HW #4

Virtual Memory Simulator

Yunmin Go

School of CSEE

HANDONG GLOBAL UNIVERSITY

Requirements

- Implement a virtual memory simulator for multi-process
  - In this assignment, you will implement a virtual memory simulator.
  - Although the virtual memory simulator you will be implementing in this assignment is highly simplified, through this implementation, you will gain a general understanding of how an operating system executes processes.
  - Please refer to the skeleton code provided through the LMS along with the assignment requirements for implementation.
  - Please read the following requirements carefully.

Requirements

- Program usage

```
Usage: ./vmsim <process image files... up to 5 files>
```

  - The simulator's name is '`vmsim`'.
  - The simulator can load highly simplified process images from text files.
  - The maximum number of processes that the simulator can load is defined as '`MAX_PROCESSES`' in '`vmsim.h`'.
    - The default value for `MAX_PROCESSES` is 5.

Requirements

- Process image
  - Example of ‘proc0.txt’

Size of process image (This process can have 16384 bytes of memory space)

```
16384 9
M 0 10
M 1 20
M 2 30
A 3 0 1
A 4 1 2
S 3 0x1000
S 4 0x2000
L 5 0x1000
L 6 0x2000
```

The number of instructions to execute
(Simulator executes the following 9 instructions)

Instructions to execute
In the simulator, instruction is stored as string in the pseudo physical memory.
Please refer to page 6 and 10.

  - When the simulator starts, the process image (including string of instructions) provided by files are loaded into pseudo physical memory.

Requirements

- Instructions
  - The simulator can execute four types of instructions.
    - ‘M’ type: Move integer value into register
      - ex) `M 0 10` → Move integer 10 to Register#0.

        Reg Integer constant
    - ‘A’ type: Add the integer value of RegSrc1 and the integer value of RegSrc2. The result is stored in RegDst.
      - ex) `A 3 0 1` → Register#3 stores the addition result ‘value in Register#0 + value in Register#1’.

        RegDst RegSrc1 RegSrc2
    - ‘L’ type: Load 4-bytes integer value from the specified memory address into register.
      - ex) `L 5 0x1000` → Register#5 stores the 4-bytes integer value loaded from the (virtual) memory address 0x1000

        Reg Memory address
    - ‘S’ type: Store 4-bytes integer value of register at the specifed memory address.
      - ex) `S 3 0x1000` → The 4-bytes integer value of Register#3 is stored at the (virtual) memory address 0x1000

        Reg Memory address

Requirements

- Instructions
  - The simulator provides 8 pseudo registers which is declared as ‘`int register_set[MAX_REGISTERS]`’ in ‘`vmsim.h`’.
    - The default value for `MAX_REGISTERS` is 8.
    - Each register has 4-bytes since it is declared as integer.
  - In the simulator, each instruction is stored as string (char array) in the pseudo physical memory and its maximum string length is defined by ‘`INSTRUCTION_SIZE`’ in ‘`vmsim.h`’.
    - The default value for `INSTRUCTION_SIZE` is 32 bytes.

Requirements

- Instructions
  - The simulator uses a program counter maintained by each process to fetch instruction strings from memory.
    - Please refer to '`int pc`' in structure '`Process`'.
    - In the simulator, the first instruction of a process is always located at 0x00000000 (virtual memory), so '`pc`' is initially set to 0x00000000.
    - PC must increment by INSTRUCTION_SIZE to point to the next instruction.
  - Instruction operations are already implemented in '`vmsim_op.c`', so there is no need to modify '`vmsim_op.c`'
    - `void op_move(Process *process, char *instruction);`
    - `void op_add(Process *process, char *instruction);`
    - `void op_load(Process *process, char *instruction);`
    - `void op_store(Process *process, char *instruction);`

