國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Institute of Artificial Intelligence Innovation
Department of Computer Science

Operating System

Homework 02: Memory Management

Shuo-Han Chen (陳碩漢)
shch@nycu.edu.tw

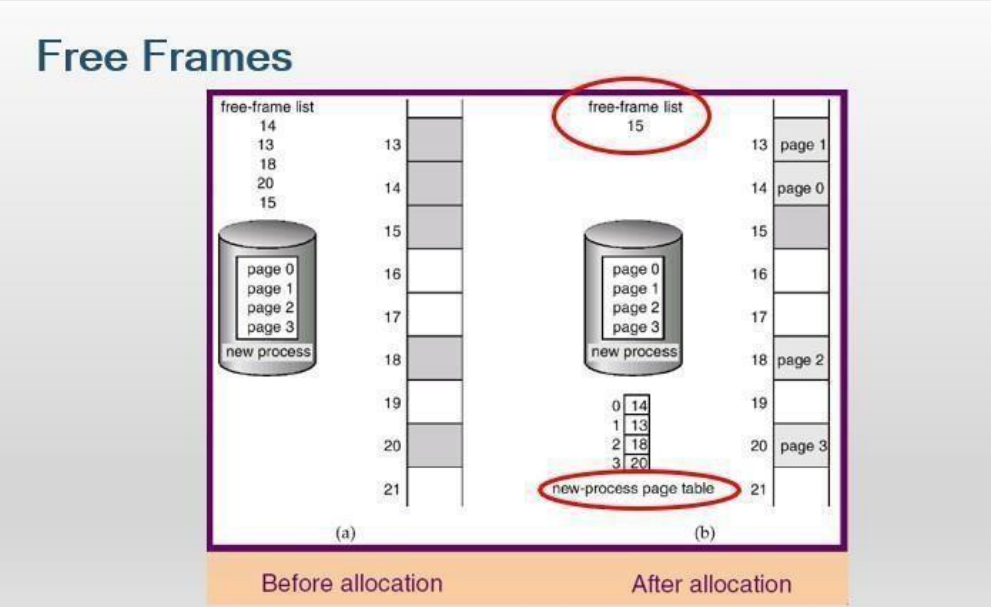
Wed. 10:10 - 12:00 EC115 +
Fri. 11:10 – 12:00 Online

Some Changes in this Homework

- The Implementation of ConsoleIO on page 8.
  - So you won't get blocked if HW01 does not work out for you.
- Relax Report Format
- More hints in source code
  - You may see "// TODO" in the files to modify
  - TA will provide another package of source code
- More explanation of the relation between the classes in NachOS
- Remove report questions from Page 5
- Remove exception handling for insufficient memory

Goal

1. Understand how memory management works in NachOS
2. Understand how to implement page table mechanism



Part 1 Trace Code

- Starting from “threads/kernel.cc Kernel::ExecAll()”, “threads/thread.cc thread::Sleep()” until “machine/mipssim.cc Machine::Run()” is called for executing the first instruction from the user program.
- You need to explain at least the function mentioned below in the report.
  1. threads/thread.cc
     - Thread::Sleep(), Thread::StackAllocate(), Thread::Finish(), Thread::Fork()
  2. userprog/addrspace.cc
     - AddrSpace::AddrSpace(), AdrSpace::Execute(), AddrSpace::Load()
  3. threads/kernel.cc
     - Kernel::Kernel(), Kernel::ExecAll(), Kernel::Exec(), Kernel::ForkExecute()
  4. threads/scheduler.cc
     - Scheduler::ReadyToRun(), Scheduler::Run()

Part 1 Trace Code (removed)

- It will help your implementation a lot if you know the answer of the following
  1. Explain how NachOS creates a thread (process), loads it into memory, and places it into the scheduling queue.
  2. How does Nachos allocate the memory space for a new thread(process)?
  3. How does Nachos initialize the memory content of a thread(process), including loading the user binary code in the memory?
  4. How does Nachos create and manage the page table?
  5. How does Nachos translate addresses?
  6. How does Nachos initialize the machine status (register, etc) before running a thread (process)?
  7. Which object in Nachos acts as the process control block?

Part 2 Implementation

- Modify its memory management code to make NachOS support multi-programming.

