# NachOS_Project3

生醫電子暨資訊所 碩一 R07945029 王思敏

Problem analysis:

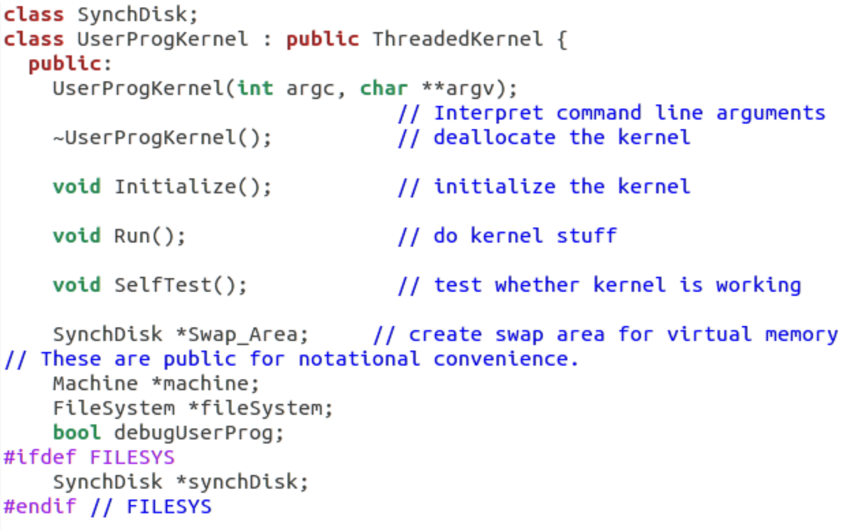
With the existing physical memory size of NachOS, some test cases required lots of memory will lead to core dumped. Thus, a virtual memory will be needed for storing pages since space in main memory may be insufficient.

Plan and implement:

The plan is to realize a virtual memory that if the main memory has no space, pages and segments of data can still be stored in other storage devices by page replacement algorithm. When the space in main memory is enough, pages will be stored in main memory. If the space is insufficient, the pages and data will swap out. When there is valid space, i.e. the space is released, and the data is needed, the pages and data will then swap in main memory. The scheduling method of the page replacement algorithm is implemented by LRU(Least Recently Used).

Code and comment:

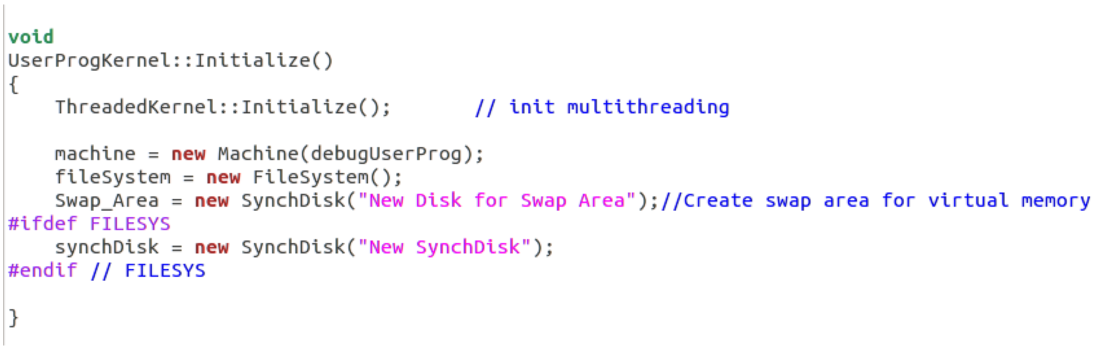
userkernel.h



```
class SynchDisk;
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 *Swap_Area;     // create swap area for virtual memory
// These are public for notational convenience.
    Machine *machine;
    FileSystem *fileSystem;
    bool debugUserProg;
#ifdef FILESYS
    SynchDisk *synchDisk;
#endif // FILESYS
```

Add SynchDisk *Swap_Area to create a disk area for storing the pages that can’t enter the main memory.

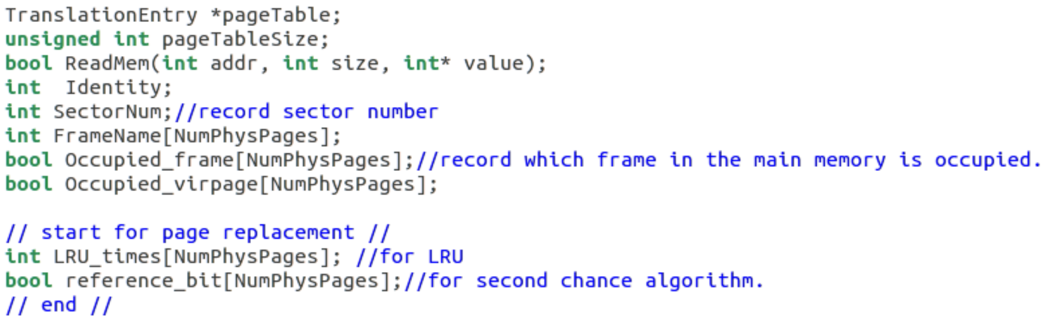
userkernel.cc



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

    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    Swap_Area = new SynchDisk("New Disk for Swap Area");//Create swap area for virtual memory
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS

