Lab 3: Memory Management

Printing page table entries() (10 marks)

- The first task is to implement a function that prints the contents of a page table.
- Define the function in **kernel/vm.c** having following prototype:

  void vmprint(pagetable_t).

- Insert a call to **vmprint** in **exec.c** to print the page table for the first user process; its output should be as below.

- Insert the function prototype in **defs.h**

- **Sample Output**

```
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
```

Note:

- The first line prints the address of the argument of **vmprint**.
- Each PTE line shows the PTE index in its page directory, the **pte**, the physical address for the PTE.
- The output should also indicate the level of the page directory:
  the top-level entries are preceeded by ”..”,
  the next level down with another ”..”, and so on.
  You should not print entries that are not mapped.
- You can use the idea from **freewalk** function in **vm.c**

Eliminate allocation from sbrk() (10 marks)

- Your next task is to delete page allocation from the **sbrk(n)** system call implementation, which is the function **sys_sbrk()** in **sysproc.c**.
- The **sbrk(n)** system call grows the process's memory size by **n** bytes, and then returns the start of the newly allocated region (i.e., the old size).
- Your new **sbrk(n)** should just increment the process's size (**myproc()->sz**) by **n** and return the old size (note. No memory is allocated)

- Remember to delete the memory allocation but you still need to increase the process size
- **Sample Output**
  Make this modification, boot xv6, and type echohi to the shell. You should see something like this:

  ```
  init: starting sh
  $ echo hi
  usertrap(): unexpected scause 0x000000000000000f pid=3
  sepc=0x0000000000001258 stval=0x0000000000004008
  va=0x0000000000004000 pte=0x0000000000000000
  panic: uvmunmap: not mapped
  ```

Submission

The assignment should be done individually.

The following artifacts need to submitted:

(a) Video: demonstrating that it works on your system. At the start of the video display the ifconfig of your system, by running the command $ifconfig and then, demonstrate the program functionality.

(b) Code files: Run make clean and zip the entire xv6-riscv repo, containing your solution.

(c) Report: Explain the steps you followed to solve each problem.

References

RISCV Privileged Instruction Set Manual
https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf

•

Lazy allocation (10 marks)

- Modify the code in **trap.c** to respond to a page fault from user space by mapping a newly-allocated page of physical memory at the faulting address, and then returning back to user space to let the process continue executing.

- You should add your code just before the **printf** call that produced the ”**usertrap() : ...**” message so that when you **call echo hi** it should be able to allocate new page.

- Sample Output $ echo hi

  statement should work correctly.

Note: you will find some additional problems that have to be solved to make it work correctly

- You can check whether a fault is a page fault by seeing if **r_scause()** is 13 or 15 in **usertrap()**.
- Look at the arguments to the **printf()** in **usertrap()** that reports the page fault, in order to see how to find the virtual address that caused the page fault.
- Steal code from **uvmalloc()** in **vm.c**, which is what **sbrk()**calls(via **growproc()**). You’ll need to call **kalloc**() and **mappages()**.
- Use **PGROUNDDOWN(va)** to round the faulting virtual address down to a page boundary.
- **uvmunmap()** will panic; modify it to not panic if some pages aren’t mapped.