## Design Solution and Implementation for Each Requirement:

### Task 1 Implement system calls and exception:

#### SC\_ConsoleRead

In Syscall.h, consoleread is defined as “int ConsoleRead(char \*buffer, int size);” In exception handler, we first get the size and address by reading register, and then them as parameter to call SysRead() function. And finally, update the PC.

case SC\_ConsoleRead:

                {

                    DEBUG(dbgSys, "ConsoleRead " << kernel->machine->ReadRegister(4) << "\n");

                    /\* Process SysAdd Systemcall\*/

                    // 5 is the size of buffer

                    int size = kernel->machine->ReadRegister(5);

                    // 4 get the first address of buffer

                    int addr = kernel->machine->ReadRegister(4);

                    //define in ksyscall.h

                    int res = SysRead(addr, size);

                    // void WriteRegister(int num, int value);

                    kernel->machine->WriteRegister(2, res);

                    cout << "SC\_ConsoleRead Done" << endl;

                    /\* Modify return point \*/

                    {

                    /\* set previous programm counter (debugging only)\*/

                    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

                    /\* set programm counter to next instruction (all Instructions are 4 byte wide)\*/

                    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

                    /\* set next programm counter for brach execution \*/

                    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);

                    }

                    // SysHalt();

                    return;

                    ASSERTNOTREACHED();

                }break;

#### SC\_ConsoleWrite

In Syscall.h, consolewriteis defined as “int ConsoleWrite(char \*buffer, int size);” In exception handler, we first get the size and address by reading register, and then them as parameter to call SysWrite() function. And finally, update the PC.

case SC\_ConsoleWrite:

                {

                    DEBUG(dbgSys, "ConsoleWrite " << kernel->machine->ReadRegister(4) << "\n");

                    /\* Process SysAdd Systemcall\*/

                    // 5 is the size of buffer

                    int size = kernel->machine->ReadRegister(5);

                    // 4 get the first address of buffer

                    int addr = kernel->machine->ReadRegister(4);

                    //define in ksyscall.h

                    int res = SysWrite(addr, size);

                    // void WriteRegister(int num, int value);

                    kernel->machine->WriteRegister(2, res);

                    //cout << "SC\_ConsoleWrite Done" << endl;

                    /\* Modify return point \*/

                    {

                    /\* set previous programm counter (debugging only)\*/

                    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

                    /\* set programm counter to next instruction (all Instructions are 4 byte wide)\*/

                    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

                    /\* set next programm counter for brach execution \*/

                    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);

                    }

                    // SysHalt();

                    return;

                    ASSERTNOTREACHED();

                }break;

#### SC\_Fork

In Syscall.h, consolewriteis defined as “SpaceId Exec(char\* exec\_name);” In exception handler, we create a new thread, and initialize the thread. After setting the register, fork this thread. And finally, update the PC

case SC\_Fork:

                {

                    DEBUG(dbgSys, "Fork address" << kernel->machine->ReadRegister(4) << "\n");

                    int funAddr = kernel->machine->ReadRegister(4);

                    // for setting

                    Thread \*t = new Thread("Syscall Fork");

                    t->space = new AddrSpace();

                    t->space = kernel->currentThread->space;

                    t->SaveUserState();

                    // cout << "PCReg: " << PCReg <<"; funAddr: " << funAddr << endl;

                    t->setUserRegister(PCReg, funAddr);

                    t->setUserRegister(NextPCReg, funAddr + 4);

                    t->Fork((VoidFunctionPtr) forkFunc, (void\*) 0);

                    /\* Modify return point \*/

                    {

                    /\* set previous programm counter (debugging only)\*/

                    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

                    /\* set programm counter to next instruction (all Instructions are 4 byte wide)\*/

                    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

                    /\* set next programm counter for brach execution \*/

                    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);

                    }

                    // SysHalt();

                    return;

                    ASSERTNOTREACHED();

                }break;

