Nachos Project 3

B08505048 工海四 劉名凱

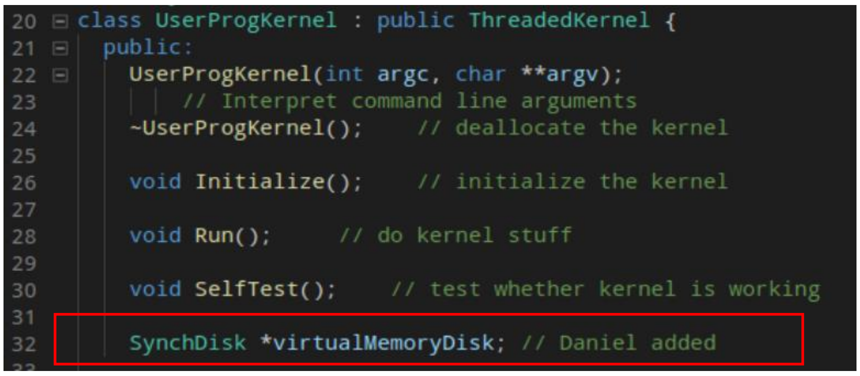
1. Motivation

Unlike project 1 and 2, task in project 3 require much more memory space. Therefore, we need to implement page replacement, swap out/in frames to acquire enough memory space and thus enable program finish successfully.Besides, project 3 also require multi-programming implemented in project 1. Only combining multi-programming and page replacement can we tackle down project 3.

The first thing to do is to decide where to deal with page fault. I choose to handle page fault whenever a page is needed and is not loaded into memory, avoiding producing any pageFaultException. Page replacement also need a disk to swap in/out frames. I follow the hint in the TA's pdf, using Synchdisk as a virtual disk. Finally, we only need to implement how page was replaced when a page fault occurs.

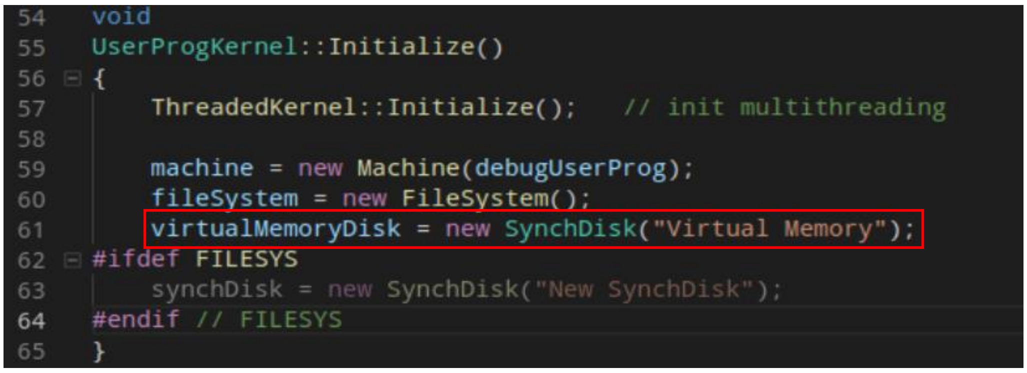
2. Implementation

First, add a SynchDisk in userkernel.h to act as a virtual disk, so that we can swap frames into/from the disk.



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

    void Initialize();    // initialize the kernel

    void Run();      // do kernel stuff

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

    SynchDisk *virtualMemoryDisk; // Daniel added
```

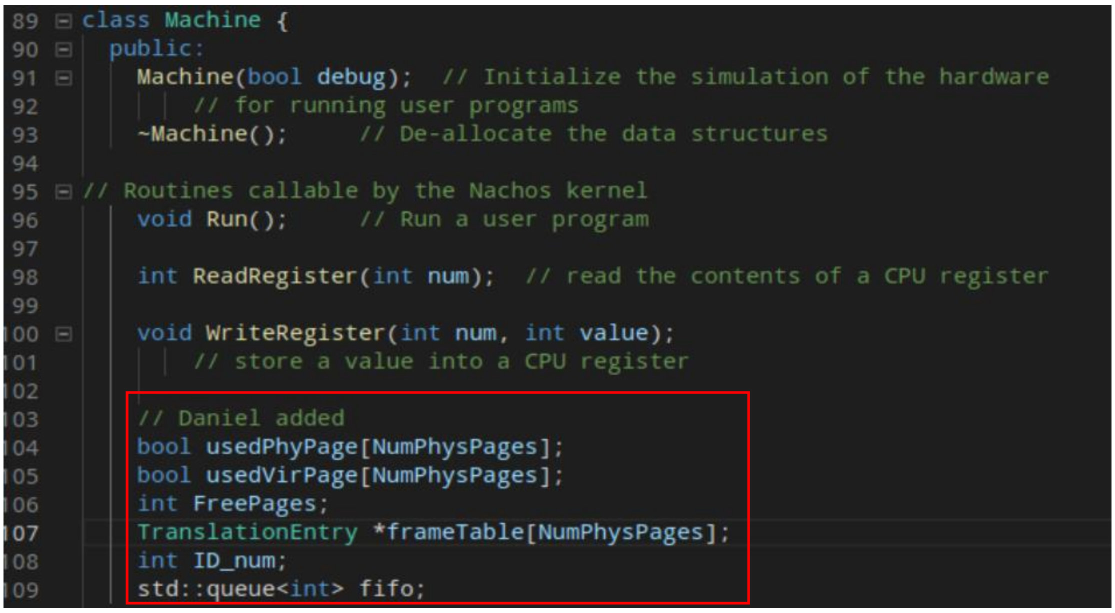
Also remember to initialize the SynchDisk in userkernel.cc