}
```

Initialize and allocate memory space to Swap_Area.

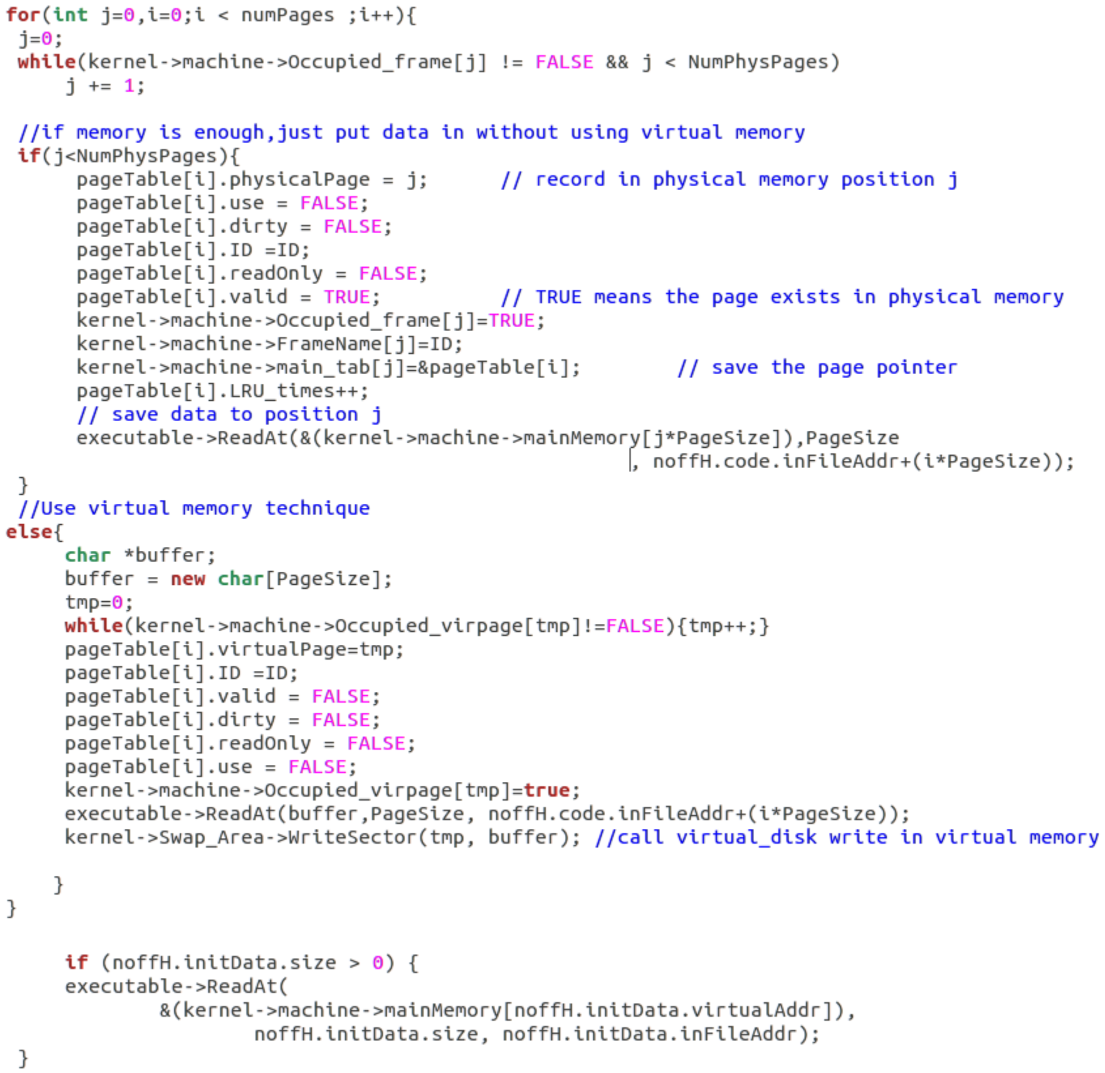
machine.h



```
TranslationEntry *pageTable;
unsigned int pageTableSize;
bool ReadMem(int addr, int size, int* value);
int  Identity;
int SectorNum;//record sector number
int FrameName[NumPhysPages];
bool Occupied_frame[NumPhysPages];//record which frame in the main memory is occupied.
bool Occupied_virpage[NumPhysPages];