#### SC\_Exec

In Syscall.h, consolewriteis defined as “void Exit(int status);” In exception handler, we read from memory to get the executable (e.g. ../test/consolewrite), and then we create a newThread to load the executable, then fork this thread. And finally, update the PC.

case SC\_Exec:

                {

                    // cout<< "Executing SC\_Exec" << endl;

                    //char\* prog = (char\*)kernel->machine->ReadRegister(4);

                    int vaddr, memval, i =0;

                    char filename[100];

                    vaddr = kernel->machine->ReadRegister(4);

                    kernel->machine->ReadMem(vaddr, 1, &memval);

                    while((\*(char\*)&memval) != '\0'){

                        filename[i] = (char) memval;

                        ++i;

                        vaddr++;

                        kernel->machine->ReadMem(vaddr, 1, &memval);

                    }

                    filename[i]  = (char)memval;

                    cout << "Executing " <<  filename << endl;

                    Thread \*t = new Thread("New Exec Thread");

                    t->space = new AddrSpace();

                    t->space->Load(filename);

                    t->Fork((VoidFunctionPtr) execFunc, t);

                    /\* Modify return point \*/

                    {

                    /\* set previous programm counter (debugging only)\*/

                    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

                    /\* set programm counter to next instruction (all Instructions are 4 byte wide)\*/

                    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

                    /\* set next programm counter for brach execution \*/

                    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);

                    }

                    // SysHalt();

                    return;

                    ASSERTNOTREACHED();

                }break;

#### SC\_Exit

In Syscall.h, consolewriteis defined as “int Fork(void);” In exception handler, There are two parts needed to be done. One is update the TLB, the other is finish the thread.

case SC\_Exit:

                {

                    cout << "Executing SC\_Exit." << endl;

                    //1. first part: free TLB

                    TranslationEntry\* pageEntry = kernel->currentThread->space->getPageTable();

                    int num = kernel->currentThread->space->getNumPages();

                    // cout << "Gonna clear" << num << " pages" << endl;

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

                        int n = pageEntry[i].physicalPage;

                        if(pageEntry[i].valid == true && pageEntry[i].physicalPage != -1){

                            cout << "Deallocate physical page number:" << n << endl;

                            kernel->freeMap->Clear(pageEntry[i].physicalPage);

                            pageEntry[i].physicalPage = -1;

                            pageEntry[i].valid = false;

                        }

                    }

                    //2. second part: finsh thread();

                    kernel->currentThread->Finish();

                    /\* Modify return point \*/

                    {

                    /\* set previous programm counter (debugging only)\*/

                    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

                    /\* set programm counter to next instruction (all Instructions are 4 byte wide)\*/

                    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

                    /\* set next programm counter for brach execution \*/

                    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);

                    }

                    // SysHalt();

                    return;

                    ASSERTNOTREACHED();

                }break;

#### Other

For the configuration, start.S and Makefile needed to be modified.

### Task 2 Implement multiprogramming:

#### Modify the Main.cc

1. Add a variable to store the quantum and container to store multi-program’s executable path.
2. Add flag to take quantum, such as “-q”

if (strcmp(argv[i], "-q") == 0) {

        quantum  = atoi(argv[i+1]);

    }

1. Check if the container is not empty, use a new thread to fork the executables.

for(int i = 0; i < programPaths.size(); i++ ){

        //cout <<"Forking " <<  programPaths[i] << endl;

        Thread\* newThread = new Thread(programPaths[i]);

        newThread->Fork((VoidFunctionPtr)RunUserProg, (char\*)programPaths[i]);

    }

#### Modify kernel.h kernel.cc

Modify the function ”void Initialize();” into ” void Initialize(int q);”. Pass the value of q to the variable quantum.

Add a variable named quantum.

