Nachos- Programming Assignment: 4

Shoaib Sheikh – SUID: 631608148

File changed:

/home/ssheikh/student/Nachos/code/userprog/exception.cc

Task -1

1) Implement Read and Write for Console:

Write

Code Snippet:

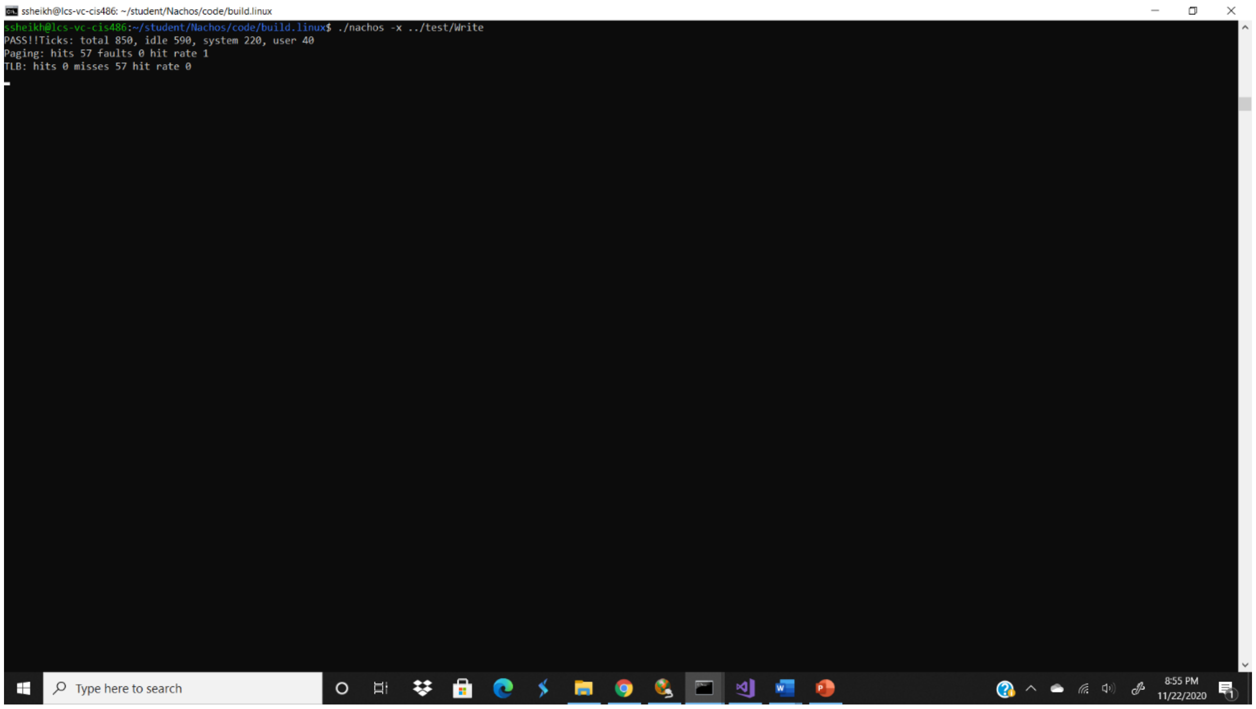
```
/*Exception call for Write */
  case SC_Write:
        DEBUG(dbgSys, "Write" << ", " <<kernel->machine->ReadRegister(4) << ", " << kernel->machine->ReadRegister(5) << ", " << kernel->machine->ReadRegister(6));
        char wcount; //word count
        int wsize, wbuffer, ws, wopenfileId; // word size, word buffer, word size and openfile id
        SynchConsoleOutput *p;
        wbuffer = (int)kernel->machine->ReadRegister(4); //reading from the register
        wsize = (int)kernel->machine->ReadRegister(5); //reading from the register
        wopenfileId = (int)kernel->machine->ReadRegister(6); //reading from the register
        if(wopenfileId == 1)
```

```
          {
          p = kernel->synchConsoleOut;
          for(ws = 0; ws<wsize ; ws++) // Assigning the input value from the console to the
main memory using a for loop
               {
                    wcount = kernel->machine->mainMemory[wbuffer+ws];
                    p->PutChar(wcount);
               }
          }
     kernel->machine->WriteRegister(2,ws); //writing to 2 register
          {
          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;
```