// start for page replacement //
int LRU_times[NumPhysPages]; //for LRU
bool reference_bit[NumPhysPages];//for second chance algorithm.
// end //
```

Add some members in class machine for recording sector number, frame paged, and the frame being occupied in main memory and virtual memory separately.

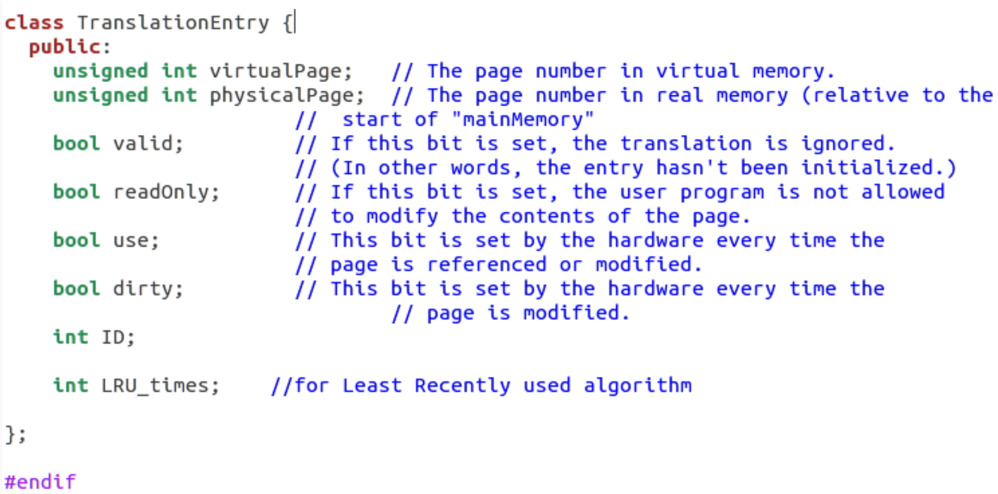
addrspace.cc



```
for(int j=0,i=0;i < numPages ;i++){
 j=0;
 while(kernel->machine->Occupied_frame[j] != FALSE && j < NumPhysPages)
     j += 1;

 //if memory is enough,just put data in without using virtual memory
 if(j<NumPhysPages){
      pageTable[i].physicalPage = j;      // record in physical memory position j
      pageTable[i].use = FALSE;
      pageTable[i].dirty = FALSE;
      pageTable[i].ID =ID;
      pageTable[i].readOnly = FALSE;
      pageTable[i].valid = TRUE;          // TRUE means the page exists in physical memory
      kernel->machine->Occupied_frame[j]=TRUE;
      kernel->machine->FrameName[j]=ID;
      kernel->machine->main_tab[j]=&pageTable[i];        // save the page pointer
      pageTable[i].LRU_times++;
      // save data to position j
      executable->ReadAt(&(kernel->machine->mainMemory[j*PageSize]),PageSize
                                                     , noffH.code.inFileAddr+(i*PageSize));
 }
 //Use virtual memory technique
else{
     char *buffer;
     buffer = new char[PageSize];
     tmp=0;
     while(kernel->machine->Occupied_virpage[tmp]!=FALSE){tmp++;}
     pageTable[i].virtualPage=tmp;
     pageTable[i].ID =ID;
     pageTable[i].valid = FALSE;
     pageTable[i].dirty = FALSE;
     pageTable[i].readOnly = FALSE;
     pageTable[i].use = FALSE;
     kernel->machine->Occupied_virpage[tmp]=true;
     executable->ReadAt(buffer,PageSize, noffH.code.inFileAddr+(i*PageSize));
     kernel->Swap_Area->WriteSector(tmp, buffer); //call virtual_disk write in virtual memory