Requirements

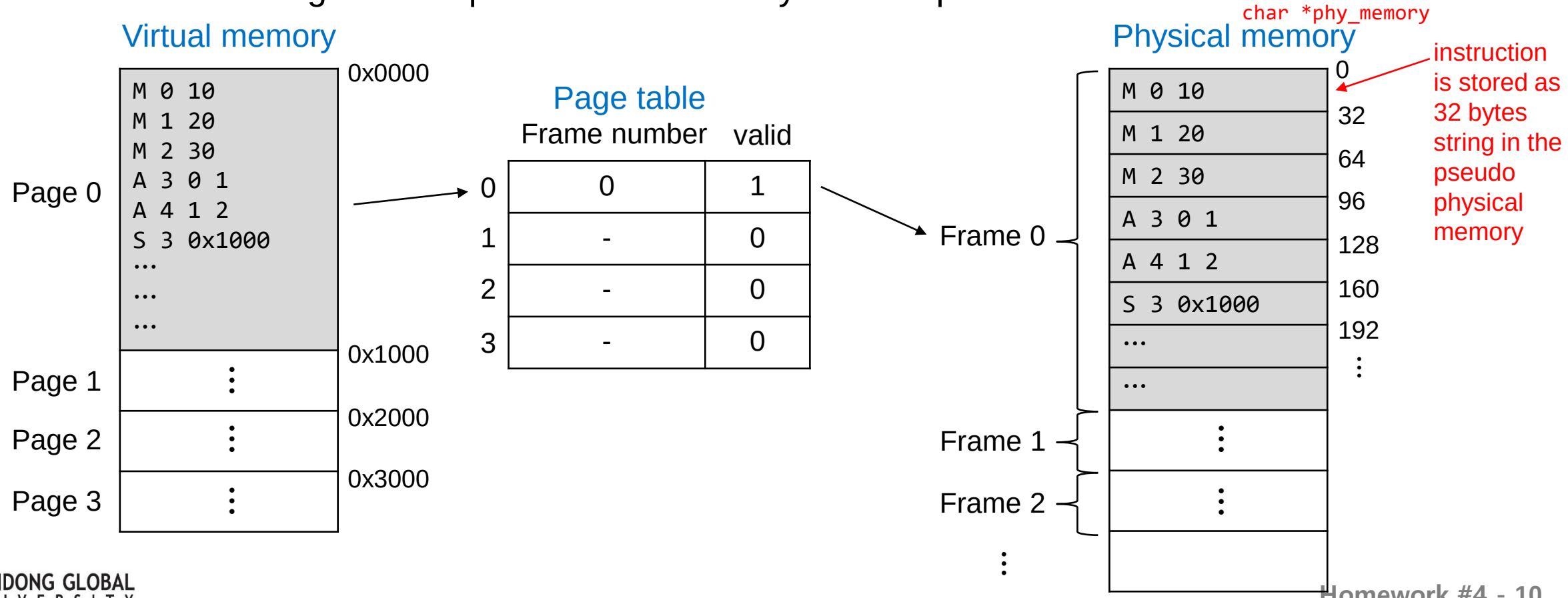
- Context switching
  - The simulator works with a pseudo clock.
  - We assume that the simulator executes any instruction within a single clock cycle.
  - The simulator performs context switching at every clock to simulate multi-process environment.
  - Process scheduling follows a round-robin scheme across all clocks.
  - Upon a context switch, the resumed process restores the saved register set from the temporary register set. Additionally, a process that completes an instruction stores its current register set in a temporary register set.
    - Please refer to '`int execute(Process *process)`' in '`vmsim_main.c`' and '`int temp_reg_set[MAX_REGISTERS]`' in structure '`Process`'.

Requirements

- Pseudo physical memory
  - In the simulator, physical memory is defined as ‘`char *phy_memory`’ in ‘`vmsim.h`’.
  - The size of ‘`phy_memory`’ is defined as ‘`PHY_MEM_SIZE`’ in ‘`vmsim.h`’.
    - The default value for `PHY_MEM_SIZE` is 16MB.
    - The default value for `PAGE_SIZE` is 4KB (4096bytes). Thus, there are 4096 frames in the ‘`phy_memory`’.

