RestoreState
//      On a context switch, restore the machine state so that
//      this address space can run.
//
//      For now, tell the machine where to find the page table.
//----------------------------------------------------------------------

void AddrSpace::RestoreState()
{
    kernel->machine->pageTable = pageTable;
    kernel->machine->pageTableSize = numPages;
    // TODO-hw3, tell machine which thread is running now
    kernel->machine->threadID = threadID;
    // cout << "Restore State for "<< threadID << endl;
}
```

3. Allow address translation with virtual memory

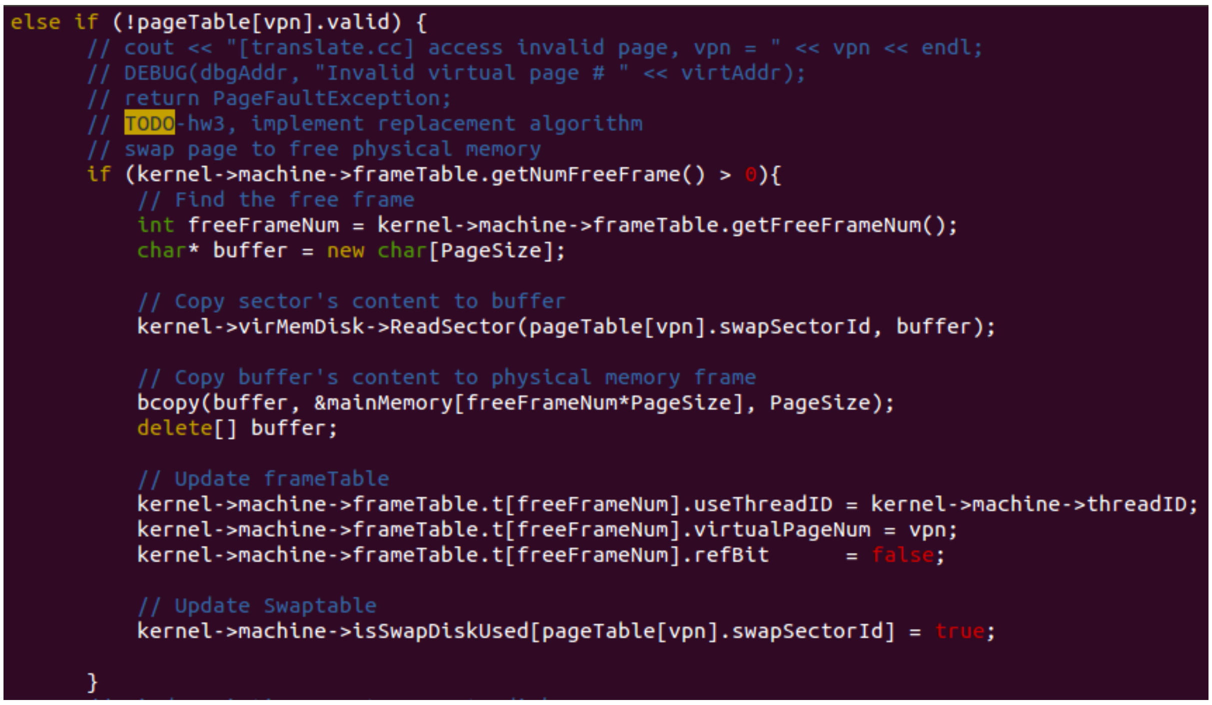
In machine/translate.h, I add a new member variable in the page table entry to record the sectorID that the virtual address points to.