    }
}

     if (noffH.initData.size > 0) {
     executable->ReadAt(
             &(kernel->machine->mainMemory[noffH.initData.virtualAddr]),
                     noffH.initData.size, noffH.initData.inFileAddr);
 }
```

The modification was almost done within Load function. First, the executable code will be allocated to main memory, and the corresponding page table will be recorded. If the frame number in main memory is enough, the pages will be stored into main memory. When the frame number is insufficient, the lasting pages will be written into virtual memory by WriteSector. The corresponding page table will also be recorded and be set to invalid.

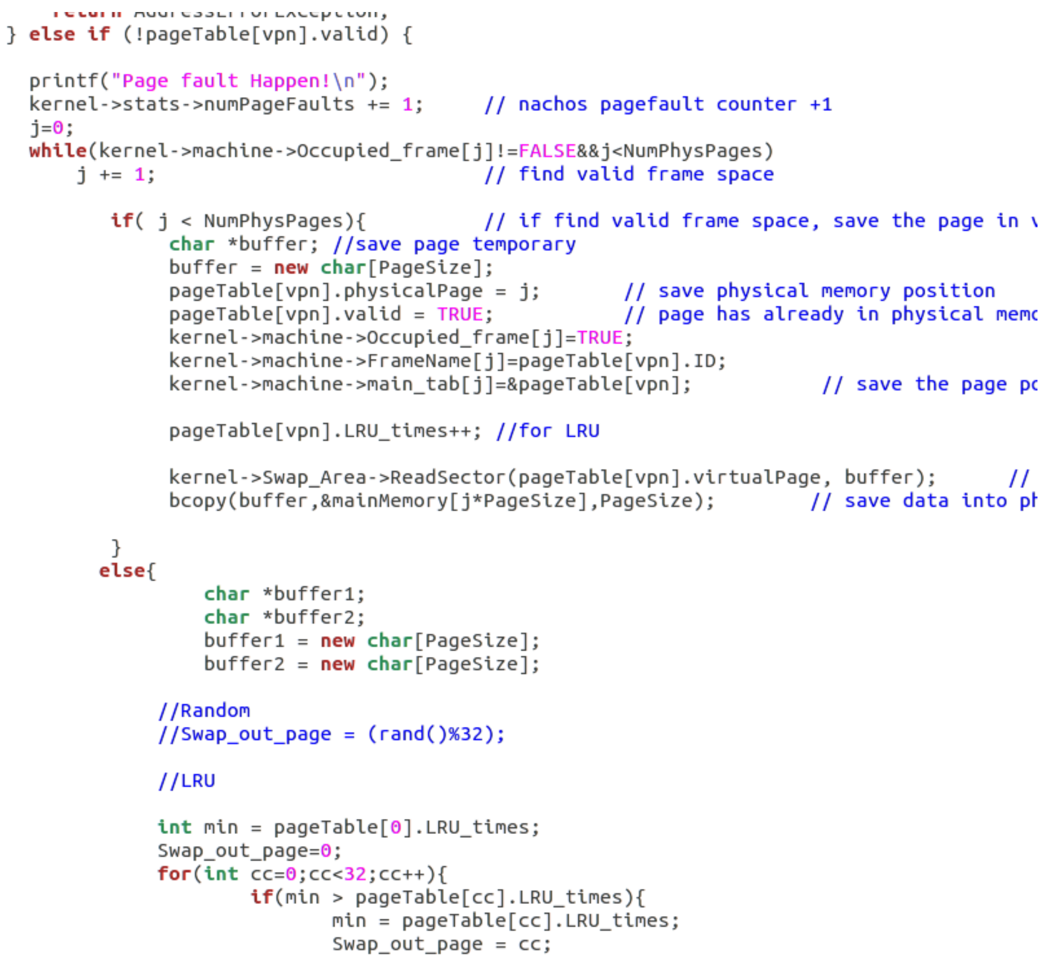
translate.h



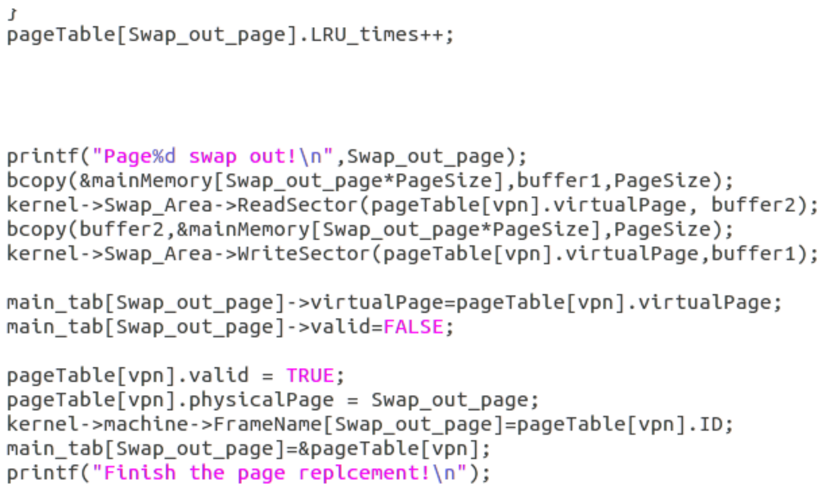
```
class TranslationEntry {
  public:
    unsigned int virtualPage;   // The page number in virtual memory.
    unsigned int physicalPage;  // The page number in real memory (relative to the
                        //  start of "mainMemory"
    bool valid;             // If this bit is set, the translation 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.
    int ID;

    int LRU_times;    //for Least Recently used algorithm

};