- Without multi-programming

```
../build.linux/nachos -e consoleIO_test1 -e consoleIO_test2
consoleIO_test1
consoleIO_test2
9
16
15
18
19
17
return value:0
return value:0
```

- After implement multi-programming

```
../build.linux/nachos -e consoleIO_test1 -e consoleIO_test2
consoleIO_test1
consoleIO_test2
9
8
7
6
15
return value:0
16
17
18
19
return value:0
```

Requirements

- Implement a data structure to record used physical memory.
- You must set up “valid, readOnly, use, and dirty” field for your page table, which is defined under “TranslationEntry class” in translate.h
- Allocate the physical memory correctly.
- ~~You must call ExceptionHandler to handle the exception when there is insufficient memory for a thread. (i.e. MemoryLimitException). Since there is no MemoryLimitException in ExceptionType class, you need to add it in machine.h and place it right before NumExceptionTypes. We will use hidden testcases to test this part.~~
- When the thread is finished, make sure to release the address space and restore physical page status.

Part 2 Prerequisite

- Before you start implement this homework feature, you need to ensure your previous NachOS assignments meet the following requirements.
  1. ConsoleIO implementation
  2. consoleIO_test file modification
  3. test scripts on Jenkins

ConsoleIO Implementation

- If you don't follow the correct method for printing an integer, the result will also be incorrect. The following code demonstrates one of the proper ways to do it.

- userprog/exception.cc

```
case SC_PrintInt:
    val = kernel->machine->ReadRegister(4);
    SysPrintInt(val);
    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();
```

- userprog/ksyscall.h

```
void SysPrintInt(int value)
{
    kernel->interrupt>PrintInt(value);
}
```

- userprog/synchconsole.h

```
class SynchConsoleOutput : public CallBackObj
{
public:
    ...
    void PutChar(char ch);
    void PutInt(int value);
    ...
};
```

- userprog/synchconsole.cc

```
void
SynchConsoleOutput::PutInt(int value)
{
    lock->Acquire();
    consoleOutput->PutInt(value);
    waitFor->P();
    lock->Release();
}
```

- userprog/syscall.h

```
#define SC_PrintInt      16

void PrintInt(int value);
```

ConsoleIO Implementation

- If you don't follow the correct method for printing an integer, the result will also be incorrect. The following code demonstrates one of the proper ways to do it.

- machine/console.h

```
class ConsoleOutput : public CallBackObj {
public:
    ...
    void PutChar(char ch);
    void PutInt(int n);
    ...
};
```

- machine/console.cc

```
void ConsoleOutput::PutInt(int value) {
    ASSERT(putBusy == FALSE);
    char * printStr = (char*)malloc(sizeof(char) * 15);
    sprintf(printStr, "%d\n", value);
    WriteFile(writeFileNo, printStr, strlen(printStr) * sizeof(char));
    putBusy = TRUE;
    kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
}
```

- threads/kernel.h

```
class Kernel{
public:
    ...
    void PrintInt(int number);
    ...
};
```

- machine/interrupt.cc

```
void Interrupt::PrintInt(int value)
{
    return kernel->PrintInt(value);
}
```

- test/start.S

```
    .globl PrintInt
    .ent   PrintInt
PrintInt:
    addiu $2,$0,SC_PrintInt
    syscall
    j   $31
    .end PrintInt
```

- threads/kernel.cc

```
void Kernel::PrintInt(int number)
{
    return  SynchConsoleOut->PutInt(number);
}
```

ConsoleIO_test modification

- consoleIO_test1.c

```c
#include "syscall.h"
int
main()
{
    int n;
    for (n=9;n>5;n--) {
        PrintInt(n);
    }
    Halt();
}
```

before

```c
#include "syscall.h"
int
main()
{
    int n;
    for (n=9;n>5;n--) {
        PrintInt(n);
    }
    // Halt();
    return 0;
}
```

after

- consoleIO_test2.c

```c
#include "syscall.h"
int
main()
{
    int n;
    for (n=15;n<=19;n++){
    PrintInt(n);
    }
    Halt();
}
```

before

```c
#include "syscall.h"
int
main()
{
    int n;
    for (n=15;n<=19;n++){
    PrintInt(n);
    }
    // Halt();
    return 0;
}
```