```
void
UserProgKernel::Initialize()
{
    ThreadedKernel::Initialize();   // init multithreading

    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    virtualMemoryDisk = new SynchDisk("Virtual Memory");
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}
```

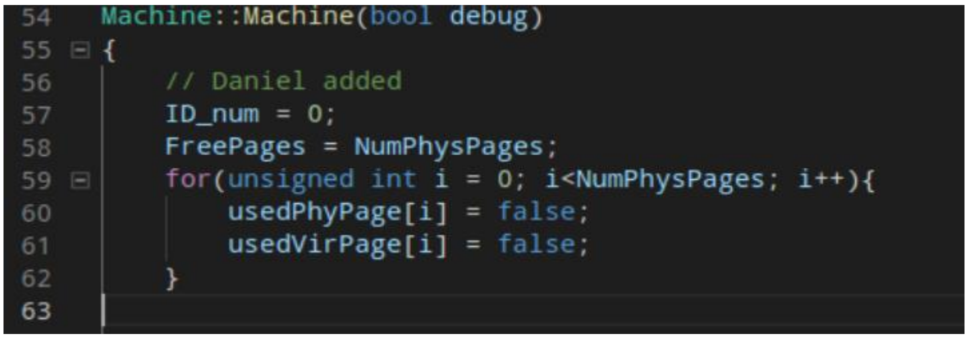
Add variable in machine.h:

usedPhyPage:	Record the usage of main memory
usedVirPage:	Record the usage of SynchDisk (virtual disk)
FreePages:	Record how many free frames in main memory
frameTable:	Record each frame's (in main memory) corresponding page (in each thread)
ID_num:	Record how many threads there are in one process
fifo:	First in first out queue, recording the sequence during allocating frames in main memory



```
class Machine {
  public:
    Machine(bool debug);  // Initialize the simulation of the hardware
        // for running user programs
    ~Machine();     // De-allocate the data structures

// Routines callable by the Nachos kernel
    void Run();     // Run a user program

    int ReadRegister(int num);  // read the contents of a CPU register

    void WriteRegister(int num, int value);
        // store a value into a CPU register

    // Daniel added
    bool usedPhyPage[NumPhysPages];
    bool usedVirPage[NumPhysPages];
    int FreePages;
    TranslationEntry *frameTable[NumPhysPages];
    int ID_num;
    std::queue<int> fifo;
```

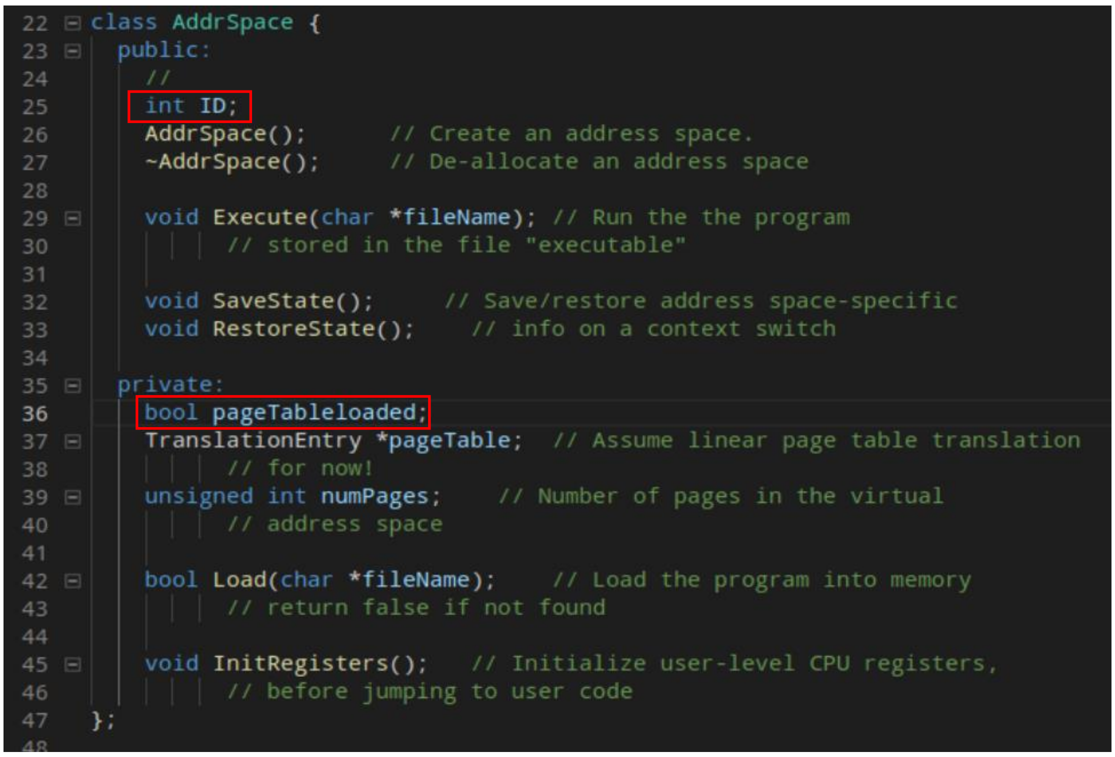
Initialize variables in machine.cc