```
class TranslationEntry {
  public:
    unsigned int virtualPage;   // The page number in virtual memory.
    unsigned int physicalPage;  // The page number in real memory (relative to the
    // TODO-hw3, record which disk sector did the page been swapped to
    int swapSectorId; // if virtual memory is used, store the disk sector id that save this page
      //  start of "mainMemory"
    bool valid;           // If this bit is set, the translation is ignored.
      // (In other words, the entry hasn't been initialized.)
    bool readOnly;        // If this bit is set, the user program is not allowed
      // to modify the contents of the page.
    bool use;             // This bit is set by the hardware every time the
      // page is referenced or modified.
    bool dirty;           // This bit is set by the hardware every time the
      // page is modified.
};
```

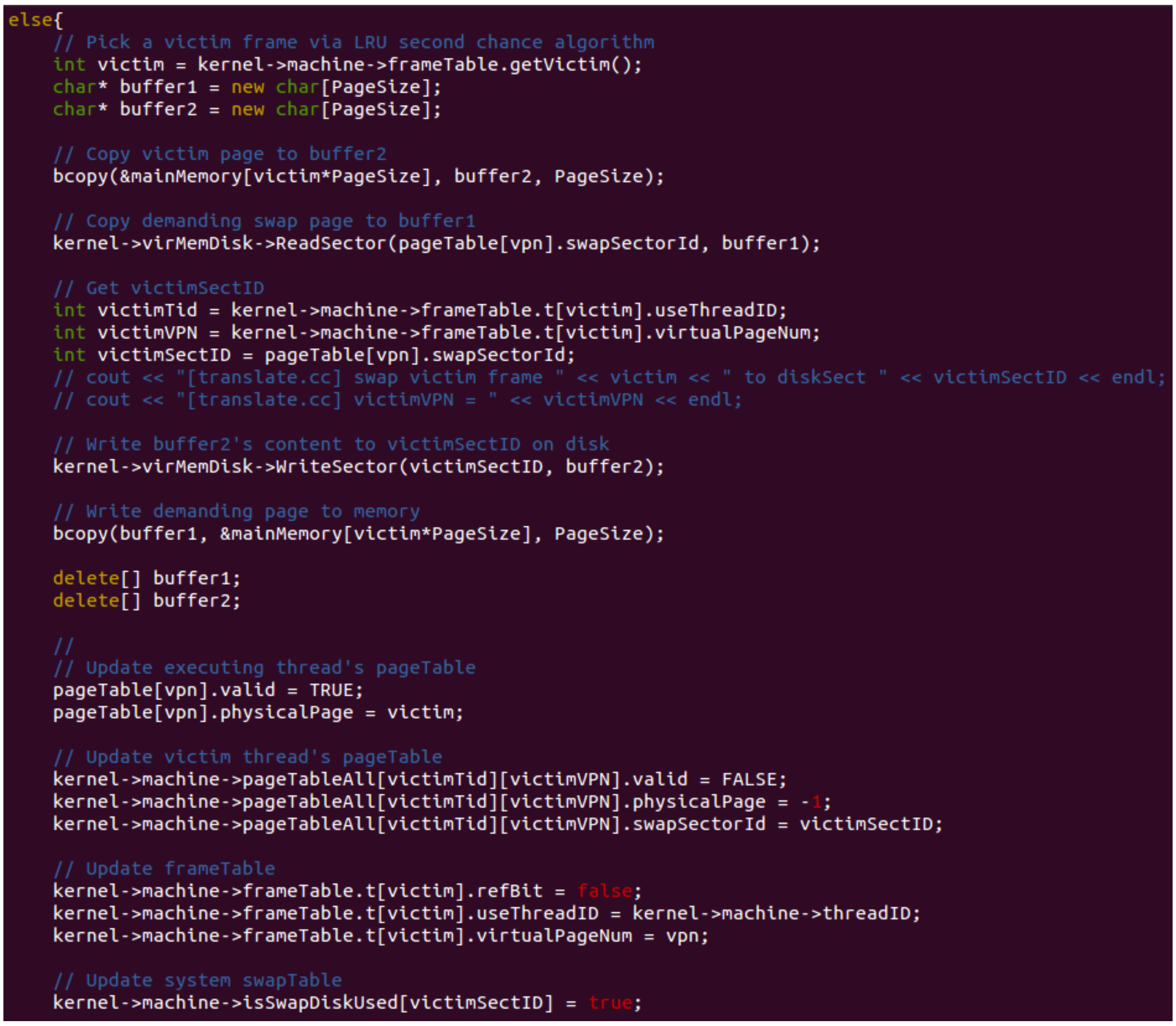
In machine/translate.cc, I need to deal with page faults. When the kernel wants to translate a virtual page that its valid bit is off, it'll execute the following code.

There are two different situations. Either the physical memory is full or not. For the latter case, it's easier to deal with. I simply read the disk sector and copy the content to a free memory frame and we're done.



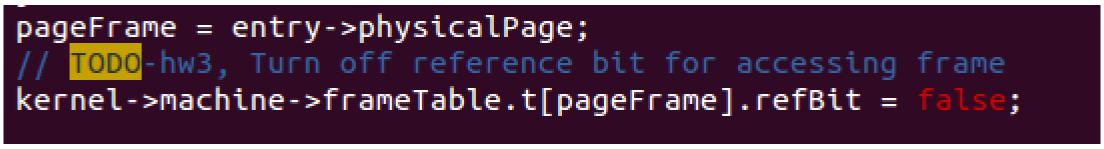
```
else if (!pageTable[vpn].valid) {
      // cout << "[translate.cc] access invalid page, vpn = " << vpn << endl;
      // DEBUG(dbgAddr, "Invalid virtual page # " << virtAddr);
      // return PageFaultException;
      // TODO-hw3, implement replacement algorithm
      // swap page to free physical memory
      if (kernel->machine->frameTable.getNumFreeFrame() > 0){
          // Find the free frame
          int freeFrameNum = kernel->machine->frameTable.getFreeFrameNum();
          char* buffer = new char[PageSize];