#### Modify Timer.cc

Modify the setInterrupt() function as following:

void

Timer::SetInterrupt()

{

    if (!disable) {

       //int delay = TimerTicks;

        int delay = kernel->quantum;

       if (randomize) {

         delay = 1 + (RandomNumber() % (kernel->quantum \* 2));

        }

       // schedule the next timer device interrupt

       kernel->interrupt->Schedule(this, delay, TimerInt);

    }

}

### Task 3 Implement Memory Manager (Virtual Memory) with software TLB:

#### Modify kernel.h, kernel.cc

Add the variables as following, and initialize them in “void Kernel::Initialize(int q)” function.

    // For assig 2

    int quantum = 100;

    OpenFile\* swapSpace; // virtual memory

    int swapSpace\_counter; // virtual memory index

    //vector<TranslationEntry\*> entryList;

    List<TranslationEntry\*>\* entryList; // entry list in physical memory

    Bitmap\* freeMap; // corresponding to the index in physical memory

#### Modify addrspace.cc

To load a program, we will modify the Load() function. According to the pages number, we initialize the TranslationEntry, for each entry, we read buffer from memory, and write to the virtual memory.

// for assig 4

    /////////////////////////////============================================

    pageTable = new TranslationEntry[numPages];

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

        pageTable[i].virtualPage = kernel->swapSpace\_counter;

        pageTable[i].physicalPage = -1;

        pageTable[i].valid = FALSE;

        pageTable[i].use = FALSE;

        pageTable[i].dirty = FALSE;

        pageTable[i].readOnly = FALSE;

        char\* buffer = new char[PageSize];

        executable->ReadAt(buffer, PageSize, noffH.code.inFileAddr + i \* PageSize);

        //write the buffer into swapspace

        kernel->swapSpace->WriteAt(buffer, PageSize, kernel->swapSpace\_counter \* PageSize );

        kernel->swapSpace\_counter++;

    }

#### Modify exception.cc

In this file, our modification is mainly on case PageFaultException. To complete this part, firstly we attain the page fault address from virtual memory. Then, we check whether the main memory has capacity to put the entry.

If there is space for new entry in physical memory, we just update the entry information and then write the main memory.

If there is no space for new entry, we need to swap the entry between the physical and virtual memory. That is, we use strategy to pick an entry as victim in the physical memory, store information to virtual memory and kick out from physical memory. Then, the entry in virtual memory will conquer the victim’s physical address.

So, as the requirement states, FIFO is forbidden. Then, I chose to randomly pick the victim from entryList.

Specific code as following:

case PageFaultException:

        {

            RandomInit(100);

            bool randTag;

            int rand\_res = RandomNumber() % 2;

            // Fetch virtual address

            int pageFaultAddr = kernel->machine->ReadRegister(BadVAddrReg);

            // Fetch virtual page number from thread pageTable

            int pageFaultNum = pageFaultAddr / PageSize;

            //check the free physical page number from main memory

            int PPN = kernel->freeMap->FindAndSet();

            //Fetch page entry of current thread [by VPN(pageFaultNum)]

            TranslationEntry\* pageEntry = &kernel->currentThread->space->getPageTable()[pageFaultNum];

            TranslationEntry\* pageTable = kernel->currentThread->space->getPageTable();

            int spaceSize = kernel->currentThread->space->getNumPages();

            int phyPageMem;

            //cout << "Checkpoint PPN:" << PPN << endl;

            if(PPN != -1)

            // if there is space for insert

            {

                cout << "Allocating PPN:" << PPN << endl;

                // cout << "        VPN:" << pageEntry->virtualPage << endl;

                pageEntry->physicalPage = PPN;

                pageEntry->valid = true;

                //read data from swap space into main memory, using ReadAt();

                phyPageMem = PPN \* PageSize;

                kernel->swapSpace->ReadAt(&(kernel->machine->mainMemory[phyPageMem]), PageSize, pageEntry->virtualPage \* PageSize);

                //================> update container, such as entryList->insert(pageEntry);

                if (rand\_res == 0 || kernel->entryList->IsInList(pageEntry) )

