Operating System Project 1

何孟遠
R11222007
October 15, 2023

Part 1

- Why the result is not congruent with the expected?

Since the initialization of the memory is ruled by the function "AddrSpace()", the original code only performs the same linear transformation of the memory table.

```
pageTable[i].virtualPage  = i;   // for now, virt page # = phys page #
pageTable[i].physicalPage = i;
```

Since the code doesn't know which physical table has been used by another execution file, the two execution files have access to the same memory table when they are executing.

As a result, the "test2" will capture the result of "test1", and vice versa, which is the reason for the bugs.

```
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:6
Print integer:7
Print integer:8
Print integer:9
Print integer:10
Print integer:12
Print integer:13
Print integer:14
Print integer:15
Print integer:16
Print integer:16
Print integer:17
Print integer:18
Print integer:19
Print integer:20
Print integer:17
Print integer:18
Print integer:19
Print integer:20
Print integer:21
Print integer:21
Print integer:23
Print integer:24
Print integer:25
return value:0
Print integer:26
```

- How you really modified Nachos, including some important code and descriptions.

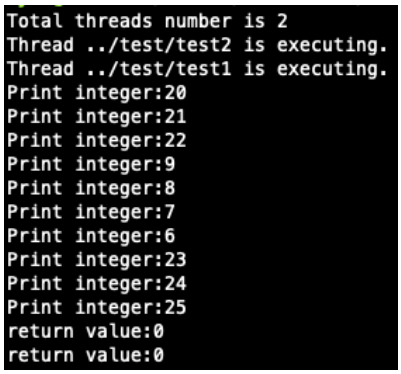
To avoid the codes using the same memory table, we need to redefine the translation table and create a flag to label whether it is in use.

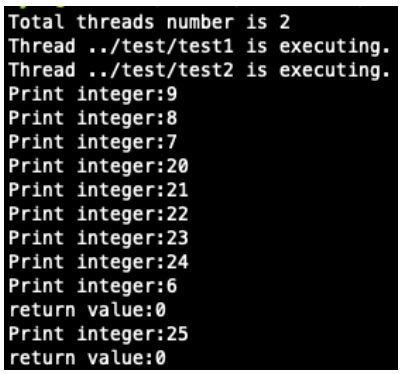
```
pageTable = new TranslationEntry[numPages];
for (unsigned int i = 0, j = 0; i < numPages; i++) {
    pageTable[i].virtualPage = i;   // for now, virt page # != phys page #

    while ( j < NumPhysPages && AddrSpace::UsedPhyPages[j] == true )
        j++;  // if the phypages have been used, just choose the next
    pageTable[i].physicalPage = j;
    AddrSpace::UsedPhyPages[j] = true;
    ... // below is the same as AddrSpace::AddrSpace()
    ...
    ...
}
```

Finally, in the execution phase, redirect the memory address for the code and data segments to use the memory more efficiently.

```
unsigned int PhysAddr = pageTable[
    noffH.code.virtualAddr / PageSize ].physicalPage * PageSize +
    noffH.code.virtualAddr % PageSize;  // this is for code

executable->ReadAt(
                &(kernel->machine->mainMemory[PhysAddr]),
                        noffH.code.size, noffH.code.inFileAddr);
// For "initData", just replace the text "code" above
```

- Experiment results and some analysis.
    - The results are correct no matter which executes first.
    - Two threads are executing at the same time, this can be shown by the below figures that the integers printed are not continuous.





    - If we execute the same file twice, the code will print every integer twice.

Part 2

Please explain the details of each function call as much as you can.

- AddrSpace::Execute()
  - at ../userprog/addrspace.cc:188
  - Run a user program.
- Machine::Run()
  - at ../machine/mipssim.cc:62
  - Execute every command or instruction inside the program.
- Machine::OneInstruction()
  - at ../machine/mipssim.cc:558
  - Execute a single instruction inside the program.
  - Pass the information to the function “RaiseException”.
    - RaiseException(SyscallException, 0);
  - Every time the function gets an interruption or exception, we run the below functions to identify it. After the return, the function returns again to Run() to restart the execution.
- Machine::RaiseException()
  - at ../machine/machine.cc:109
  - Set the kernel to “System Mode” to run the interruption.
    - kernel->interrupt->setStatus(SystemMode);
  - Pass “SyscallException” to the function ExceptionHandler().
    - ExceptionHandler(which);
  - Set the kernel back to “User Mode”.

- ExceptionHandler()
  - at ../userprog/exception.cc:62
  - Get the type of exception call and print the message.
    - If the system calls, split it into four cases.
      - SC_Halt: This case.
      - SC_PrintInt
      - SC_Exec
      - SC_Exit
    - Or, print “the unexpected call by the user”.
- Interrupt::Halt()
  - at ../machine/interrupt.cc:236
  - Print the info: "Machine halting!\n\n"