#endif
```

Add ID and LRU_times in class TranslationEntry.

translate.cc



```
    return AddressErrorException;
} else if (!pageTable[vpn].valid) {

  printf("Page fault Happen!\n");
  kernel->stats->numPageFaults += 1;    // nachos pagefault counter +1
  j=0;
  while(kernel->machine->Occupied_frame[j]!=FALSE&&j<NumPhysPages)
      j += 1;                           // find valid frame space

         if( j < NumPhysPages){         // if find valid frame space, save the page in v
              char *buffer; //save page temporary
              buffer = new char[PageSize];
              pageTable[vpn].physicalPage = j;      // save physical memory position
              pageTable[vpn].valid = TRUE;          // page has already in physical memo
              kernel->machine->Occupied_frame[j]=TRUE;
              kernel->machine->FrameName[j]=pageTable[vpn].ID;
              kernel->machine->main_tab[j]=&pageTable[vpn];          // save the page po

              pageTable[vpn].LRU_times++; //for LRU

              kernel->Swap_Area->ReadSector(pageTable[vpn].virtualPage, buffer);      //
              bcopy(buffer,&mainMemory[j*PageSize],PageSize);       // save data into ph

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

             //Random
             //Swap_out_page = (rand()%32);

             //LRU

             int min = pageTable[0].LRU_times;
             Swap_out_page=0;
             for(int cc=0;cc<32;cc++){
                     if(min > pageTable[cc].LRU_times){
                            min = pageTable[cc].LRU_times;
                            Swap_out_page = cc;
```

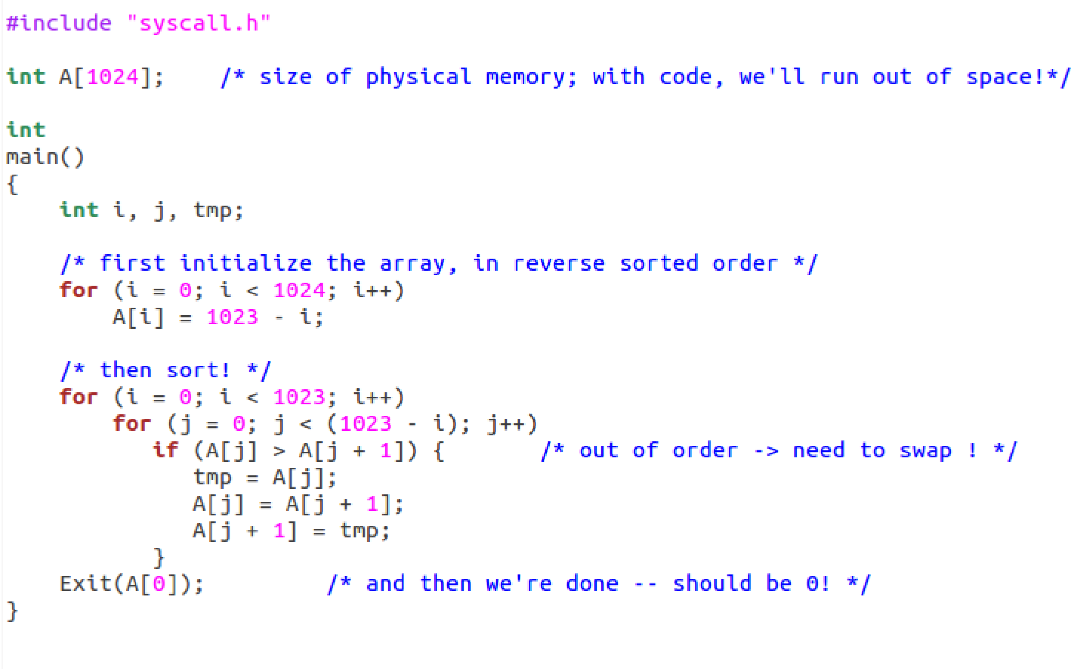


```
}
pageTable[Swap_out_page].LRU_times++;



printf("Page%d swap out!\n",Swap_out_page);
bcopy(&mainMemory[Swap_out_page*PageSize],buffer1,PageSize);
kernel->Swap_Area->ReadSector(pageTable[vpn].virtualPage, buffer2);
bcopy(buffer2,&mainMemory[Swap_out_page*PageSize],PageSize);
kernel->Swap_Area->WriteSector(pageTable[vpn].virtualPage,buffer1);

main_tab[Swap_out_page]->virtualPage=pageTable[vpn].virtualPage;
main_tab[Swap_out_page]->valid=FALSE;