          // Copy sector's content to buffer
          kernel->virMemDisk->ReadSector(pageTable[vpn].swapSectorId, buffer);

          // Copy buffer's content to physical memory frame
          bcopy(buffer, &mainMemory[freeFrameNum*PageSize], PageSize);
          delete[] buffer;

          // Update frameTable
          kernel->machine->frameTable.t[freeFrameNum].useThreadID = kernel->machine->threadID;
          kernel->machine->frameTable.t[freeFrameNum].virtualPageNum = vpn;
          kernel->machine->frameTable.t[freeFrameNum].refBit      = false;

          // Update Swaptable
          kernel->machine->isSwapDiskUsed[pageTable[vpn].swapSectorId] = true;

      }
```

However, if the memory is full, then it's much more complicated. I need to choose a victim frame, put it to virtual memory, read the disk sector, and copy the required page back to memory. After all operations are done, I also need to update the new page status in the page table, frame table, and swap table.



```
else{
    // Pick a victim frame via LRU second chance algorithm
    int victim = kernel->machine->frameTable.getVictim();
    char* buffer1 = new char[PageSize];
    char* buffer2 = new char[PageSize];

    // Copy victim page to buffer2
    bcopy(&mainMemory[victim*PageSize], buffer2, PageSize);

    // Copy demanding swap page to buffer1
    kernel->virMemDisk->ReadSector(pageTable[vpn].swapSectorId, buffer1);

    // Get victimSectID
    int victimTid = kernel->machine->frameTable.t[victim].useThreadID;
    int victimVPN = kernel->machine->frameTable.t[victim].virtualPageNum;
    int victimSectID = pageTable[vpn].swapSectorId;
    // cout << "[translate.cc] swap victim frame " << victim << " to diskSect " << victimSectID << endl;
    // cout << "[translate.cc] victimVPN = " << victimVPN << endl;

    // Write buffer2's content to victimSectID on disk
    kernel->virMemDisk->WriteSector(victimSectID, buffer2);

    // Write demanding page to memory
    bcopy(buffer1, &mainMemory[victim*PageSize], PageSize);

    delete[] buffer1;
    delete[] buffer2;

    //
    // Update executing thread's pageTable
    pageTable[vpn].valid = TRUE;
    pageTable[vpn].physicalPage = victim;

    // Update victim thread's pageTable
    kernel->machine->pageTableAll[victimTid][victimVPN].valid = FALSE;
    kernel->machine->pageTableAll[victimTid][victimVPN].physicalPage = -1;
    kernel->machine->pageTableAll[victimTid][victimVPN].swapSectorId = victimSectID;

    // Update frameTable
    kernel->machine->frameTable.t[victim].refBit = false;
    kernel->machine->frameTable.t[victim].useThreadID = kernel->machine->threadID;
    kernel->machine->frameTable.t[victim].virtualPageNum = vpn;