```
Machine::Machine(bool debug)
{
    // Daniel added
    ID_num = 0;
    FreePages = NumPhysPages;
    for(unsigned int i = 0; i<NumPhysPages; i++){
        usedPhyPage[i] = false;
        usedVirPage[i] = false;
    }

```

Add variable in addrspace.h:

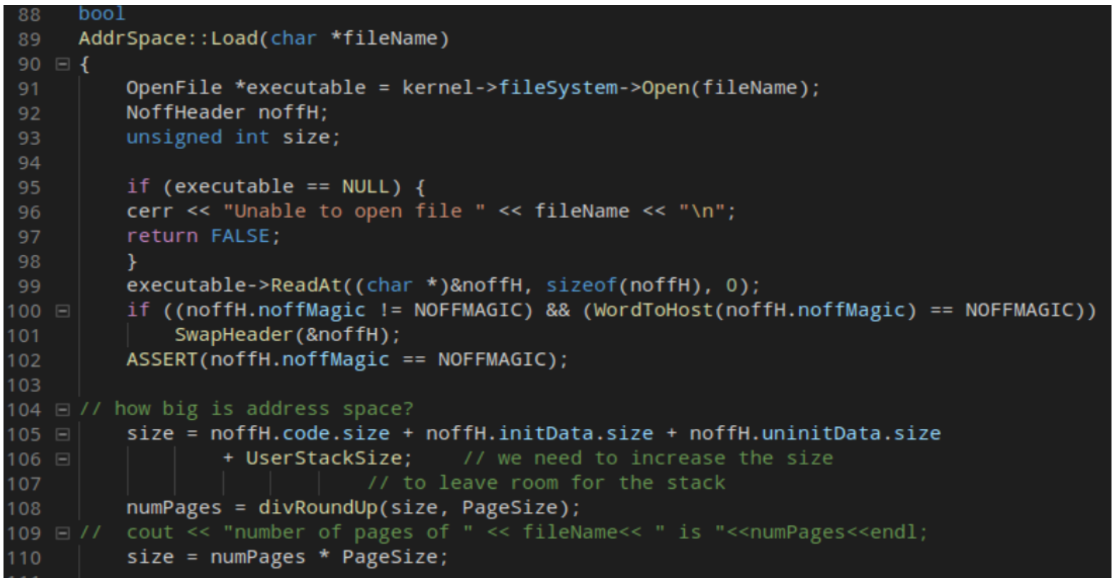
ID:	Thread's id
pageTableloaded:	check whether pageTable is fully loaded



```
class AddrSpace {
  public:
    //
    int ID;
    AddrSpace();        // Create an address space.
    ~AddrSpace();       // De-allocate an address space

    void Execute(char *fileName); // Run the the program
          // stored in the file "executable"