pageTable[vpn].valid = TRUE;
pageTable[vpn].physicalPage = Swap_out_page;
kernel->machine->FrameName[Swap_out_page]=pageTable[vpn].ID;
main_tab[Swap_out_page]=&pageTable[vpn];
printf("Finish the page replcement!\n");
```

It’s the main part of LRU implement. First, the valid bit of page table will be identified to ensure the place of the page, i.e. in main memory or in virtual memory. If the page is in virtual memory, it will check if there is free space in main memory now. The page replacement will happen in the situation that there is no space in main memory. Two buffers will be declared for swap in/out. Then, to imply LRU, a for loop will be used to find the least used page, and the steps such as reading from virtual memory, copying to main memory, and selecting the swap out page to virtual memory are implied by bcopy, ReadSector, and WriteSector.

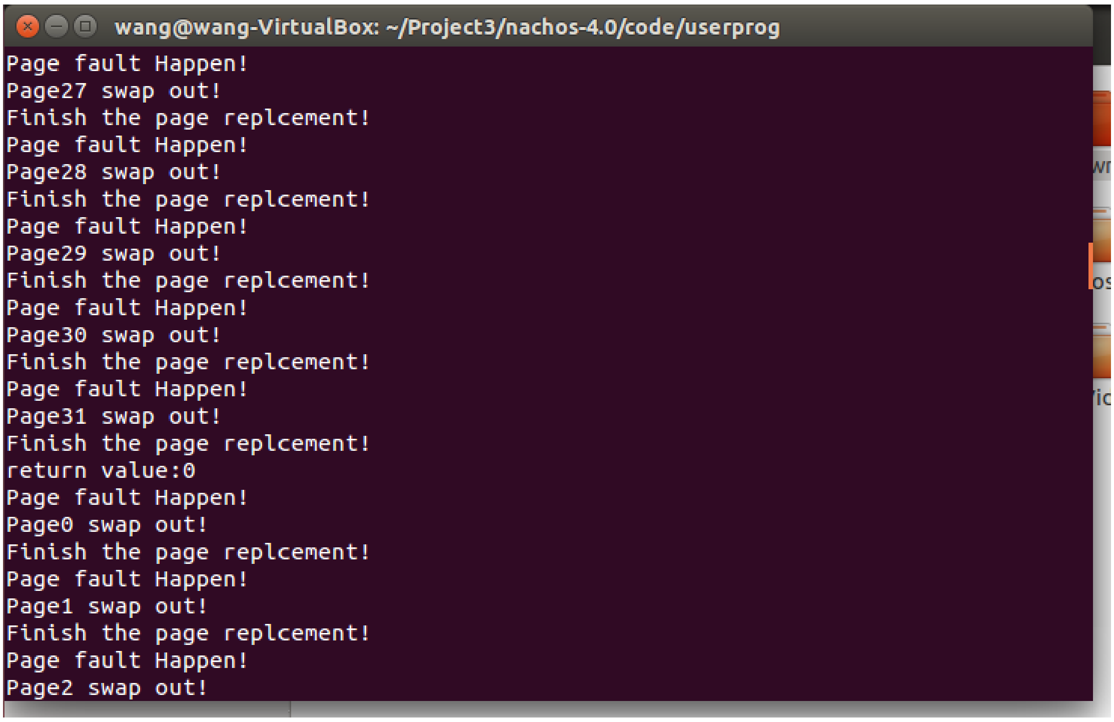
Experiment result:

Because of find the wrong part within in sort test code, I modify the sort test code to the correct sort algorithm.

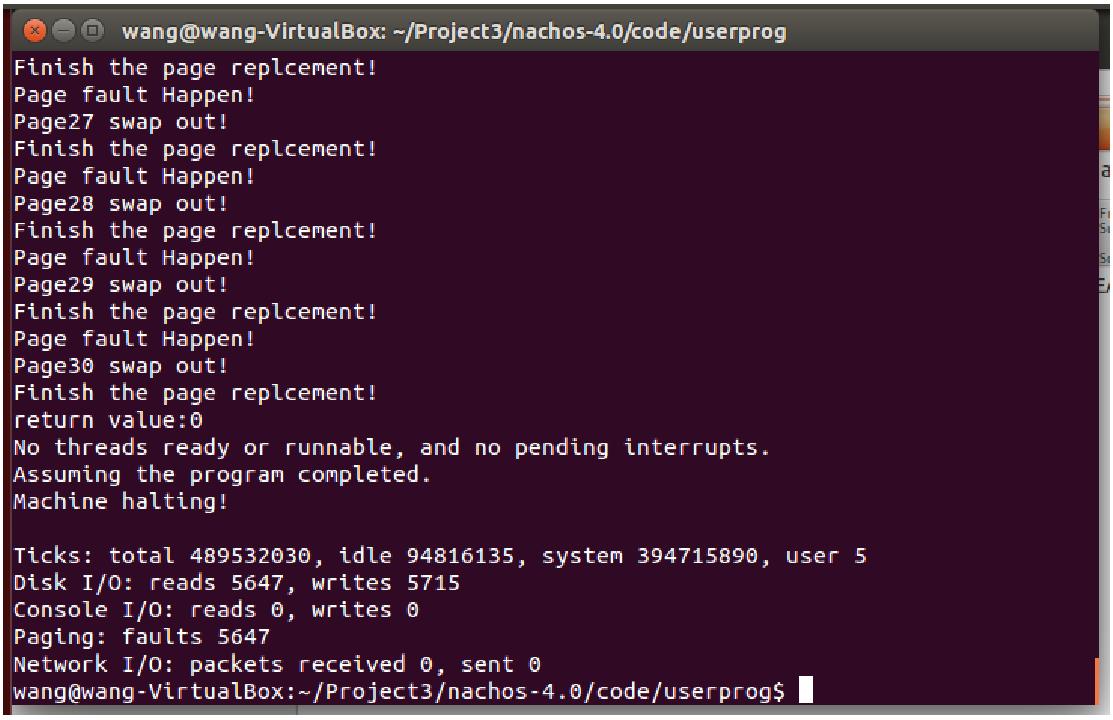