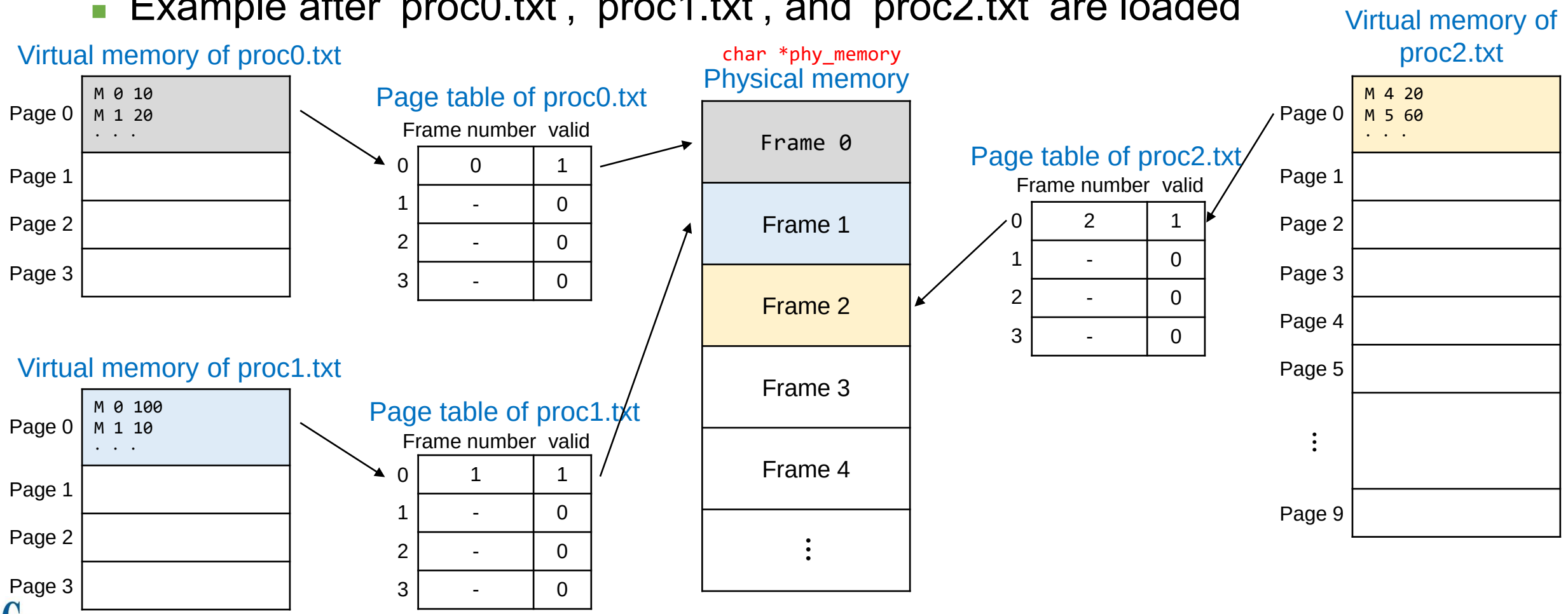
Requirements

- Demand paging
  - Example after ‘proc0.txt’ is loaded
    - Process image size of proc0.txt is 16384 bytes. It requires 4 frames.



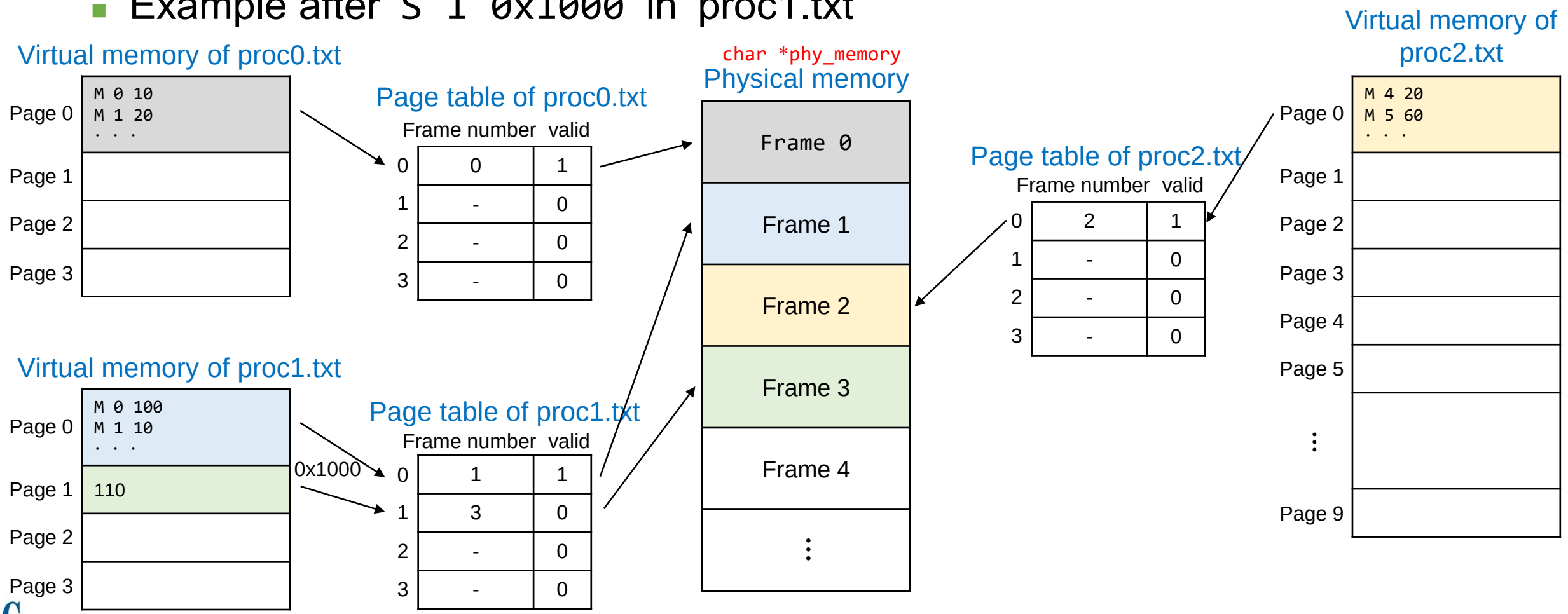
Requirements

- Demand paging
  - Example after ‘proc0.txt’, ‘proc1.txt’, and ‘proc2.txt’ are loaded