                {

                    kernel->entryList->Append(pageEntry);

                }

                else

                {

                    kernel->entryList->Prepend(pageEntry);

                }

            }

            else

            // if there is no free memory

            {

                // Get the victim page from entryList

                TranslationEntry\* victim = kernel->entryList->RemoveFront();

                //cout << "Select the victim page, PPN: " << victim->physicalPage << "; VPN: " << victim->virtualPage << endl;

                if(victim->physicalPage != -1 && victim->valid == true){

                    int virPageIndex = victim->virtualPage \* PageSize;

                    int phyPageIndex = victim->physicalPage \* PageSize;

                    // copy the victim from main memory into the swapspace

                    kernel->swapSpace->WriteAt(&(kernel->machine->mainMemory[phyPageIndex]), PageSize, victim->virtualPage \* PageSize);

                    cout << "Reallocating PPN:" << victim->physicalPage << endl;

                    // update the victime page

                    victim->physicalPage = -1;

                    victim->valid = false;

                    // update the pageEntry

                    pageEntry->physicalPage = phyPageIndex / PageSize;

                    pageEntry->valid = true;

                    // write the pageEntry from swapspace into main memory

                    kernel->swapSpace->ReadAt(&(kernel->machine->mainMemory[phyPageIndex]), PageSize, pageEntry->virtualPage \* PageSize);

                    if (rand\_res == 0 || kernel->entryList->IsInList(pageEntry) )

                    {

                        kernel->entryList->Append(pageEntry);

                    }

                    else

                    {

                        kernel->entryList->Prepend(pageEntry);

                    }

                }

            }

            return;

            ASSERTNOTREACHED();

        }break;

# Implement user programs in test directory:

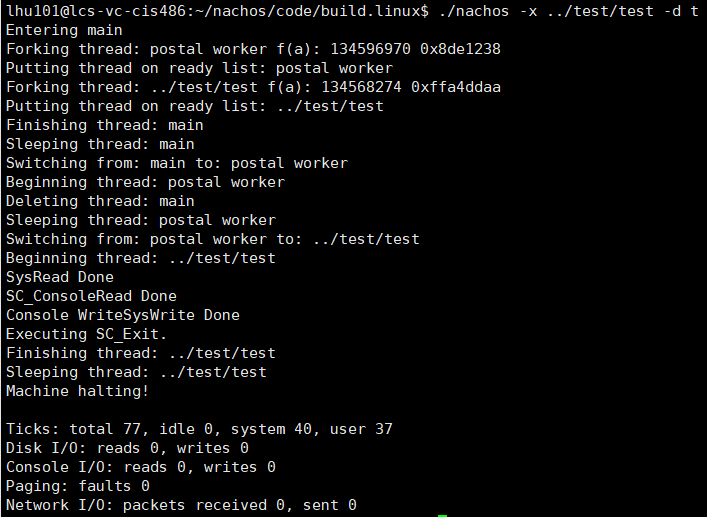
## Task 1, run a single user program.

### Test.c:

Cmd: “./nachos -x ../test/test -d t”

Decription: In this file, the main function executes three syscall: ConsoleRead, ConsoleWrite, Exit.

Snapshot:

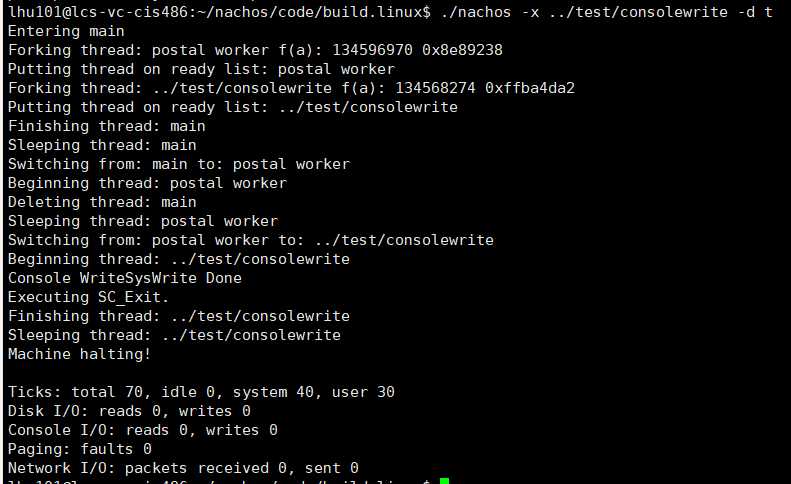


### Consolewrite.c:

Cmd: “./nachos -x ../test/consolewrite -d t”

Decription: In this file, the main function executes syscall consolewrite() one time .

Snapshot:

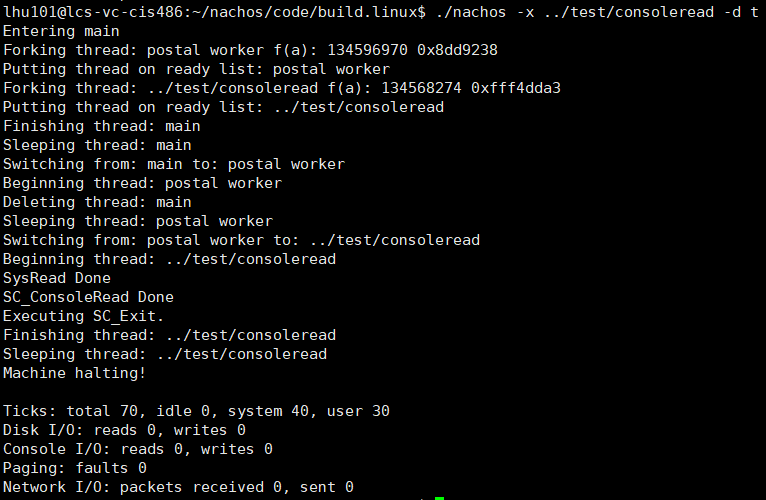


### Consoleread.c:

Cmd: “./nachos -x ../test/ consoleread -d t”

Decription: In this file, the main function executes syscall consoleread() one time .

Snapshot:

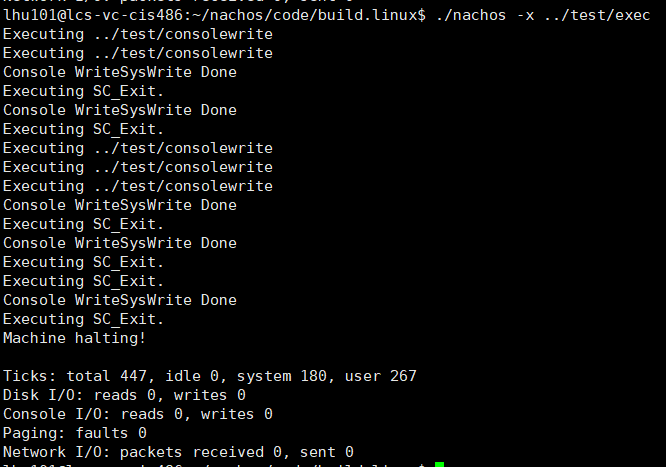


### Exec.c:

Cmd: “./nachos -x ../test/ exec -d t”

Decription: In this file, the main function executes syscall exec() five time and exit(). The number of times is set in the main function.