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

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

    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
};
```

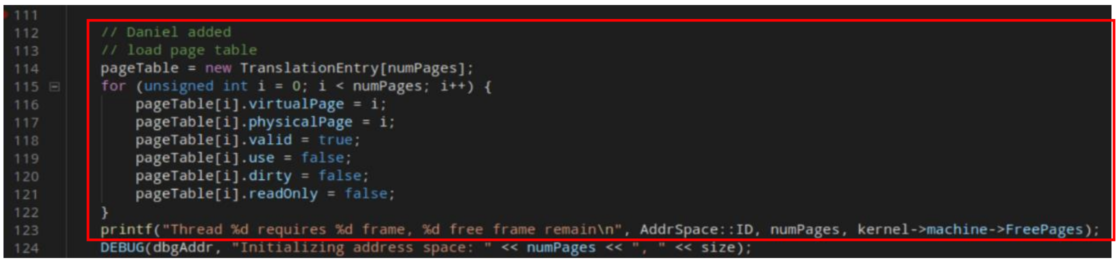
Next step is to make sure when loading thread's address space, thread can run successfully even when main memory is full. To do this, we need to modify code in addrspace::Load which control the process of loading address space.

1. Remove ASSERTION(numPages <= FreePages) since we can deal with page fault now.



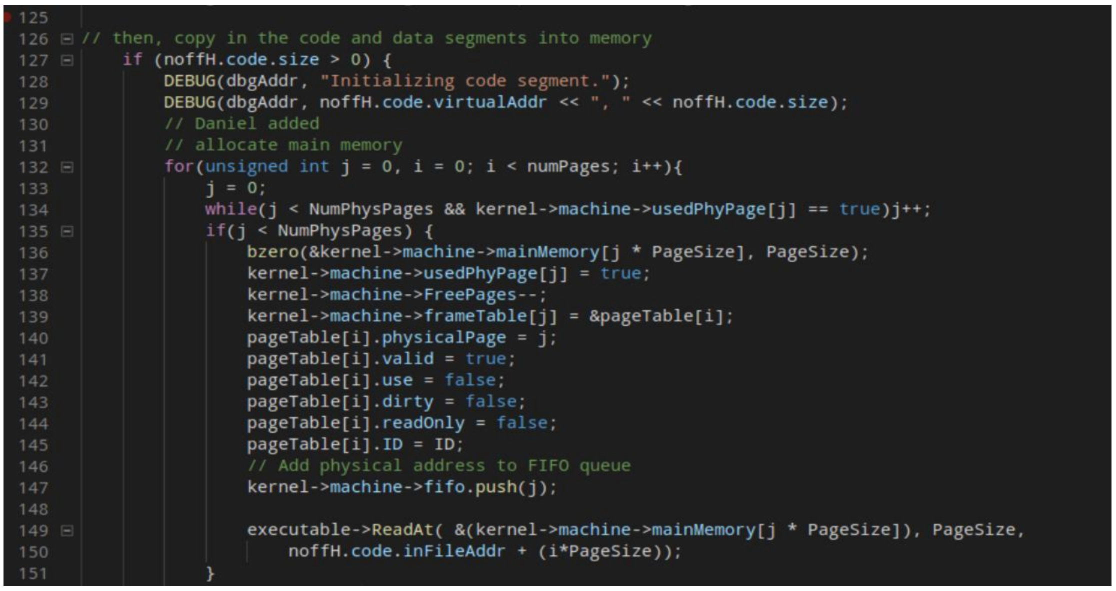
```
bool
AddrSpace::Load(char *fileName)
{
    OpenFile *executable = kernel->fileSystem->Open(fileName);
    NoffHeader noffH;
    unsigned int size;

    if (executable == NULL) {
    cerr << "Unable to open file " << fileName << "\n";
    return FALSE;
    }
    executable->ReadAt((char *)&noffH, sizeof(noffH), 0);
    if ((noffH.noffMagic != NOFFMAGIC) && (WordToHost(noffH.noffMagic) == NOFFMAGIC))
        SwapHeader(&noffH);
    ASSERT(noffH.noffMagic == NOFFMAGIC);

// how big is address space?
    size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
            + UserStackSize;    // we need to increase the size
                  // to leave room for the stack
    numPages = divRoundUp(size, PageSize);
//  cout << "number of pages of " << fileName<< " is "<<numPages<<endl;
    size = numPages * PageSize;
```

2. Initializing pageTable and printing how many frames this thread requires.



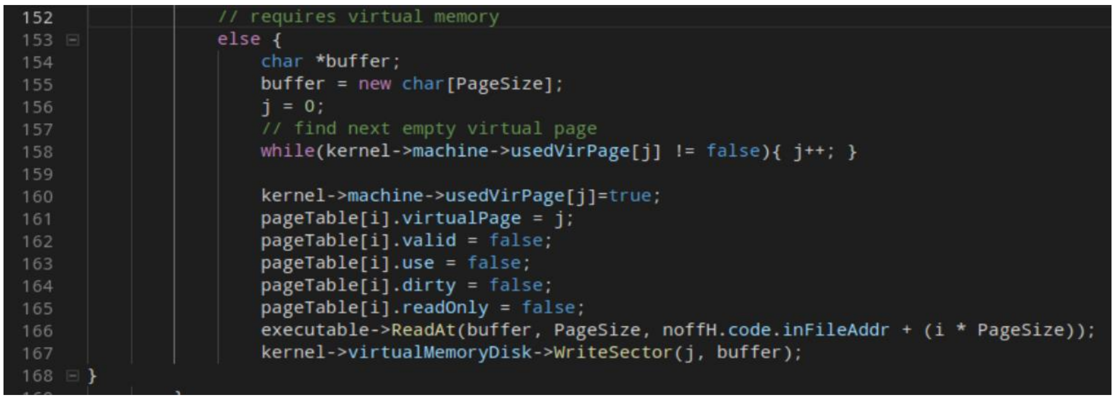
```

// Daniel added
// load page table
pageTable = new TranslationEntry[numPages];
for (unsigned int i = 0; i < numPages; i++) {
    pageTable[i].virtualPage = i;
    pageTable[i].physicalPage = i;
    pageTable[i].valid = true;
    pageTable[i].use = false;
    pageTable[i].dirty = false;
    pageTable[i].readOnly = false;
}
printf("Thread %d requires %d frame, %d free frame remain\n", AddrSpace::ID, numPages, kernel->machine->FreePages);
DEBUG(dbgAddr, "Initializing address space: " << numPages << ", " << size);
```

3. Allocating frames to thread. Looping through all pages this thread needs. If main memory has space, assigning frame to page. Make sure to record frame's corresponding page and push frame number into fifo. Then read file's content according to thread's allocated main memory size.



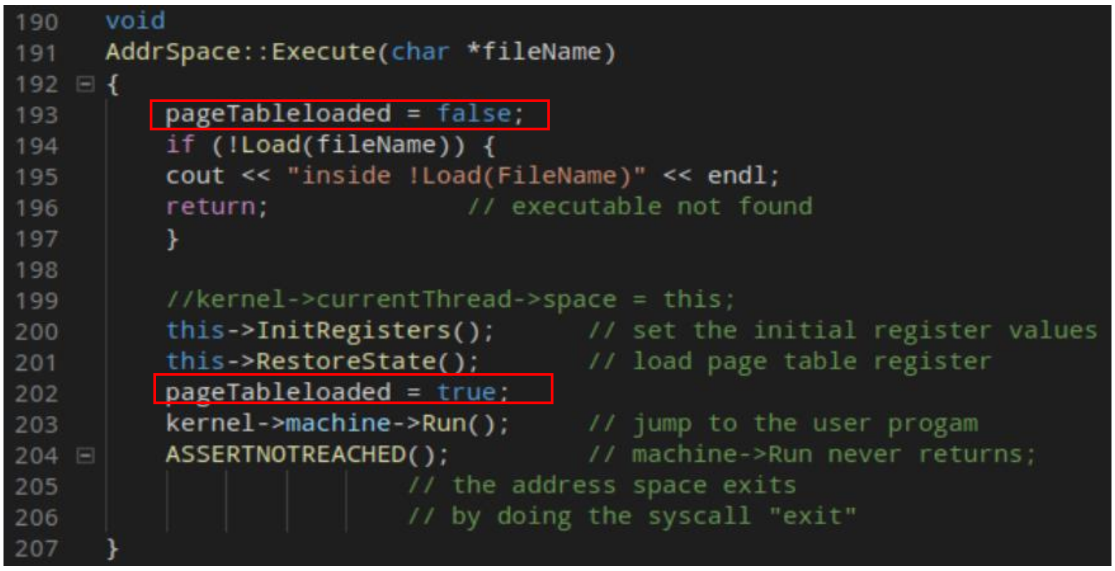
```

// 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);
        // Daniel added
        // allocate main memory
        for(unsigned int j = 0, i = 0; i < numPages; i++){
            j = 0;
            while(j < NumPhysPages && kernel->machine->usedPhyPage[j] == true)j++;
            if(j < NumPhysPages) {
                bzero(&kernel->machine->mainMemory[j * PageSize], PageSize);
                kernel->machine->usedPhyPage[j] = true;
                kernel->machine->FreePages--;
                kernel->machine->frameTable[j] = &pageTable[i];
                pageTable[i].physicalPage = j;
                pageTable[i].valid = true;
                pageTable[i].use = false;
                pageTable[i].dirty = false;
                pageTable[i].readOnly = false;
                pageTable[i].ID = ID;
                // Add physical address to FIFO queue
                kernel->machine->fifo.push(j);

                executable->ReadAt( &(kernel->machine->mainMemory[j * PageSize]), PageSize,
                    noffH.code.inFileAddr + (i*PageSize));
            }
```

4. If main memory's space is not enough, find space in SynchDisk (using first fit) and read file's content into SynchDisk. Make sure to set pageTable[i].valid = false, so that kernel knows this page is not in main memory.



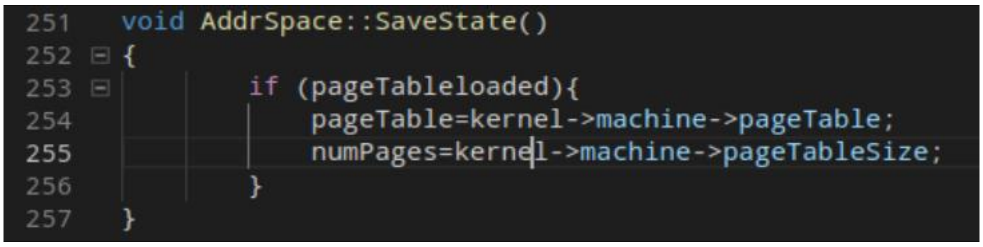
```
            // requires virtual memory
            else {
                char *buffer;
                buffer = new char[PageSize];
                j = 0;
                // find next empty virtual page
                while(kernel->machine->usedVirPage[j] != false){ j++; }

                kernel->machine->usedVirPage[j]=true;
                pageTable[i].virtualPage = j;
                pageTable[i].valid = false;
                pageTable[i].use = false;
                pageTable[i].dirty = false;
                pageTable[i].readOnly = false;
                executable->ReadAt(buffer, PageSize, noffH.code.inFileAddr + (i * PageSize));
                kernel->virtualMemoryDisk->WriteSector(j, buffer);
}
```

5. Set pageTableloaded = true after pageTable is successful loaded. (RestoreState will load the pagetable).