Requirements

- Demand paging
  - Example after '`S 1 0x1000`' in 'proc1.txt'



Requirements

- Demand paging
  - You don’t have to implement swap in/out with backing store.
  - When page fault occurs, just allocate the page.
  - When the process has no more instructions to execute, the process terminates, and at this time, all frames allocated to the process are reclaimed.
  - The simulator finds free frame to allocate new page from pseudo physical memory using first fit scheme.
  - You don’t have to implement page replacement.

Requirements

- Page Table
  - Page table entry is declared with structure ‘`PageTableEntry`’ in ‘`vmsim.h`’.
  - Each process maintains a page table defined as '`PageTableEntry *page_table`' in the structure '`Process`’.
  - For the predefined structures please refer to ‘`vmsim.h`’.

Requirements

- Expected results

```
yunmin@mcnl-PowerEdge-R740:~/lecture/os/hw04/sol$ ./vmsim proc0.txt
[Clock= 0][PID=0] Opcode=M, RegNo=0, Value=10
[Clock= 1][PID=0] Opcode=M, RegNo=1, Value=20
[Clock= 2][PID=0] Opcode=M, RegNo=2, Value=30
[Clock= 3][PID=0] Opcode=A, RegDest=3, RegSrc1=0, RegSrc2=1
[Clock= 4][PID=0] Opcode=A, RegDest=4, RegSrc1=1, RegSrc2=2
[Clock= 5][PID=0] Opcode=S, RegNo=3, Addr=0x1000
[Clock= 5][PID=0] Page fault at virtual address 0x1000 (page_number=1) --> Allocated frame_number=1
[Clock= 6][PID=0] Opcode=S, RegNo=4, Addr=0x2000
[Clock= 6][PID=0] Page fault at virtual address 0x2000 (page_number=2) --> Allocated frame_number=2
[Clock= 7][PID=0] Opcode=L, RegNo=5, Addr=0x1000
[Clock= 8][PID=0] Opcode=L, RegNo=6, Addr=0x2000
[Clock= 8][PID=0] [RegisterSet]: R[0]=10, R[1]=20, R[2]=30, R[3]=30, R[4]=50, R[5]=30, R[6]=50, R[7]=0
```

※ Frame number = 0 has instructions.

R[0]=10
R[1]=20
R[2]=30
R[3]=10+20=30
R[4]=20+30=50
Page fault because it is first access at 0x1000
R[3]=30 is stored at 0x1000 (frame number=1)
Page fault because it is first access at 0x2000
R[4]=50 is stored at 0x2000 (frame number=3)
R[5]=30 since 30 is loaded from 0x1000 into R[5]
R[6]=50 since 50 is loaded from 0x2000 into R[6]
Show all values of Register Set since process has no more instruction to execute

```
yunmin@mcnl-PowerEdge-R740:~/lecture/os/hw04/sol$ ./vmsim proc1.txt
[Clock= 0][PID=0] Opcode=M, RegNo=0, Value=100
[Clock= 1][PID=0] Opcode=M, RegNo=1, Value=10
[Clock= 2][PID=0] Opcode=A, RegDest=1, RegSrc1=0, RegSrc2=1
[Clock= 3][PID=0] Opcode=S, RegNo=1, Addr=0x1000
[Clock= 3][PID=0] Page fault at virtual address 0x1000 (page_number=1) --> Allocated frame_number=1
[Clock= 4][PID=0] Opcode=L, RegNo=2, Addr=0x2000
[Clock= 4][PID=0] Page fault at virtual address 0x2000 (page_number=2) --> Allocated frame_number=2
[Clock= 5][PID=0] Opcode=L, RegNo=3, Addr=0x1000
[Clock= 5][PID=0] [RegisterSet]: R[0]=100, R[1]=110, R[2]=0, R[3]=110, R[4]=0, R[5]=0, R[6]=0, R[7]=0
```

