Operating Systems
Project 1 - Thread Management

機械所 吳冠甫 R10522532

October 21th, 2022

Motivation

Issue

From the source code of `nachos-4.0/code/test/test1.c` (see Listing 8), the result should be a series of $9, 8, 7, 6$ (see Listing 9), and the result of `nachos-4.0/code/test/test2.c` (see Listing 10) should be a series of $20, 21, 22, 23, 24, 25$ (see Listing 11).
However, when executing `test1` and `test2` simultaneously by `./nachos -e ../test/test1 -e ../test/test2`, the result is not what we want (see Listing 12).

Problem analysis

Because the mapping of `nachos` is 1 to 1 since it only support uniprogramming by default, we have only a single unsegmented page table (address translation table). All task will use the same page table and therefore will map to the same physical page.
If we want to excute the multiprogramming command to get desired result, we must change this scheme. Because the process will execute the same code segment in multiprogramming, we have to modify the process of page table creation in `nachos-4.0/code/userprog/addrspace.h` and `nachos-4.0/code/userprog/addrspace.cc`.

Implementation

Record usage of physical pages

In the constructor of `AddrSpace`, the virtual and physical page has the same number. Thus, I add a `static bool` array in `addrspace.h` to record usage of physical pages , and instantiate the array in `addrspace.cc`

```
22 class AddrSpace {
23     ...
24 private:
25     ...
26 // --- Add ------------------------------------------------------
27     static bool UsedPhyPages[];
28 // --------------------------------------------------------------
29 }
```

Listing 1: addrspace.h

```
31 // --- Add ------------------------------------------------------
32 bool AddrSpace::UsedPhyPages[NumPhysPages] = {FALSE};
33 // --------------------------------------------------------------
```

Listing 2: addrspace.cc

```
81 AddrSpace::~AddrSpace()
82 {
83 // --- Add ------------------------------------------------------
84     // Free memory space
85     for (unsigned int i = 0; i < numPages; ++i) {
86         AddrSpace::UsedPhyPages[pageTable[i].physicalPage] = FALSE;
87     }
88 // --------------------------------------------------------------
89     delete[] pageTable;
90 }
```

Listing 3: addrspace.cc

Find the first unused physical page

When executing, each process correspond to `AddrSpace` instance. In multiprogramming, we have to keep track of the corresponding information of physical page (`pageTable[i].physicalPage`). Thus, when we execute a certain process, the operating system will find the specific physical page in the page table.
In the original code, every process shares the same physical page initially, and leads to the result which is not as expected in multilprogramming (Listing 12).
When the process load into memory, page table will record the corresponding physical page. I create a page table in `addrspace.cc`, and modify the function `AddrSpace::Load()` to find the first unused physical page and update the page of the process.

```
127 // --- Add ----------------------------------------------------------------
128 pageTable = new TranslationEntry[numPages];
129 for (unsigned int i = 0; i < numPages; ++i){
130     unsigned int j = 0;
131     pageTable[i].virtualPage = i;
132     while ((j < NumPhysPages) && (AddrSpace::UsedPhyPages[j] == TRUE)) {
133         j++;
134     }
135     ASSERT(j < NumPhysPages);
136     pageTable[i].physicalPage  = j;
137     AddrSpace::UsedPhyPages[j] = TRUE;
138     pageTable[i].valid         = TRUE;
139     pageTable[i].use           = FALSE;
140     pageTable[i].dirty         = FALSE;
141     pageTable[i].readOnly      = FALSE;
142 }
143 // ----------------------------------------------------------------------
```

Listing 4: addrspace.cc

Find the physical address

In the excecution, we have to calculate the physical address from the virtual memory address. The relation can be calculate as physical page number + page offset.

```
155 if (noffH.code.size > 0) {
156     DEBUG(dbgAddr, "Initializing code segment.");
157     DEBUG(dbgAddr, noffH.code.virtualAddr << ", " << noffH.code.size);
158
159 // --- Add ------------------------------------------------------------
160     // Find the physical address (= physical page number + offset)
161     unsigned int phyAddr =
162         pageTable[noffH.code.virtualAddr / PageSize].physicalPage * PageSize
163         + noffH.code.virtualAddr % PageSize;
164 // --------------------------------------------------------------------
165
166     executable->ReadAt(
167         &(kernel->machine->mainMemory[phyAddr]),
168         noffH.code.size, noffH.code.inFileAddr);
169 }
```

Listing 5: addrspace.cc

```
170 if (noffH.initData.size > 0) {
171     DEBUG(dbgAddr, "Initializing data segment.");
172     DEBUG(dbgAddr, noffH.initData.virtualAddr << ", " << noffH.initData.size);
173
174 // --- Add ------------------------------------------------------------
175     // Find the physical address (= physical page number + offset)
176     unsigned int phyAddr =
177         pageTable[noffH.initData.virtualAddr / PageSize].physicalPage * PageSize
178         + noffH.code.virtualAddr % PageSize;
179 // --------------------------------------------------------------------
180
181     executable->ReadAt(
182         &(kernel->machine->mainMemory[phyAddr]),
183         noffH.initData.size, noffH.initData.inFileAddr);
184 }
```

Listing 6: addrspace.cc

Result

After these modification, recompile `nachos` again and execute test by `./nachos -e ../test/test1 -e ../test/test2`. We can get desired results (see Listing 7).

```
Total threads number is 2
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
return value:0
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 300, idle 8, system 70, user 222
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Listing 7: desired result

Appendix

test1

```
#include "syscall.h"

main()
    {
        int     n;
        for (n=9;n>5;n--)
            PrintInt(n);
    }
```

Listing 8: test1.c

```
Total threads number is 1
Thread ../test/test1 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:6
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 200, idle 66, system 40, user 94
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Listing 9: result of test1

Appendix continued on next page...

test2

```c
#include "syscall.h"

main()
    {
        int     n;
        for (n=20;n<=25;n++)
            PrintInt(n);
    }
```

Listing 10: test2.c

```
Total threads number is 1
Thread ../test/test2 is executing.
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 200, idle 32, system 40, user 128
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Listing 11: result of test2

Appendix continued on next page...

Original Result

```
Total threads number is 2
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
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 800, idle 67, system 120, user 613
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Listing 12: original result