Description:

In the above implementation of the Write system call, I declared 3 integer values for the word size, word buffer, ws(which is used for the for loop to write to the main memory) and openfile id openfile id = 0 using the Macro CONSOLEOUTPUT 0 from syscall.h header file to differentiate with the Console input. All these elements are used to read from the register. An if statement is used to check if we have the right fileid and the SynchConsoleOut(to initialize the console device) from element p is used with the PutChar(to write a character) from the syscall.h. We are then able to write to the 2nd register with the ws. We then change the program counter for the next instruction and then next program counter for the branch execution.

Result:



```
ssheikh@lcs-vc-cis486: ~/student/Nachos/code/build.linux
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$ ./nachos -x ../test/Write
PASS!!Ticks: total 850, idle 590, system 220, user 40
Paging: hits 57 faults 0 hit rate 1
TLB: hits 0 misses 57 hit rate 0
```

As seen from the above test case from the test directory. The System call for Write passes. As seen from the above screenshot without the Halt() function the test does not exit.

SC_Read

Code Snippet:

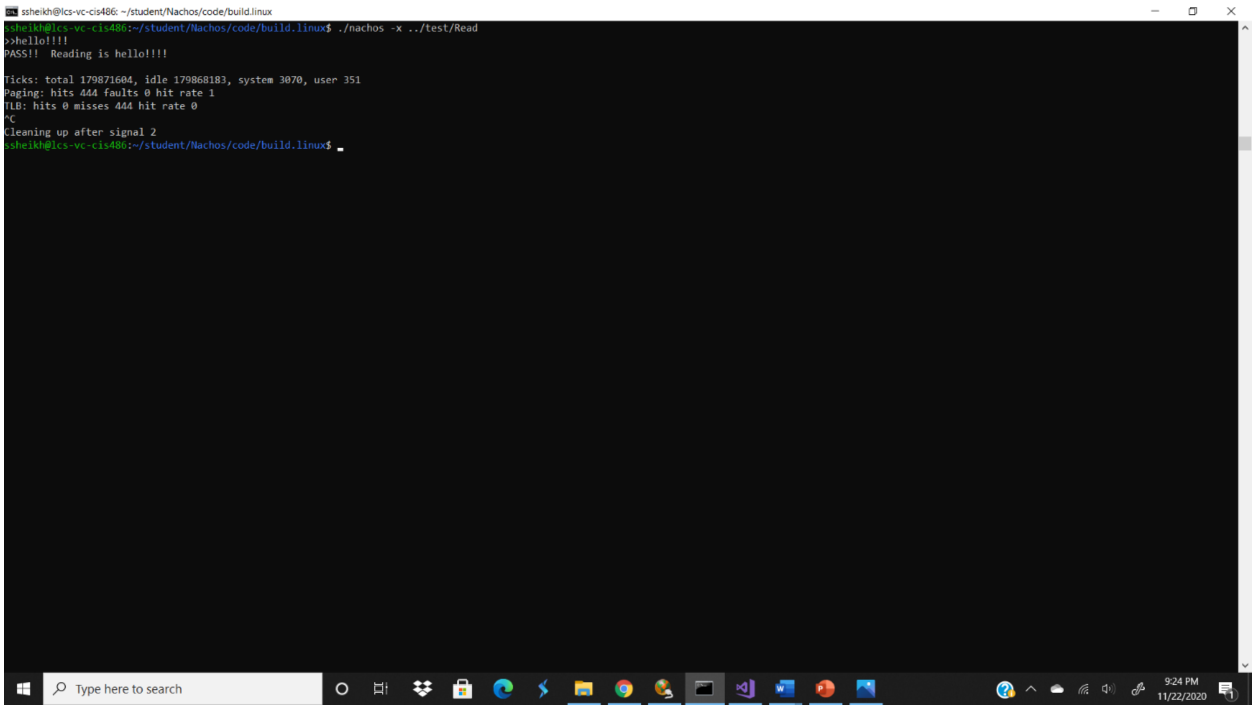
```
  //Exception handling of SC_Read
        case SC_Read:
        DEBUG(dbgSys, "Read" << ", " <<kernel->machine->ReadRegister(4) << ", " << kernel->machine->ReadRegister(5) << ", " << kernel->machine->ReadRegister(6));
        char readc;
        int readsize, readbuff, reads, readopenfileId;
        SynchConsoleInput *pr;

        readbuff = (int)kernel->machine->ReadRegister(4); //Read from register
        readsize = (int)kernel->machine->ReadRegister(5); //Read from register
        readopenfileId = (int)kernel->machine->ReadRegister(6); //Read from register
```

```
    if(readopenfileId == 0) //if the open file id == 0
        {
        pr = kernel->synchConsoleIn; //take input from console
        for(reads = 0; reads<readsize ; reads++) //for loop for reading from the main memory
            {
                readc = pr->GetChar();
                kernel->machine->mainMemory[readbuff+reads]=readc;
                if(readc == '\n') { break ; };
            }
        DEBUG(dbgSys, "Read: openfileid = 0, Read from console \n"); //debug statement if
the syscall is handled
        }
    else
        {
        DEBUG(dbgSys, "Read: openfileid = 0, Read from non-console not handled\n");
//debug statement if the syscall is not handled
        }
    kernel->machine->WriteRegister(2,reads); //Write to 2 register
        {
        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;
```