    // Update system swapTable
    kernel->machine->isSwapDiskUsed[victimSectID] = true;
```

Finally, I need to switch the reference bit to off when the kernel accesses the frame; however, this is just for second chance algorithm implementation. Even if I don't add this line, the virtual memory will still work.

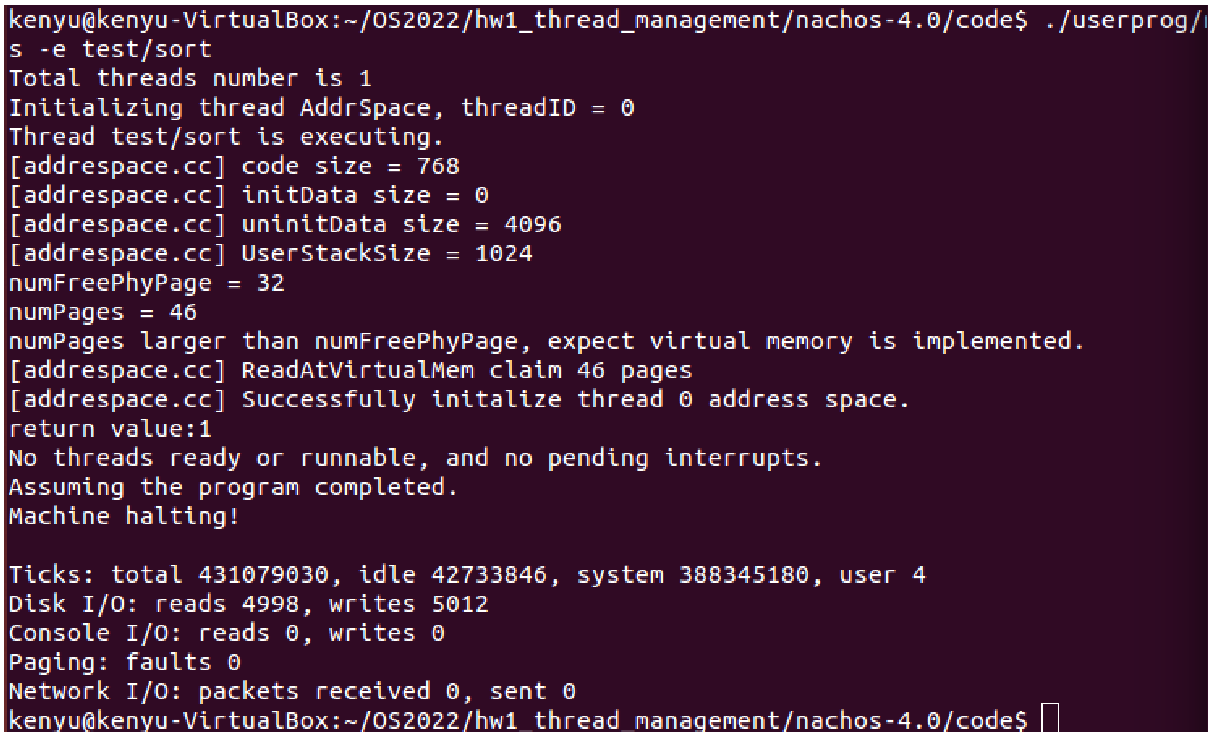


```
pageFrame = entry->physicalPage;
// TODO-hw3, Turn off reference bit for accessing frame
kernel->machine->frameTable.t[pageFrame].refBit = false;
```

C. Result

I execute sort.c alone and the return value is one. This is the correct answer.

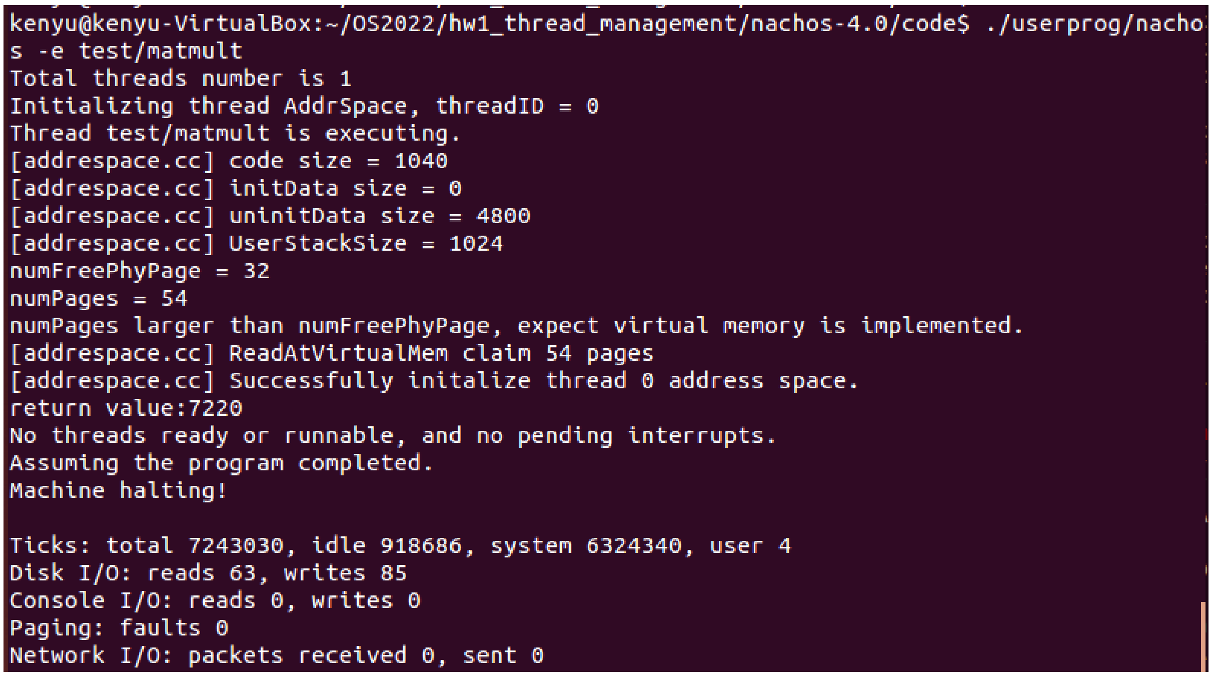
command I run: ./userprog/nachos -e test/sort



