National and Kapodistrian University of Athens

Department of Informatics and Telecommunications

Operating Systems (K22) / Winter Semester 2022-2023

Assignment 2

Lazy Page Allocation Support in xv6

One of the techniques used in operating systems in conjunction with hardware that supports paging is lazy allocation of heap memory in the user-space. Applications in xv6 request heap memory from the kernel using the system call sbrk(). In the existing kernel, sbrk() allocates physical memory and maps it to the process's virtual memory space. However, there are programs that use sbrk() to request large memory sizes without ever using the whole or even the majority of it (e.g., implementations of large sparse arrays). To optimize this case, advanced operating system kernels allocate user-space memory lazily. In this scenario, sbrk() does not allocate physical memory but only records which addresses have been assigned. When the process attempts to use any page of memory for the first time, the CPU generates a page fault, which the kernel handles by allocating real physical memory, filling it with zeros, and mapping it to the virtual memory pages.

Step 1 – Print the Page Table

You will start by creating code that can help with debugging later. Implement a function that prints the contents of a page table. Define the function in kernel/vm.c as follows:

```
void vmprint(pagetable_t pagetable);
```

Add a call to vmprint() at the end of exec() in kernel/exec.c to print the page table for the first user process. The result should be as follows:

```
page table 0x0000000087f6e000
..0: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. ..0: pte 0x0000000021fda401 pa 0x0000000087f69000
.. .. ..0: pte 0x0000000021fdac1f pa 0x0000000087f6b000
.. .. ..1: pte 0x0000000021fda00f pa 0x0000000087f68000
.. .. ..2: pte 0x0000000021fd9c1f pa 0x0000000087f67000
..255: pte 0x0000000021fdb401 pa 0x0000000087f6d000
.. ..511: pte 0x0000000021fdb001 pa 0x0000000087f6c000
.. .. ..510: pte 0x0000000021fdd807 pa 0x0000000087f76000
.. .. ..511: pte 0x000000002000200b pa 0x0000000080008000
```

The first line prints the address of the vmprint() argument. Each PTE (Page Table Entry) line prints the PTE index in the page directory, the PTE, and the physical address (pa). The printing should indicate the level of the page directory: in the top-level entries, there is only one ".." string before the PTE, in the immediately lower level there is one more ".." string, and in the lowest level, there is another ".." string. Entries that have not been mapped should not be printed.

Step 2 – Remove Memory Allocation from sbrk()

The page allocation from the sbrk() system call should be removed, which means modifying the sys_sbrk() function in kernel/sysproc.c. The sbrk(n) system call increases the size of the process's memory by n bytes and then returns the start of the newly allocated region (i.e., the old size). The new sbrk(n) should simply increase the process's size (myproc()->sz) by n and return the old size. It should not allocate memory – pay attention to the growproc() call.

The result of this modification will be similar to the following:

```
init: starting sh

$ echo hi

usertrap(): unexpected scause 0x000000000000000f pid=3

sepc=0x0000000000001258 stval=0x0000000000004008

va=0x0000000000004000 pte=0x0000000000000000

panic: uvmunmap: not mapped
```

The message "usertrap(): ..." is produced by the user trap handler in kernel/trap.c. It encounters an exception it doesn't know how to handle. You will need to understand why this page fault error is occurring. The indication "stval=0x0000000000004008" points to the virtual address that caused the page fault.

<u>**Step 3 – Lazy Allocation**</u>

Modify the code in trap.c to respond to page faults from the user environment by mapping a newly allocated physical page to the address that caused the fault and then return to the user environment to let the process continue executing. You should add your code as an alternative case just before the printf() call that printed the "usertrap(): ..." message. You will also need to modify other parts of xv6 to make the "echo hi" command work.

<u>Instructions:</u>

- You can check if an error is a page fault by checking the value of r_scause() in the usertrap() and verifying if it is 13 or 15.
- The r_stval() function returns the value of the stval register of the RISC-V CPU, which contains the virtual address that caused the page fault.
- Use code from uvmalloc() in kernel/vm.c, which is called by sbrk() through growproc().
- Use PGROUNDDOWN(va) to round down the virtual address that caused the page fault to the beginning of the page.
- Calling uvmunmap() will cause a panic(). Modify it to handle the case where some pages are not mapped.
- Use the vmprint() function from Step 1 to print the contents of a page table.

- You may need to include the spinlock.h and proc.h files if the "incomplete type proc" error occurs.
- In the end, the command echo hi should work, encountering at least one page fault.
- Additionally, to handle all cases, you should address the following:
- Handle negative arguments in the call to sbrk().
- Terminate a process if it causes a page fault at a virtual memory address higher than any assigned via sbrk().
- Handle memory copying from parent to child in fork().
- Handle the case where a process passes a valid address from sbrk() to a system call like read() or write(), but the memory for that address has not been allocated yet.
- Properly handle memory exhaustion situations. If kalloc() fails in the page fault handler, terminate the process.
- Handle errors in invalid user stack pages. When you complete the task, the kernel should pass all tests in the lazytest and usertests programs, i.e.: [tests description].

```
$ lazytests

lazytests starting

running test lazy alloc

test lazy alloc: OK

running test lazy unmap...

test lazy unmap: OK

running test out of memory

test out of memory: OK

ALL TESTS PASSED

$ usertests

...

ALL TESTS PASSED$
```

You can also execute the command 'make grade', which will check for the correct implementation of vmprint().

Code

To use the code for the assignment, follow these steps:

```
$ git clone git://gallagher.di.uoa.gr/xv6-project-2022

Cloning into 'xv6-project-2022'...

...

$ cd xv6-project-2022
```

The repository xv6-project-2022 adds some additional functionality compared to the main one. To see the changes that have been made, you can issue the command:

```
$ git log
```

The files you will need for this assignment are distributed through the version control system git. You can find information about git in the git book or other public sources. Git allows you to maintain information about all the changes you have made to the code. For example, if you finish part of the assignment and want to locally record your changes, you can commit your changes using the command:

```
$ git commit -am 'my solution for cow implementation project'

Created commit 60d2135: my solution for project 2 exercise 1

1 file changed, 1 insertion(+), 0 deletions(-)

$
```