Description:

In the above implementation of the Read system call, I declared 3 integer values for the word size, word buffer, rs(which is used for the for loop to read from the main memory) and openfile id = 1 using the Macro CONSOLEINPUT 1 from syscall.h header file to differentiate with the Console input. All these elements are used to read from the register. An if statement is used to check if we have the right fileid and the SynchConsoleOut(to initialize the console device) from element p is used with the PutChar(to write a character) from the syscall.h header file. We are then able to write to the 2nd register with the reads. DEBUG statements were used to handle if thE read from console is called or not. We then change the program counter for the next instruction and then next to the program counter for the branch execution.

Result:



```
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$ ./nachos -x ../test/Read
>>hello!!!!
PASS!!  Reading is hello!!!!

Ticks: total 179871604, idle 179868183, system 3070, user 351
Paging: hits 444 faults 0 hit rate 1
TLB: hits 0 misses 444 hit rate 0
^C
Cleaning up after signal 2
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$
```

2) Implementing Fork System Call

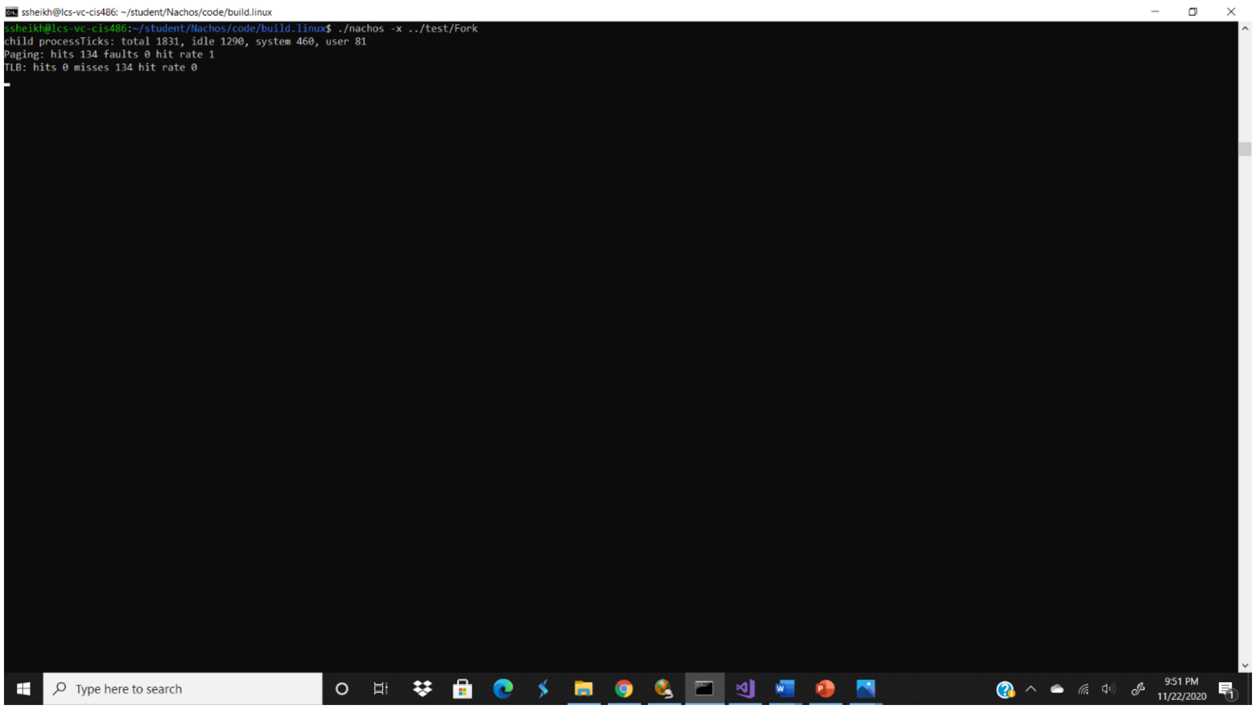
Code Snippet:

```
// Implementing System call for Fork
        case SC_SysFork:


        {
        Thread* thread; // assigning a thread element
        thread = new Thread("Fork");
        AddrSpace* space = new AddrSpace(); //assiging a new address space
        thread->space = space;
        kernel->machine->WriteRegister(2,0); // Writing to the 2 register
        thread->SaveUserState(); // saving the user state of the thread
          {
         /* 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);
        }


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

Description:

We assign a thread element and call a new Thread. We assign a new address space for the child process and assign address space to thread space. The code was supposed to make a fork exec by which it would create 2 pids one for the child process(0) and one for the parent but in my case I was able to only create one process which can only runs the child process.

Result:



```
ssheikh@lcs-vc-cis486: ~/student/Nachos/code/build.linux
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$ ./nachos -x ../test/Fork
child processTicks: total 1831, idle 1290, system 460, user 81
Paging: hits 134 faults 0 hit rate 1
TLB: hits 0 misses 134 hit rate 0
```

3) Implementing Exec System Call

Code Snippet:

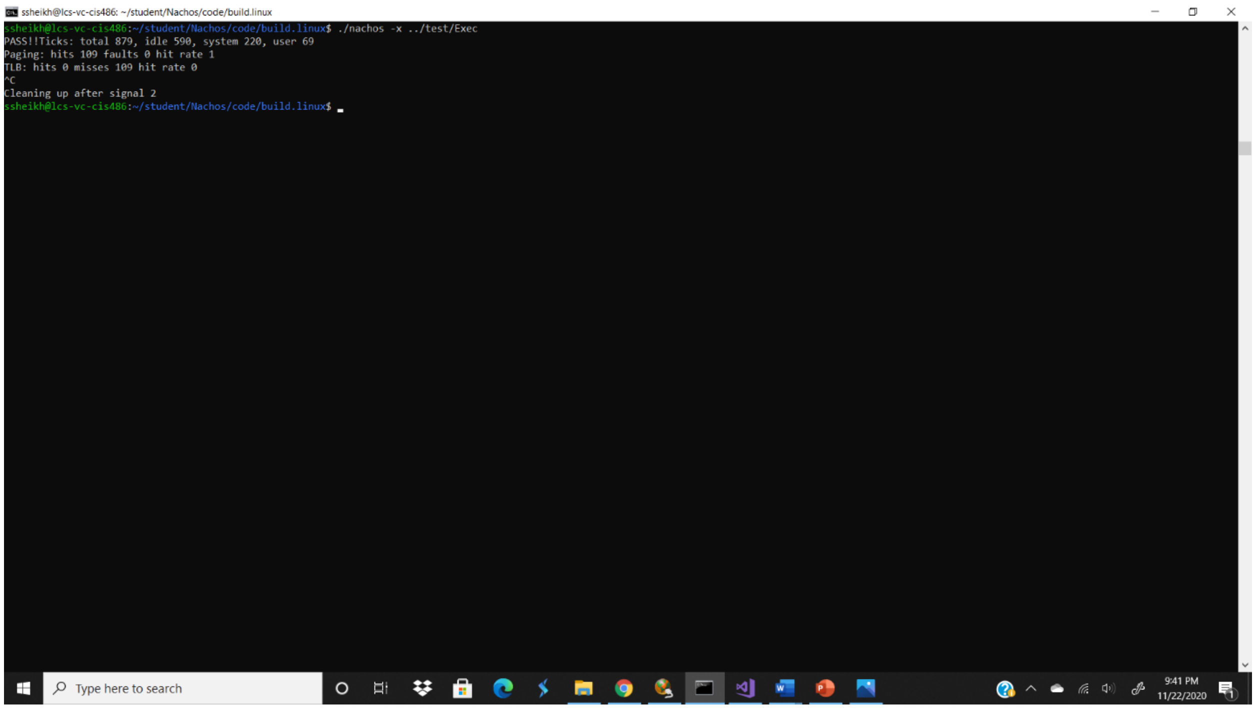
```
// Implementing the System call for Exec

case SC_Exec:

                DEBUG(dbgSys, "Exec" << " \n");

        {

                char execBuffer[80], temp=1;

                int i;

                for (i=0; (i<80) && (temp!='\0') ;i++)

                        {
```

```
                temp = kernel->machine->mainMemory[kernel->machine->ReadRegister(4)+i];
                execBuffer[i]= temp;
               }
          execBuffer[i] = '\0';
           delete kernel->currentThread->space;
           kernel->currentThread->space = new AddrSpace ;
           kernel->currentThread->space->Load(execBuffer);
           kernel->currentThread->space->RestoreState(); // set the kernel page table
      {
       /* set previous programm counter (debugging only)*/
       kernel->machine->WriteRegister(PrevPCReg, 0);
       /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
       kernel->machine->WriteRegister(PCReg, 4);
       /* set next programm counter for brach execution */
       kernel->machine->WriteRegister(NextPCReg, 4);
      }
       kernel->currentThread->space->Execute();
      }
       return;
       ASSERTNOTREACHED();
       break;
```

Description:

In the implementation of the Exec System call, I first initialized a buffer of size 80 and a temporary character which is used to test the null termination character in the for loop which is used to execute the thread. The buffer is loaded with the values which are from the main memory which is read from the register and the last element of the buffer is then null terminated to end the process. We then delete the space which the current thread in the system and then load the current thread with the exec buffer after which we restore the state of the thread.

Result:



ssheikh@lcs-vc-cis486: ~/student/Nachos/code/build.linux