```
kenyu@kenyu-VirtualBox:~/OS2022/hw1_thread_management/nachos-4.0/code$ ./userprog/
s -e test/sort
Total threads number is 1
Initializing thread AddrSpace, threadID = 0
Thread test/sort is executing.
[addrespace.cc] code size = 768
[addrespace.cc] initData size = 0
[addrespace.cc] uninitData size = 4096
[addrespace.cc] UserStackSize = 1024
numFreePhyPage = 32
numPages = 46
numPages larger than numFreePhyPage, expect virtual memory is implemented.
[addrespace.cc] ReadAtVirtualMem claim 46 pages
[addrespace.cc] Successfully initalize thread 0 address space.
return value:1
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 431079030, idle 42733846, system 388345180, user 4
Disk I/O: reads 4998, writes 5012
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
kenyu@kenyu-VirtualBox:~/OS2022/hw1_thread_management/nachos-4.0/code$
```

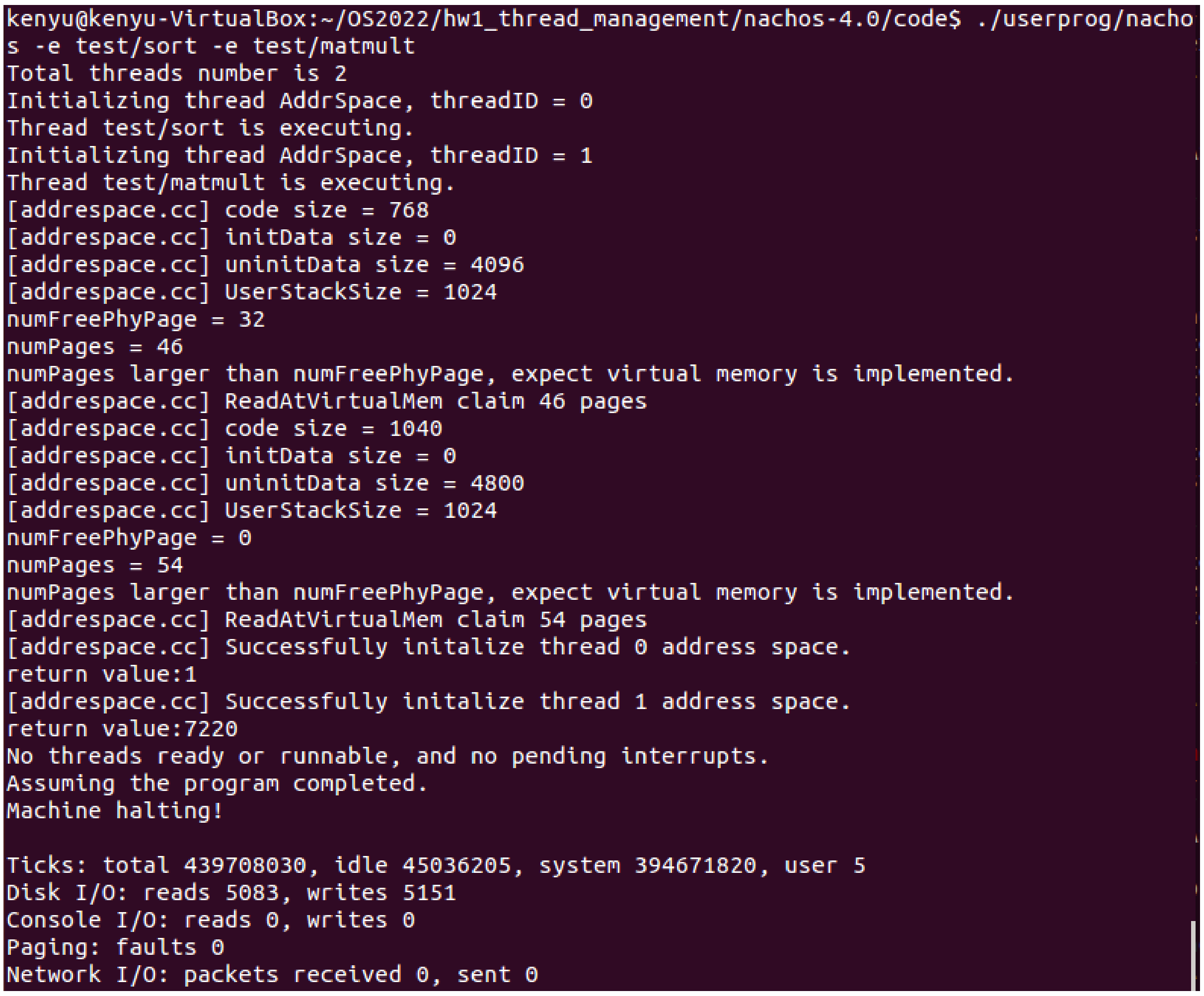
I execute matmult.c alone and the return value is 7220. This is the correct answer.

command I run : ./userprog/nachos -e test/matmult