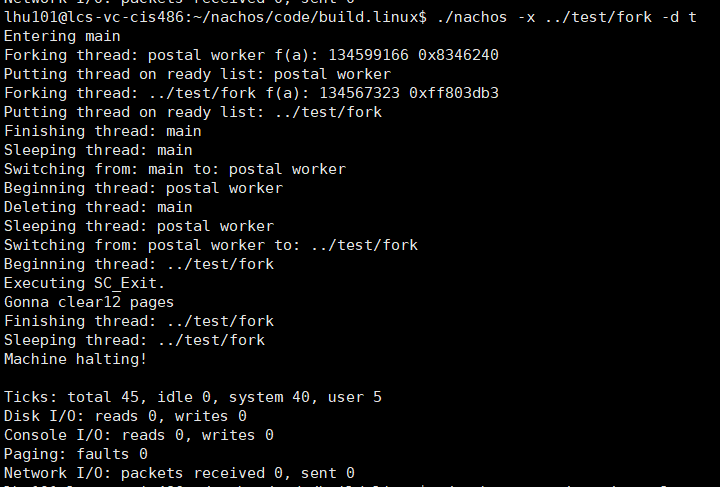
Snapshot:



Fork.c:

Cmd: “./nachos -x ../test/fork -d t”

Snapshot:

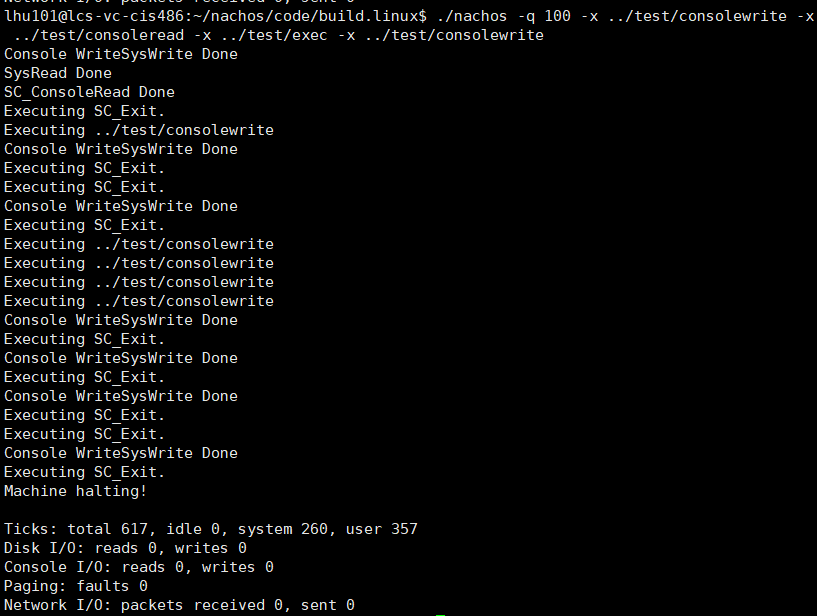


## Task 2, run multiple programs.

Cmd: ” ./nachos -q 100 -x ../test/consolewrite -x ../test/consoleread -x ../test/exec -x ../test/consolewrite”

Decription: In this cmd, “-q 100” represents the time quantum is 100. In this case, we run a consolewrite, consoleread, exec and then a consolewrite.

Snapshot:

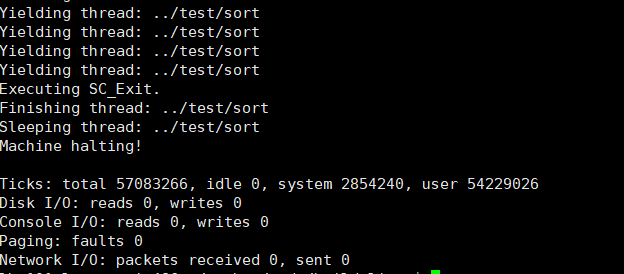


## Task 3, run larger memory need programs.

Cmd: ” ./nachos -q 200 -x ../test/matmult -x ../test/matmult -x ../test/sort -d t”

Decription: In this cmd, we run three user program: matmult, matmult and sort.

Snapshot: Due to limited page, it’s partial snapshot



Cmd: ” ./nachos -q 200 -x ../test/matmult -x ../test/matmult -x ../test/sort”

Decription: Compared to last one, this time we don’t use “-d” mode.

Snapshot:

