CSCI 3150 Introduction to Operating Systems

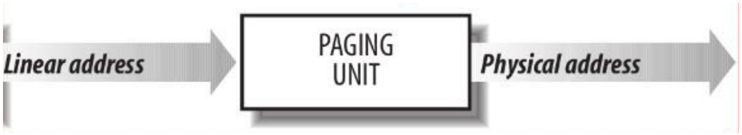
Bonus Assignment Four

Deadline: 23:59, April 21, 2024

Total Marks: 100

1. Overview

Implement a simple simulator of translating processes' linear address (virtual address) to physical address using paging technique.

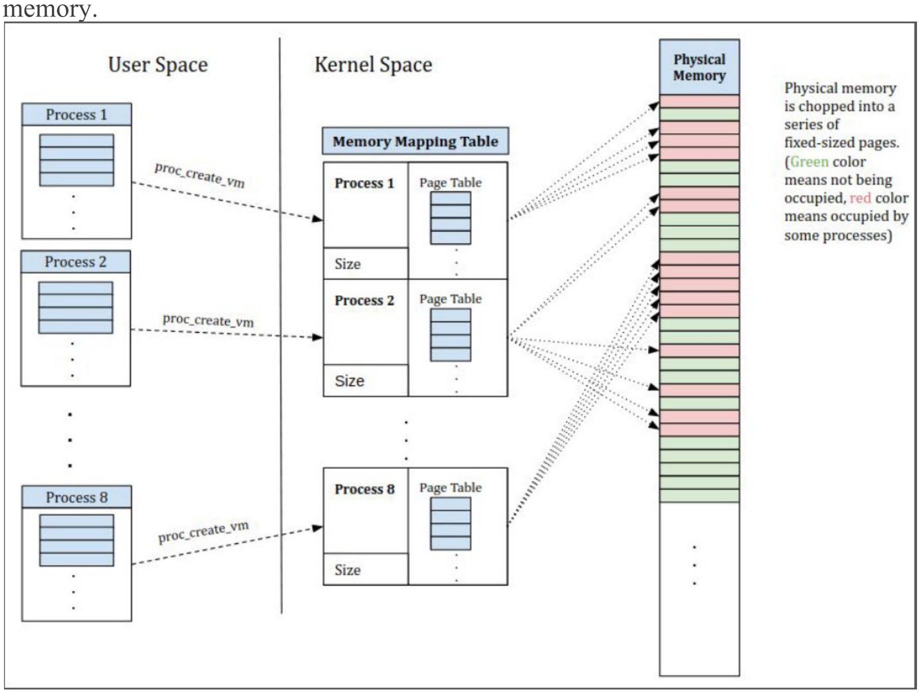


User Space

- User can create (proc_create_vm) process with specified virtual memory size through the API.
- User can read/write (vm_read/vm_write) the process' virtual memory through API.
- User can terminate (proc_exit_vm) process through the API.

Kernel Space

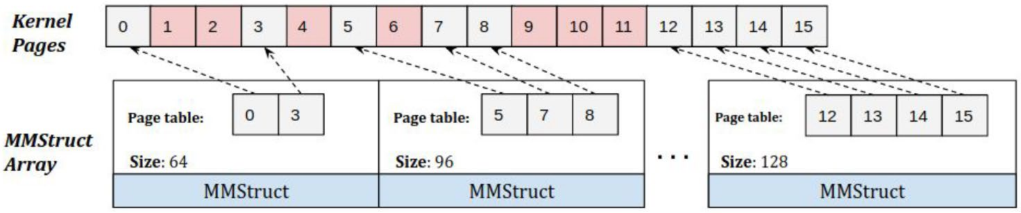
- The kernel simulator maintains a page table for each process and a shared physical memory.
- The kernel simulator handles the mappings from processes' virtual memory to physical memory.



Physical memory is chopped into a series of fixed-sized pages. (Green color means not being occupied, red color means occupied by some processes)

2.Data Structures

We define some data structures for kernel simulator. You can find them in "kernel.c".



PAGE_SIZE = 32
KERNEL_SPACE_SIZE = 512

(kernel.h) Kernel

```
struct Kernel {
  char* space;
  int allocated_pages;
  char* occupied_pages;
  char* running;
  struct MMStruct* mm;
};
```

- **space**: the physical memory.
- **allocated_pages**: Total number of allocated pages for all running processes.
- **occupied_pages**: bitmap (size is the number of kernel pages) to indicate the free pages, 0 -> free, 1 -> occupied.
- **running**: an array marking if the corresponding process is running.
- **mm**: an array of page tables.
    - Array of **MMStruct**

(kernel.h) MMStruct:

```
struct MMStruct {
  int size;
  struct PageTable* page_table;
};
```

- ***size*** determines the number of allocated pages.
- ***page table*** contains an array of PTEs.

3. Demo

You can find the demo program in “demo.c”. It contains a simple test case for you to debug.

4.Your tasks

Implement totally 4 functions (API) in the “kernel.c”.

(1) **int proc_create_vm(struct Kernel* kernel, int size)**

1.1. Check if a free process slot exists (check the running, the slot will be the pid returned).

1.2. Check if there's enough free space (check allocated_pages).

1.3. Allocate the space for page_table (the size of it depends on the pages you needed. e.g. if size=33 and PAGE_SIZE=32, then you need 2 pages) and update allocated_pages.

1.4. The mapping to kernel-managed memory is not built up, all the PFN should be set to -1 and present byte to 0 (PTE) and set the corresponding element in running to 1.

1.5. Return the pid if success, -1 if failure.

(2) **int vm_read(struct Kernel* kernel, int pid, char* addr, int size, char* buf)**

2.1. Check if the reading range is out-of-bounds.

2.2. If the pages in the range [addr, addr+size) of the user space of that process are not present, you should firstly map them to the free kernel-managed memory pages (**first fit policy: scan from the beginning**).

(3) int vm_write(struct Kernel* kernel, int pid, char* addr, int size, char* buf)

Similar with vm_read function.

(4) **int proc_exit_vm(struct Kernel* kernel, int pid)**

4.1. Reset the corresponding pages in occupied_pages to 0.

4.2. Release the page_table in the corresponding MMStruct and set to NULL. Return 0 when success, -1 when failure.

Submission:

You only need to submit "kernel.c" that can be compiled with the same "Makefile".

TA CHEN Xiangao is in charge of this assignment, if you have any questions about this assignment, you can enquiry with this email: **xachen23@cse.cuhk.edu.hk**