```
kenyu@kenyu-VirtualBox:~/OS2022/hw1_thread_management/nachos-4.0/code$ ./userprog/nacho
s -e test/matmult
Total threads number is 1
Initializing thread AddrSpace, threadID = 0
Thread test/matmult is executing.
[addrespace.cc] code size = 1040
[addrespace.cc] initData size = 0
[addrespace.cc] uninitData size = 4800
[addrespace.cc] UserStackSize = 1024
numFreePhyPage = 32
numPages = 54
numPages larger than numFreePhyPage, expect virtual memory is implemented.
[addrespace.cc] ReadAtVirtualMem claim 54 pages
[addrespace.cc] Successfully initalize thread 0 address space.
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 7243030, idle 918686, system 6324340, user 4
Disk I/O: reads 63, writes 85
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

I execute sort.c and matmult.c concurrently and the results are still correct. This result proves my implementation of virtual memory is correct as well.
command I run: ./userprog/nachos -e test/sort -e test/matmult



```
kenyu@kenyu-VirtualBox:~/OS2022/hw1_thread_management/nachos-4.0/code$ ./userprog/nacho
s -e test/sort -e test/matmult
Total threads number is 2
Initializing thread AddrSpace, threadID = 0
Thread test/sort is executing.
Initializing thread AddrSpace, threadID = 1
Thread test/matmult is executing.
[addrespace.cc] code size = 768
[addrespace.cc] initData size = 0
[addrespace.cc] uninitData size = 4096
[addrespace.cc] UserStackSize = 1024
numFreePhyPage = 32
numPages = 46
numPages larger than numFreePhyPage, expect virtual memory is implemented.
[addrespace.cc] ReadAtVirtualMem claim 46 pages
[addrespace.cc] code size = 1040
[addrespace.cc] initData size = 0
[addrespace.cc] uninitData size = 4800
[addrespace.cc] UserStackSize = 1024
numFreePhyPage = 0
numPages = 54
numPages larger than numFreePhyPage, expect virtual memory is implemented.
[addrespace.cc] ReadAtVirtualMem claim 54 pages
[addrespace.cc] Successfully initalize thread 0 address space.
return value:1
[addrespace.cc] Successfully initalize thread 1 address space.
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 439708030, idle 45036205, system 394671820, user 5
Disk I/O: reads 5083, writes 5151
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

D. Difficult I have Encountered

The most difficult bug I have encountered in this project is the context switch bug. It happens when I haven't finished initializing the page table and an interrupt happens to the kernel. Then, the context switch function will overwrite my half-initialized page table and incur an unexpected result. This bug is extremely hard to spot because it's hard to find out when the context switch happens. I ended up tracing code line by line to identify this problem. But I can imagine if this is the real Operating System I am dealing with, the bug will be even harder to find.