```
#include "syscall.h"

int A[1024];    /* size of physical memory; with code, we'll run out of space!*/

int
main()
{
    int i, j, tmp;

    /* first initialize the array, in reverse sorted order */
    for (i = 0; i < 1024; i++)
        A[i] = 1023 - i;

    /* then sort! */
    for (i = 0; i < 1023; i++)
        for (j = 0; j < (1023 - i); j++)
           if (A[j] > A[j + 1]) {       /* out of order -> need to swap ! */
              tmp = A[j];
              A[j] = A[j + 1];
              A[j + 1] = tmp;
           }
    Exit(A[0]);         /* and then we're done -- should be 0! */
}
```

Execute    ./nachos –e ../test/sort –e ../test/marmult



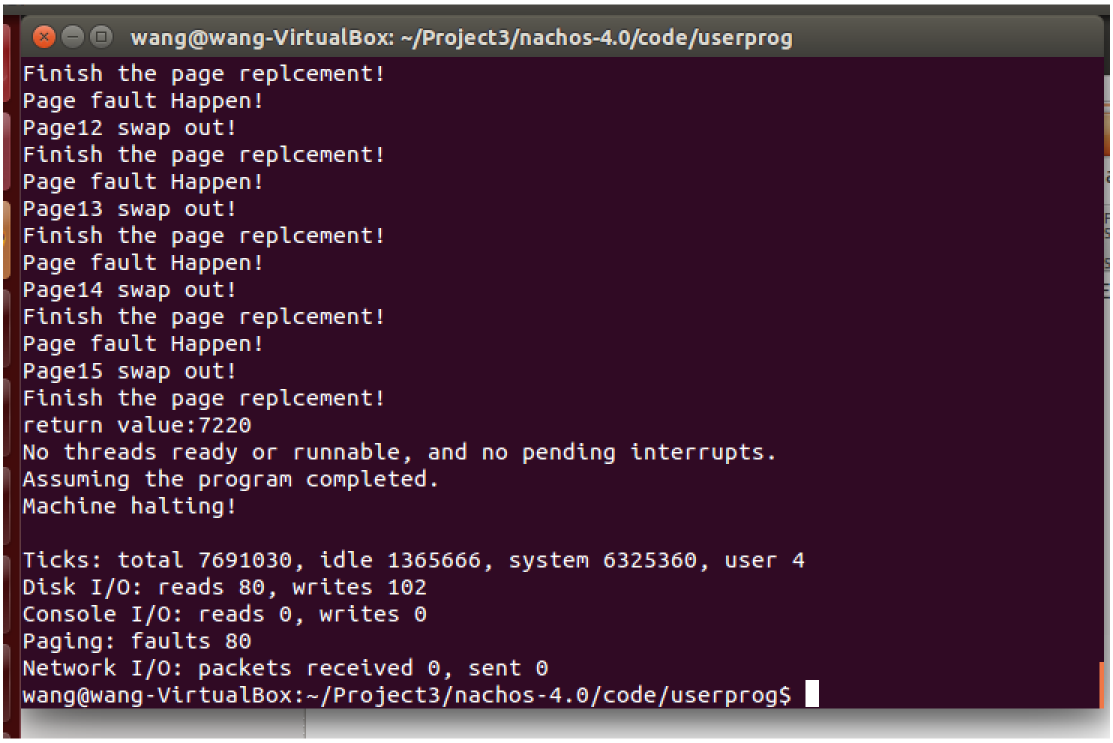
```
Page fault Happen!
Page27 swap out!
Finish the page replcement!
Page fault Happen!
Page28 swap out!
Finish the page replcement!
Page fault Happen!
Page29 swap out!
Finish the page replcement!
Page fault Happen!
Page30 swap out!
Finish the page replcement!
Page fault Happen!
Page31 swap out!
Finish the page replcement!
return value:0
Page fault Happen!
Page0 swap out!
Finish the page replcement!
Page fault Happen!
Page1 swap out!
Finish the page replcement!
Page fault Happen!
Page2 swap out!
```