after

Test Scripts

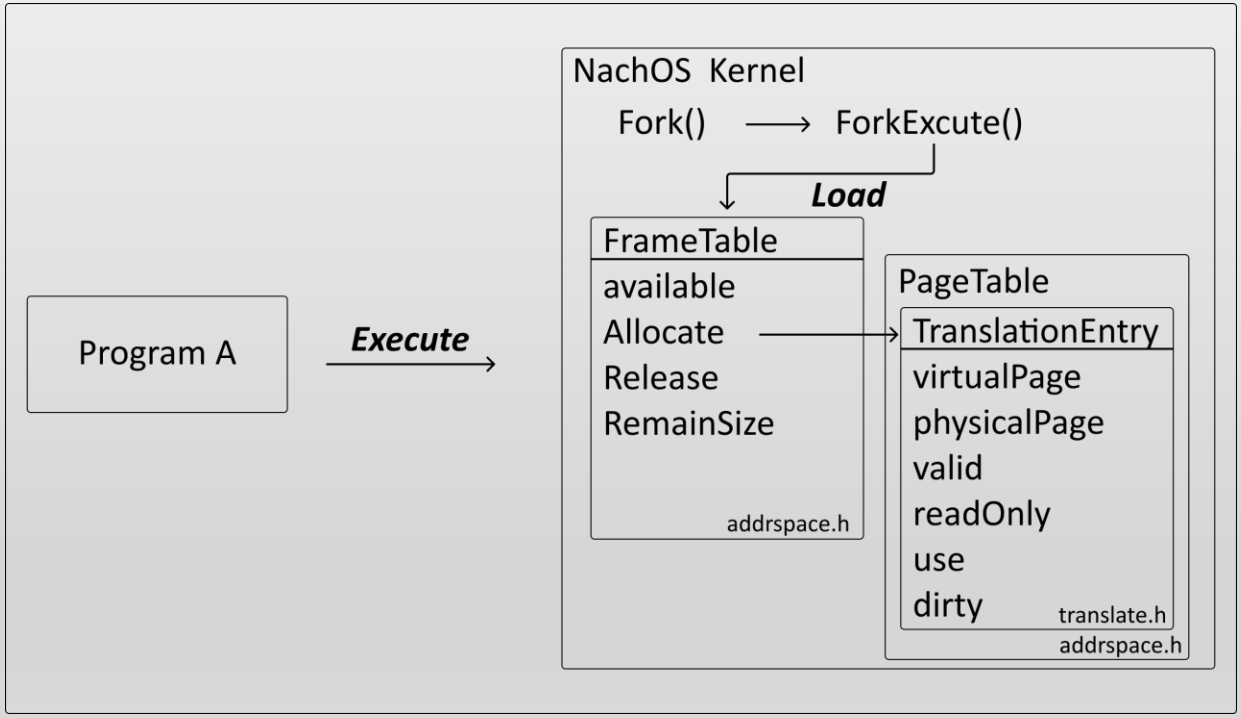
- Since we do not use the Halt command to stop NachOS (as we want to avoid shutting down NachOS after the completion of each user program).
- The NachOS program will not shut down by itself, even after finishing multiple tasks.
- Therefore, in your test scripts, you will need to modify your commands as follows:

```
make distclean
make
timeout 1 ../build.linux/nachos -e consoleIO_test1 -e consoleIO_test2
echo "done"
```

code/test/os_students.sh

Hint

- Implementation details should help you understanding HW02.



Hint

- The following files “may” be modified...
  - userprog/addrspace.*
    - Implement a data structure(FrameTable)
    - Initialize FrameTable.
  - Threads/kernel.h
    - Declare the new data structure you implemented.
  - machine/machine.h
    - Handle “MemoryLimitException”.

Jenkins Verification

- The TA's job will involve running two tests.
  1. The first one uses the following command:
     "../build.linux/nachos -e consoleIO_test1 -e consoleIO_test2"
  2. The output of multiple program could be interleaved.
  3. The second is a hidden test, but you need to ensure that your output contains all the expected values (three ranges of numbers).
  4. It's okay if "return value: 0", "mp2_test1", etc., appear in the console output, but make sure they don't interrupt a printInt system call.

```
==========================
Running the test: 2
==========================
mp2_test3
mp2_test1
mp2_test2
20
22
9
8
24
26
28
15
16
17
7
6
18
return value:0
19
return value:0
30
32
34
36
38
40
return value:0
```

Grading

- Part 1 Trace – 28%
    1. Explain function – 28%
    2. Answer questions of Page 5 – Removed
- Part 2 Implementation – 70%
    1. Each test case – 35%
- Report Format – 2%
- Deadline: 11/2 23:59

Report Format

- Please follow the word file to form your report for HW02
- Format guide
  - Content format: 12pt front, ~~16pt row height~~, and align to the left.
  - Caption format: 18pt and Bold font.
  - ~~Font format: Times New Roman, 標楷體~~
  - ~~Figure: center with single line row height.~~
  - Upload pdf file with the filename:
    - OS_HW02_GROUP_XX.pdf (change XX to your Group Number)

Reminder

- 0 will given to cheaters. Do not copy & paste!
  - TA will check your repository
- Feel free to ask TA questions
  - The TA has created a channel named '**作業討論**'. If you have any questions about the homework, please ask and discuss them in the channel. The records in the channel may help classmates who have similar issues.
  - The TA will not help you debug your code.
- Late submission:
  - 3 days: 90%, 1 week: 80%
  - 2 weeks: 70%, 3 weeks: 60%
  - Further, will not be accepted

Q& A

Thank you for your attention