Show all values of Register Set since process has no more instruction to execute

Requirements

- Expected results

```
yunmin@mcnl-PowerEdge-R740:~/lecture/os/hw04/sol$ ./vmsim proc0.txt proc1.txt proc2.txt
[Clock= 0][PID=0] Opcode=M, RegNo=0, Value=10
[Clock= 1][PID=1] Opcode=M, RegNo=0, Value=100
[Clock= 2][PID=2] Opcode=M, RegNo=4, Value=20
[Clock= 3][PID=0] Opcode=M, RegNo=1, Value=20
[Clock= 4][PID=1] Opcode=M, RegNo=1, Value=10
[Clock= 5][PID=2] Opcode=M, RegNo=5, Value=60
[Clock= 6][PID=0] Opcode=M, RegNo=2, Value=30
[Clock= 7][PID=1] Opcode=A, RegDest=1, RegSrc1=0, RegSrc2=1
[Clock= 8][PID=2] Opcode=A, RegDest=1, RegSrc1=4, RegSrc2=5
[Clock= 9][PID=0] Opcode=A, RegDest=3, RegSrc1=0, RegSrc2=1
[Clock=10][PID=1] Opcode=S, RegNo=1, Addr=0x1000
[Clock=10][PID=1] Page fault at virtual address 0x1000 (page_number=1) --> Allocated frame_number=3
[Clock=11][PID=2] Opcode=S, RegNo=1, Addr=0x3000
[Clock=11][PID=2] Page fault at virtual address 0x3000 (page_number=3) --> Allocated frame_number=4
[Clock=12][PID=0] Opcode=A, RegDest=4, RegSrc1=1, RegSrc2=2
[Clock=13][PID=1] Opcode=L, RegNo=2, Addr=0x2000
[Clock=13][PID=1] Page fault at virtual address 0x2000 (page_number=2) --> Allocated frame_number=5
[Clock=14][PID=2] Opcode=S, RegNo=5, Addr=0x2000
[Clock=14][PID=2] Page fault at virtual address 0x2000 (page_number=2) --> Allocated frame_number=6
[Clock=15][PID=0] Opcode=S, RegNo=3, Addr=0x1000
[Clock=15][PID=0] Page fault at virtual address 0x1000 (page_number=1) --> Allocated frame_number=7
[Clock=16][PID=1] Opcode=L, RegNo=3, Addr=0x1000
[Clock=16][PID=1] [RegisterSet]: R[0]=100, R[1]=110, R[2]=0, R[3]=110, R[4]=0, R[5]=0, R[6]=0, R[7]=0
[Clock=17][PID=2] Opcode=L, RegNo=1, Addr=0x3000
[Clock=18][PID=0] Opcode=S, RegNo=4, Addr=0x2000
[Clock=18][PID=0] Page fault at virtual address 0x2000 (page_number=2) --> Allocated frame_number=1
[Clock=19][PID=2] Opcode=L, RegNo=5, Addr=0x2000
[Clock=20][PID=0] Opcode=L, RegNo=5, Addr=0x1000
[Clock=21][PID=2] Opcode=S, RegNo=4, Addr=0x4000
[Clock=21][PID=2] Page fault at virtual address 0x4000 (page_number=4) --> Allocated frame_number=3
[Clock=22][PID=0] Opcode=L, RegNo=6, Addr=0x2000
[Clock=22][PID=0] [RegisterSet]: R[0]=10, R[1]=20, R[2]=30, R[3]=30, R[4]=50, R[5]=30, R[6]=50, R[7]=0
[Clock=23][PID=2] Opcode=L, RegNo=6, Addr=0x4000
[Clock=23][PID=2] [RegisterSet]: R[0]=0, R[1]=80, R[2]=0, R[3]=0, R[4]=20, R[5]=60, R[6]=20, R[7]=0
```

→ Process#1 terminated

→ Frame 1 is allocated to Process#0 using first fit scheme.
Frame 1 was released by Process#1.

→ Frame 3 is allocated to Process#2 using first fit scheme.
Frame 1 was released by Process#1.

→ Process#0 terminated

→ Process#2 terminated

Requirements

- Please check ‘TODO’ in ‘`vmsim_main.c`’ and ‘`vmsim_main.h`’
- For unmentioned requirements, you can implement freely.
- Your program should be executed on Ubuntu.
- You can compile with makefile. (use ‘`make`’ or ‘`make clean`’)
- Write clean source code for readability and add proper comment in your source code
- Test your source codes with many cases for self verification
- Upload zip file on LMS by compressing all your source codes
  - File name: hw04_student id.zip (ex: hw04_20400022.zip)
- Due date: 11:59pm, 6/21 (Fri)