```
void
AddrSpace::Execute(char *fileName)
{
    pageTableloaded = false;
    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
    pageTableloaded = true;
    kernel->machine->Run();     // jump to the user progam
    ASSERTNOTREACHED();         // machine->Run never returns;
                        // the address space exits
                        // by doing the syscall "exit"
}
```

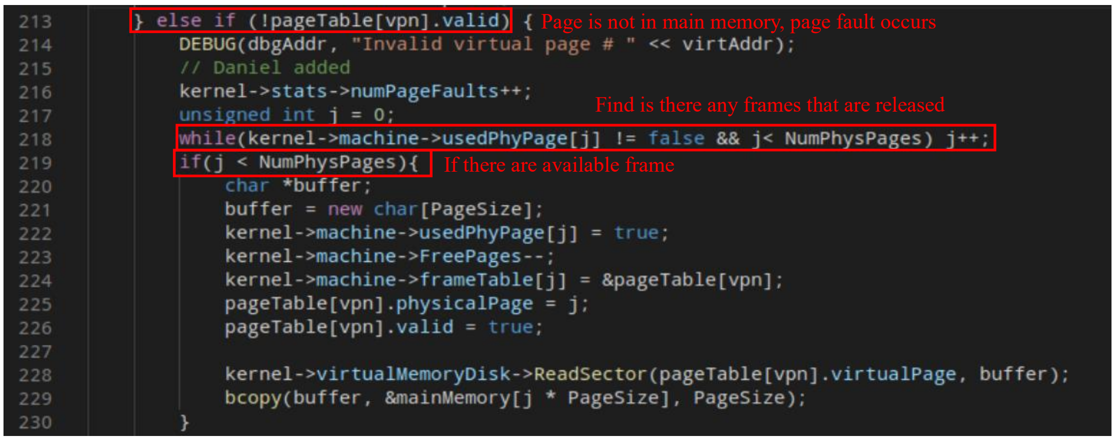
6. On a context switch, save pageTable status only if pageTable is successful loaded.



```
void AddrSpace::SaveState()
{
        if (pageTableloaded){
            pageTable=kernel->machine->pageTable;
            numPages=kernel->machine->pageTableSize;
        }
}
```

After finish modifying the process of loading thread's address space. We need to implement page replacement. I implement page replacement in translate.cc. There are two cases when a page fault occurred

Case1: Some frames are released, so we can just allocate that frame to thread.



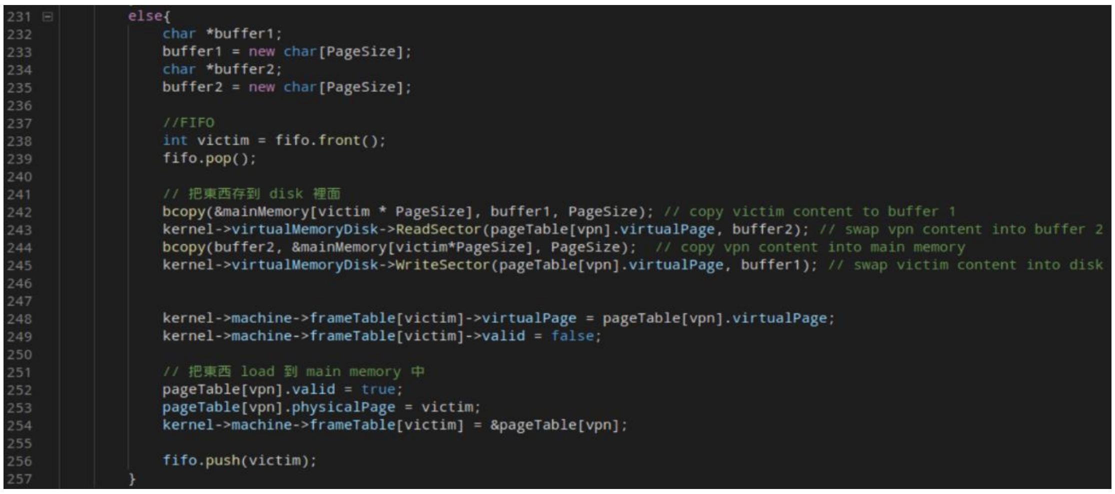
```
    } else if (!pageTable[vpn].valid) {
        DEBUG(dbgAddr, "Invalid virtual page # " << virtAddr);
        // Daniel added
        kernel->stats->numPageFaults++;
        unsigned int j = 0;
        while(kernel->machine->usedPhyPage[j] != false && j< NumPhysPages) j++;
        if(j < NumPhysPages){
            char *buffer;
            buffer = new char[PageSize];
            kernel->machine->usedPhyPage[j] = true;
            kernel->machine->FreePages--;
            kernel->machine->frameTable[j] = &pageTable[vpn];
            pageTable[vpn].physicalPage = j;
            pageTable[vpn].valid = true;