```
Finish the page replcement!
Page fault Happen!
Page27 swap out!
Finish the page replcement!
Page fault Happen!
Page28 swap out!
Finish the page replcement!
Page fault Happen!
Page29 swap out!
Finish the page replcement!
Page fault Happen!
Page30 swap out!
Finish the page replcement!
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 489532030, idle 94816135, system 394715890, user 5
Disk I/O: reads 5647, writes 5715
Console I/O: reads 0, writes 0
Paging: faults 5647
Network I/O: packets received 0, sent 0
wang@wang-VirtualBox:~/Project3/nachos-4.0/code/userprog$
```

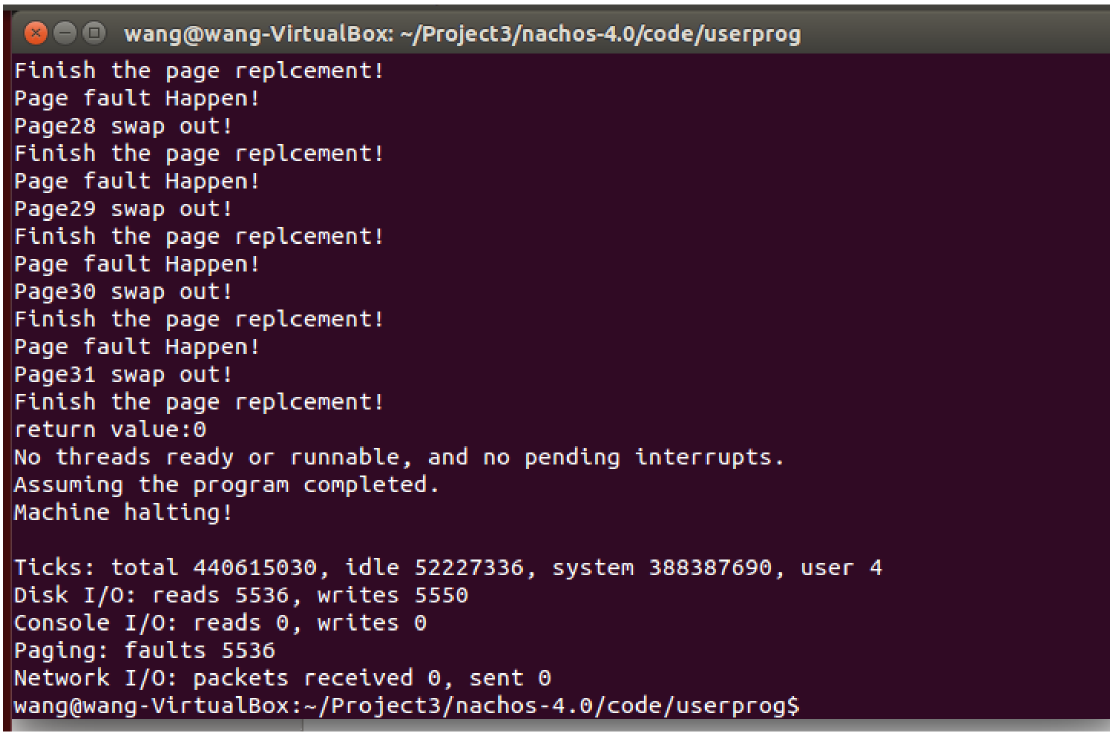
The sort part returns 0 and the matmult part returns 7220. Thus, the result is correct.

Execute      ./nachos –e ../test/marmult



```
Finish the page replcement!
Page fault Happen!
Page12 swap out!
Finish the page replcement!
Page fault Happen!
Page13 swap out!
Finish the page replcement!
Page fault Happen!
Page14 swap out!
Finish the page replcement!
Page fault Happen!
Page15 swap out!
Finish the page replcement!
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 7691030, idle 1365666, system 6325360, user 4
Disk I/O: reads 80, writes 102
Console I/O: reads 0, writes 0
Paging: faults 80
Network I/O: packets received 0, sent 0
wang@wang-VirtualBox:~/Project3/nachos-4.0/code/userprog$
```

Execute      ./nachos –e ../test/sort



```
Finish the page replcement!
Page fault Happen!
Page28 swap out!
Finish the page replcement!
Page fault Happen!
Page29 swap out!
Finish the page replcement!
Page fault Happen!
Page30 swap out!
Finish the page replcement!
Page fault Happen!
Page31 swap out!
Finish the page replcement!
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 440615030, idle 52227336, system 388387690, user 4
Disk I/O: reads 5536, writes 5550
Console I/O: reads 0, writes 0
Paging: faults 5536
Network I/O: packets received 0, sent 0
wang@wang-VirtualBox:~/Project3/nachos-4.0/code/userprog$
```

We can get the correct results when executing separately also.