```
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$ ./nachos -x ../test/Exec
PASS!!Ticks: total 879, idle 590, system 220, user 69
Paging: hits 109 faults 0 hit rate 1
TLB: hits 0 misses 109 hit rate 0
^C
Cleaning up after signal 2
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$
```

4) Implementing Exit System Call

Code Snippet:

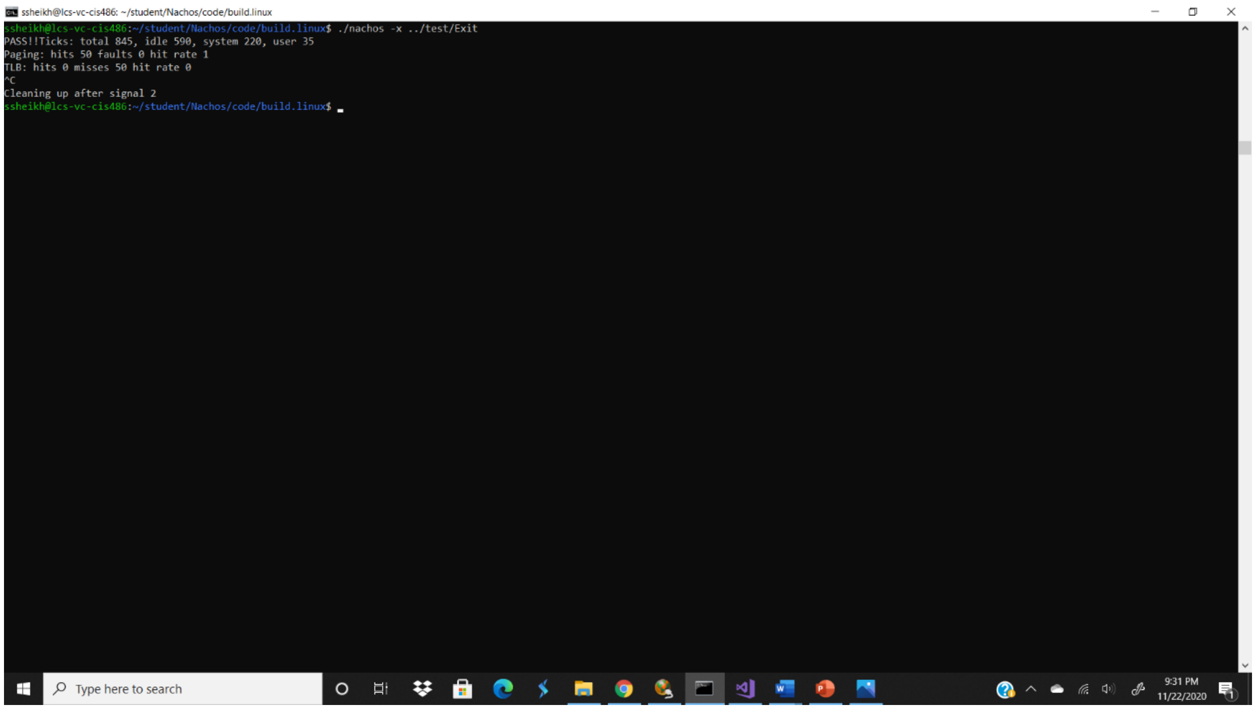
```
//Exception Handling for Exit

case SC_Exit:

        DEBUG(dbgSys, "Exit Called" << " \n");

       {

        int exitstat = (int) kernel->machine->ReadRegister(4); // exit status

        kernel->stats->Print(); //printing the status of the process

        kernel->currentThread->Finish(); //Finishing the current thread to exit the process

       {

        /* 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);
 }
kernel->machine->WriteRegister(2, exitstat);//Writing to the register to exit the process
 }
 return;
 ASSERTNOTREACHED();
 break;
```

Description:

In the exit system call we are initializing an integer with the value of thread which is read from the register. We are printing the status of the process. Then calling the Finish function from the current thread will exit the process.

Result:



```
ssheikh@lcs-vc-cis486: ~/student/Nachos/code/build.linux
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$ ./nachos -x ../test/Exit
PASS!!Ticks: total 845, idle 590, system 220, user 35
Paging: hits 50 faults 0 hit rate 1
TLB: hits 0 misses 50 hit rate 0
^C
Cleaning up after signal 2
ssheikh@lcs-vc-cis486:~/student/Nachos/code/build.linux$ 
```