            kernel->virtualMemoryDisk->ReadSector(pageTable[vpn].virtualPage, buffer);
            bcopy(buffer, &mainMemory[j * PageSize], PageSize);
        }
```

Page is not in main memory, page fault occurs

Find is there any frames that are released

If there are available frame

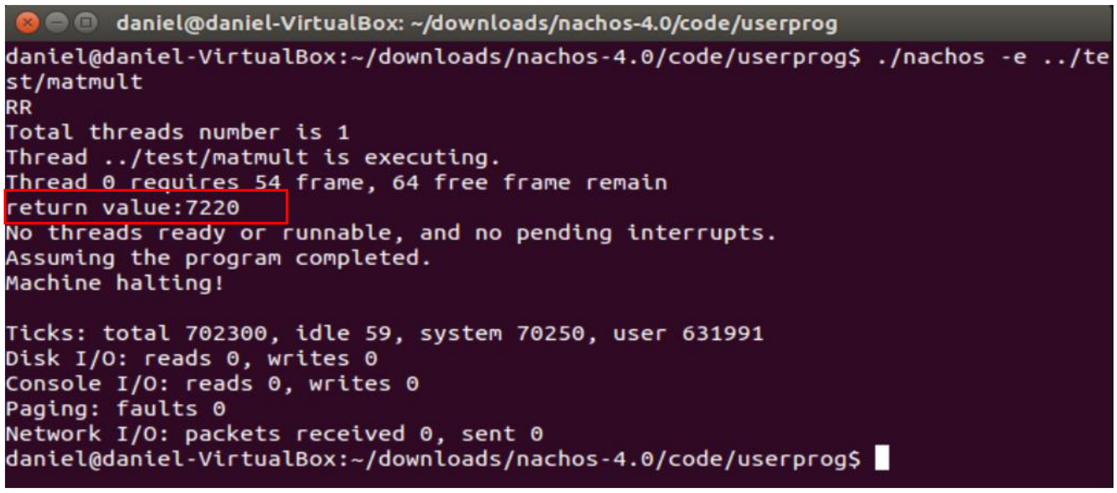
Case 2: There are no available frame, swap out a frame (using first in first out) and swap in page’s corresponding content from SynchDisk.



```
    else{
        char *buffer1;
        buffer1 = new char[PageSize];
        char *buffer2;
        buffer2 = new char[PageSize];

        //FIFO
        int victim = fifo.front();
        fifo.pop();

        // 把東西存到 disk 裡面
        bcopy(&mainMemory[victim * PageSize], buffer1, PageSize); // copy victim content to buffer 1
        kernel->virtualMemoryDisk->ReadSector(pageTable[vpn].virtualPage, buffer2); // swap vpn content into buffer 2
        bcopy(buffer2, &mainMemory[victim*PageSize], PageSize);  // copy vpn content into main memory
        kernel->virtualMemoryDisk->WriteSector(pageTable[vpn].virtualPage, buffer1); // swap victim content into disk


        kernel->machine->frameTable[victim]->virtualPage = pageTable[vpn].virtualPage;
        kernel->machine->frameTable[victim]->valid = false;

        // 把東西 load 到 main memory 中
        pageTable[vpn].valid = true;
        pageTable[vpn].physicalPage = victim;
        kernel->machine->frameTable[victim] = &pageTable[vpn];

        fifo.push(victim);
    }
```

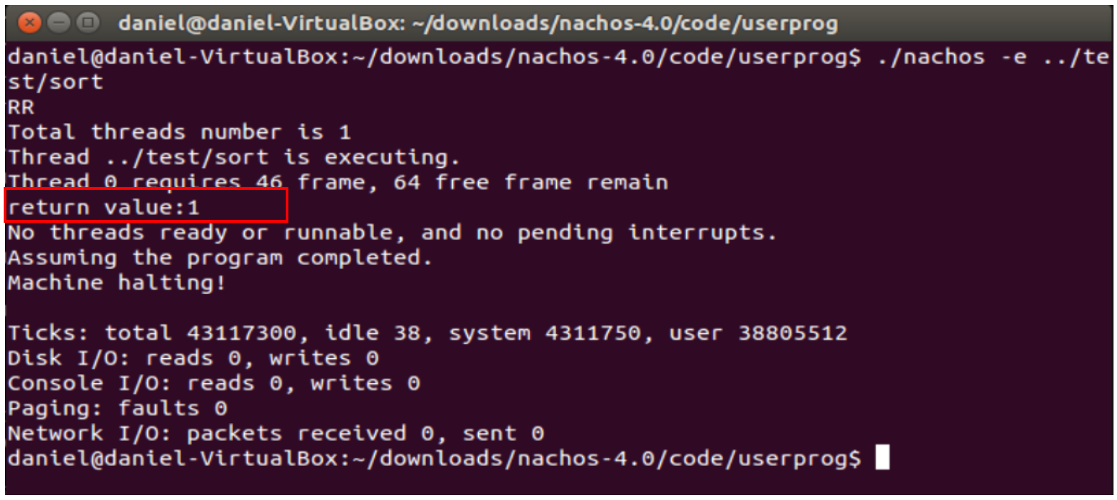
3. Result

matmult.c



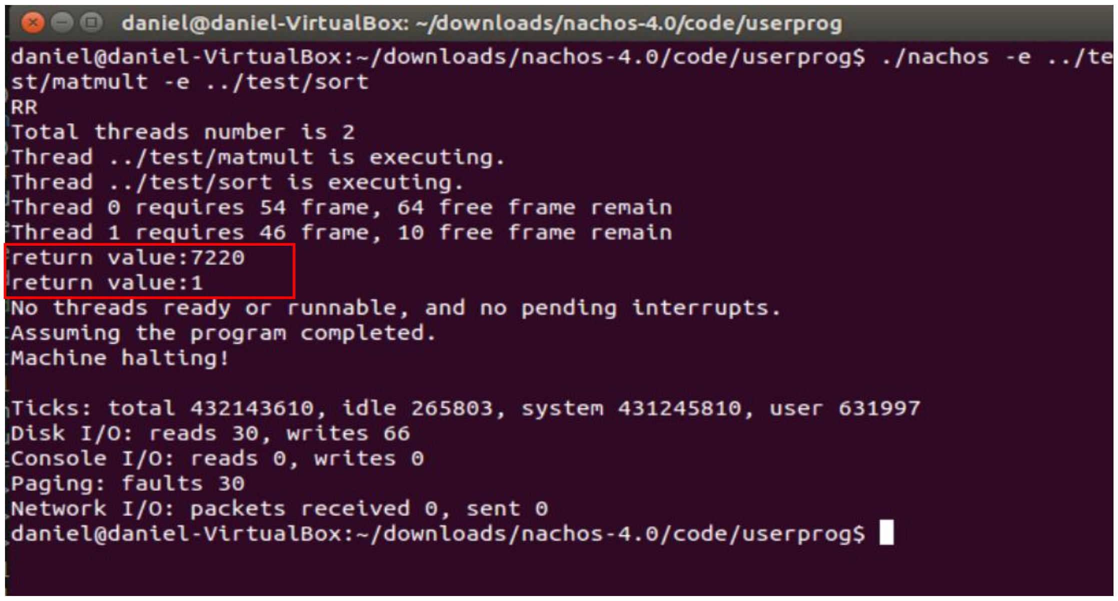
```
daniel@daniel-VirtualBox: ~/downloads/nachos-4.0/code/userprog
daniel@daniel-VirtualBox:~/downloads/nachos-4.0/code/userprog$ ./nachos -e ../te
st/matmult
RR
Total threads number is 1
Thread ../test/matmult is executing.
Thread 0 requires 54 frame, 64 free frame remain
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 702300, idle 59, system 70250, user 631991
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
daniel@daniel-VirtualBox:~/downloads/nachos-4.0/code/userprog$
```

sort.c



```
daniel@daniel-VirtualBox: ~/downloads/nachos-4.0/code/userprog
daniel@daniel-VirtualBox:~/downloads/nachos-4.0/code/userprog$ ./nachos -e ../te
st/sort
RR
Total threads number is 1
Thread ../test/sort is executing.
Thread 0 requires 46 frame, 64 free frame remain
return value:1
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 43117300, idle 38, system 4311750, user 38805512
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
daniel@daniel-VirtualBox:~/downloads/nachos-4.0/code/userprog$
```

matmult.c and sort.c simultaneously



daniel@daniel-VirtualBox: ~/downloads/nachos-4.0/code/userprog

```
daniel@daniel-VirtualBox:~/downloads/nachos-4.0/code/userprog$ ./nachos -e ../te
st/matmult -e ../test/sort
RR
Total threads number is 2
Thread ../test/matmult is executing.
Thread ../test/sort is executing.
Thread 0 requires 54 frame, 64 free frame remain
Thread 1 requires 46 frame, 10 free frame remain
return value:7220
return value:1
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 432143610, idle 265803, system 431245810, user 631997
Disk I/O: reads 30, writes 66
Console I/O: reads 0, writes 0
Paging: faults 30
Network I/O: packets received 0, sent 0
daniel@daniel-VirtualBox:~/downloads/nachos-4.0/code/userprog$
```

At first, I couldn’t successfully run mamult.c and sort.c at the same time. Kernel kept reporting illegal instruction exception. After debugging for three days, I found out that I forgot to load file into SynchDisk (virtual disk). Therefore, nothing was in the disk when swapping occurred.

There is another page replacement method called Least Recently Used which swap out least recently used page. All I need to do is to add a variable in pageTable to record number of each page is used. However, I didn’t implement LRU, since FIFO is quick enough.