Project 1 Report

電機碩二　r06921048　李友岐

1. Why the result is not congruent with expected:

```
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
return value:0
```

The error occurred because of address management. We executed test1 and test2 on two different threads but they loaded same physical address during the process. In test2, the number is increasing from 20. It loaded, however, the data equal to 6, which is from test1. Consequently, the number is increasing from 6. The result is different with what we expect due to the fact that the physical address they used is overlapped.

2. The plan you take to fix the problem in Nachos:

To fix this problem, we need to record whether the physical address has been accessed by a global array. With this method, we can always make sure the loaded physical address is unused and consequently the physical address that test1 and test2 access won't overlap.

3. How you really modified Nachos, including some (not all) important code segments and comments:

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

    void Execute(char *fileName);       // Run the the program
                                        // stored in the file "executable"
    static bool isUsedPhyPage[NumPhysPages];
                                        // Record whether the main memory page
                                        // is used.
```

First, we announce a static member called isUsedPhyPage in “addrspace.h”. It is used to record whether the main memory page is accessed.

```
// Initialize isUsedPhyPage to FALSE
bool AddrSpace::isUsedPhyPage[NumPhysPages]={0};
```

Next, we initialize the value of each element of the static member to zero (FALSE) in “addrspace.cc”.

```
AddrSpace::~AddrSpace()
{
   unsigned int j=0;
   while (j< numPages) {
         AddrSpace::isUsedPhyPage[pageTable[j].physicalPage]=FALSE;
         j++;
   }
   delete pageTable;
}
```

In destructor, we should change back the value of elements in isUsedPhyPage corresponding to each page to FALSE before deleting pageTable, which means this physical memory are not used.

```
  pageTable = new TranslationEntry[numPages];
  for (unsigned int  j= 0, k=0; j < numPages; j++) {
      pageTable[j].virtualPage = j;   // for now, virt page # = phys page #
      while ( true){
         if (k< NumPhysPages and AddrSpace::isUsedPhyPage[k] ){
             k++;
         }
         else {
             break;
         }
      }
      AddrSpace::isUsedPhyPage[k] = TRUE;
      pageTable[j].physicalPage = k;
      pageTable[j].valid = TRUE;
      pageTable[j].use = FALSE;
      pageTable[j].dirty = FALSE;
      pageTable[j].readOnly = FALSE;
  }
```

In Addrspace::Load(char *fileName), when we load the memory, we need to find a physical address space that is not used. Therefore, we add a while loop to accomplish this.

```
int pageBase=pageTable[ noffH.code.virtualAddr / PageSize ].physicalPage*PageSize;
int pageOffset= noffH.code.virtualAddr % PageSize;
executable->ReadAt(
        &(kernel->machine->mainMemory[ pageBase + pageOffset ] ),
        noffH.code.size, noffH.code.inFileAddr);

}
```

```
int pageBase=pageTable[ noffH.initData.virtualAddr / PageSize ].physicalPage*PageSize;
int pageOffset= noffH.initData.virtualAddr % PageSize;
executable->ReadAt(
        &(kernel->machine->mainMemory[ pageBase+pageOffset ] ),
        noffH.initData.size, noffH.initData.inFileAddr);
}
```

Finally, we calculate the memory address, which is equal to the sum of pageBase and pageOffset. The pageBase is equal to the product of page number and page size. The pageOffset is equal to the code address mod page size.

4. Experiment result and some analysis:

(1)

```
./userprog/nachos -e test/test1 -e test/test2
```

```
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
return value:0
Print integer:25
return value:0
```

After fixing the bug, we observe that the result is correct. test1 is decreasing from 9 to 6 while test2 is increasing from 20 to 25. we also find that part of the numbers of test1 (9, 8, 7) are first output and then the first number of test2 (20) outputs. The outputs of two tests are crossed with each other.

(2)

```
./userprog/nachos -e test/test2 -e test/test1
```

```
Print integer:20
Print integer:21
Print integer:22
Print integer:9
Print integer:8
Print integer:7
Print integer:6
Print integer:23
Print integer:24
Print integer:25
return value:0
return value:0
```

Now we change the order of two tests. As a result, part of the numbers of test2 first output and the numbers of test1 output then. We can find that the result is still correct. The outputs of two tests are crossed with each other.

test/test3.c (self-defined)

```
int     n;
for (n=10;n<14;n++)
        PrintInt(n*n);
```

To do more experiment, we define another test, which is test3. it would output 100, 121, 144 and 169.

(3)

```
./userprog/nachos -e test/test1 -e test/test3
```

```
Print integer:9
Print integer:8
Print integer:7
Print integer:100
Print integer:121
Print integer:144
Print integer:169
Print integer:6
return value:0
return value:0
```

In this experiment, part of the numbers of test1 first output and then the numbers of test3 output. Still, the result is correct. The outputs of two tests are crossed with each other.

(4)

```
./userprog/nachos -e test/test2 -e test/test3
```

```
Print integer:20
Print integer:21
Print integer:22
Print integer:100
Print integer:121
Print integer:144
Print integer:169
Print integer:23
Print integer:24
Print integer:25
return value:0
return value:0
```

In this experiment, part of the numbers of test2 first output and then the numbers of test3 output. Still, the result is correct. The outputs of two tests are crossed with each other.