American University of Central Asia

Software Engineering Department

Operating Systems (COM 341)

Course Project

Part 2

Task #1: Basic Memory Management

The memory management model of the first part of the project is very simple. The kernel is keeping track of the next free memory slot and placing new executable images sequentially one after another. Because of that, the unloading handler (*isr_ 3*) cannot reclaim free space from the process to use it for newer processes. The first task of this part of the project is to create a basic physical memory manager with the use of a linked list data structure.

Objectives

Students are required to implement the following set of functions. Note that function names can be changed.

```
Memory::<pointer or index>_type AllocateMemory(Memory::<pointer or index>_type
number_of_memory_units);
```

The function allocates an unused block of physical memory (from *board.memory.ram*) of at least *number_ of_ memory_ units* (can allocate more) and returns a pointer/iterator to or an index of the first block on success (or an invalid value such as -1 or *nullptr* in case of error). The function should use a linked list to keep track of blocks of memory. A list can be stored separately or directly in the physical memory (*board.memory.ram*). The *first fit*, *next fit*, or *best-fit* approach described in the course book can be used. Students can adapt an algorithm presented in the book The C Programming Language (Second Edition) by Brian W. Kernighan and Dennis M. Ritchie (8.7 Example - A Storage Allocator).

```
void FreeMemory(Memory::<pointer or index>_type position);
```

The function frees a block of allocated memory at the specified *position* and merges it (if possible) with neighboring free blocks before and after it.

Task #2: Virtual Memory

SVMASM

Processes should be able to access the virtual memory via a set of specific instructions. Students should add or extend the current set of operation codes to read and write data to memory by modifying the SVMASM project.

For example, we can modify the *mov* instruction to support *memory-register* and *register-memory* data transfer.

Another possible option is to add separate instructions to load data from memory to a register and to store (place) it from register to a specified memory address.

Sample

Instructions

ld reg imm32

Loads data from a virtual address specified by an immediate value to a register

- *0x40: ld a imm32*
- *0x41: ld b imm32*
- *0x42: ld c imm32*
- *0x43: ld d imm32*

```
ld a 100 # Load the value from virtual memory at address 100 to the register ‘a’.
```

st reg imm32

Stores data at a specified virtual address (provided with the immediate value) from a register

- *0x50: st a imm32*
- *0x51: st b imm32*
- *0x52: st c imm32*
- *0x53: st d imm32*

```
st a 100 # Load the value in register ‘a’ to virtual memory at address 100.
```

VM

Students should use the new template of the *VM* project on the server; merge it with their own *kernel.cpp* from the previous part, and implement the following set of methods (detailed description can be found in the project sources):

MMU (memory.h, memory.cpp)

Implement the following (data types can be found in the header file):

A method that creates empty page tables for the kernel or new processes

```
static page_table_type* CreateEmptyPageTable();
```

A method that returns a page index and offset to a physical memory for a provided virtual memory address

```
page_index_offset_pair_type GetPageIndexAndOffsetForVirtualAddress(vmem_size_type address);
```

A method that searches for a new empty physical page

```
page_entry_type AcquireFrame();
```

A method that releases a physical page

```
void ReleaseFrame(page_entry_type page);
```

Algorithms for page allocators are discussed during the class and are based on those provided in section #4.3 in the course book.

CPU

Implement logic of the new instructions to access virtual memory from SVMASM. Use the MMU to translate virtual memory addresses (*GetPageIndexAndOffsetForVirtualAddress*). Generate *page faults* (call isr_4) for invalid pages.

Kernel

Add a new *page fault* handler for *isr_4*. The index of the faulty page should be placed by the CPU to register *a*. The handler should try to find a new empty physical page (via MMU) and set it to the faulty index in the page table.

```
machine.pic.isr_4 = [&]() {
     // ToDo
}
```

Modify scheduling algorithms to change the current page table during a context switch.

Additional tasks

Rename and modify the memory management routines in task #1 (AllocateMemory, FreeMemory) to work with the new virtual memory subsystem.

Notes

The name of the emulator is s*vm*. The name of the assembler is s*vmasm*.

The following compilers are allowed: *VC (>=10)*, *Clang (>=3.1)*, and *GCC (>=4.2)*

Homework

Reading

Operating Systems Design and Implementation, Third Edition by Andrew S. Tanenbaum, Section #4.1–4.6 (AUCA Library Call Number: QA76.76.O63 T35 2006)

The C Programming Language (Second Edition) by Brian W. Kernighan and Dennis M. Ritchie, 8.7 Example - A Storage Allocator (AUCA Library Call Number: QA76.73. C15 K47 1988)

Language Reference

The C++ Programming Language, Third Edition by Bjarne Stroustrup (AUCA Library Call Number: QA 76.73.C153 S773 2005)

C++ Language and Library Reference <http://en.cppreference.com>

Submission

Students have two weeks to finish the task. The kernel source tree should be packed and sent to toksaitov_d@auca.kg (pack the solution or a set of source files into a *zip* or *tar* archive).

It is recommended to use a version control system (VCS) such as Git or Mercurial to record you changes. You will get 0.1 extra points if you will provide your kernel source tree via the local repository or through a public web-based one (GitHub, Bitbucket, etc.).

Compose project description properly in a formal way. Describe the approach used for the solution.

The structure of the archive/repository should be the following:

```
+ <last_name>_<first_name>_course_project
|--+ svm-memory
|  |--- Project/Solution files and directories
|  |--- ...
|  |--+ assemblies
|     |--- change_registers_and_exit.vmasm
|     |--- read_write_to_virtual_memory.vmasm
|     |--- write_to_register_in_loop.vmasm
|     |--- ...
|--- CPU_Documentation.<extension>
```

Grading

To be eligible for a certain grade you need to implement the following set of elements. You final grade can be lower if your solution is not properly decomposed into a set of classes and methods or has an inconsistent code style (e.g., different naming conventions used for variables and methods, inconsistent code indentation).

None: F

Basic Memory Management

- Freeing memory: D
- Memory allocation: C

Virtual Memory Management

- CPU Memory Management Unit and virtual memory support for the kernel: B
- Virtual memory support for the Basic Memory Manager: A

Page Replacement

- +0.2 bonus points for any page replacement algorithm

Rules

Students are required to follow the rules of conduct of the Software Engineering Department and American University of Central Asia.

Teamwork is NOT encouraged. Equal blocks of code or similar structural pieces in separate works will be considered as academic dishonesty and all *authors* will get zero points for the task.