

Lab: xv6 lazy page allocation

One of the many neat tricks an O/S can play with page table hardware is lazy allocation of user-space heap memory. Xv6 applications ask the kernel for heap memory using the `sbrk()` system call. In the kernel we've given you, `sbrk()` allocates physical memory and maps it into the process's virtual address space. It can take a long time for a kernel to allocate and map memory for a large request. Consider, for example, that a gigabyte consists of 262,144 4096-byte pages; that's a huge number of allocations even if each is individually cheap. In addition, some programs allocate more memory than they actually use (e.g., to implement sparse arrays), or allocate memory well in advance of use. To allow `sbrk()` to complete more quickly in these cases, sophisticated kernels allocate user memory lazily. That is, `sbrk()` doesn't allocate physical memory, but just remembers which user addresses are allocated and marks those addresses as invalid in the user page table. When the process first tries to use any given page of lazily-allocated memory, the CPU generates a page fault, which the kernel handles by allocating physical memory, zeroing it, and mapping it. You'll add this lazy allocation feature to xv6 in this lab.

Before you start coding, read Chapter 4 (in particular 4.6) of the [xv6 book](#), and related files you are likely to modify:

- `kernel/trap.c`
- `kernel/vm.c`
- `kernel/sysproc.c`

To start the lab, switch to the lazy branch:

```
$ git fetch
$ git checkout lazy
$ make clean
```

Eliminate allocation from `sbrk()` (**easy**)

Your first 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. It should not allocate memory -- so you should delete the call to `growproc()` (but you still need to increase the process's size!).

Try to guess what the result of this modification will be: what will break?

Make this modification, boot xv6, and type `echo hi` to the shell. You should see something like this:

```
init: starting sh
$ echo hi
usertrap(): unexpected scause 0x000000000000000f pid=3
          sepc=0x00000000000001258 stval=0x0000000000004008
va=0x0000000000004000 pte=0x0000000000000000
panic: uvmunmap: not mapped
```

The "usertrap(): ..." message is from the user trap handler in `trap.c`; it has caught an exception that it does not know how to handle. Make sure you understand why this page fault occurs. The "stval=0x0..04008" indicates that the virtual address that caused the page fault is 0x4008.

Lazy allocation ([moderate](#))

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. Modify whatever other xv6 kernel code you need to in order to get `echo hi` to work.

Here are some hints:

- You can check whether a fault is a page fault by seeing if `r_scause()` is 13 or 15 in `usertrap()`.
- `r_stval()` returns the RISC-V `stval` register, which contains the virtual address that caused the page fault.
- Steal code from `uvmmalloc()` 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.
- `uvmmunmap()` will panic; modify it to not panic if some pages aren't mapped.
- If the kernel crashes, look up `sepc` in `kernel/kernel.asm`
- Use your `vmprint` function from `pgtbl` lab to print the content of a page table.
- If you see the error "incomplete type proc", include "spinlock.h" then "proc.h".

If all goes well, your lazy allocation code should result in `echo hi` working. You should get at least one page fault (and thus lazy allocation), and perhaps two.

Lazytests and Usertests ([moderate](#))

We've supplied you with `lazytests`, an xv6 user program that tests some specific situations that may stress your lazy memory allocator. Modify your kernel code so that all of both `lazytests` and `usertests` pass.

- Handle negative `sbrk()` arguments.
- Kill a process if it page-faults on a virtual memory address higher than any allocated with `sbrk()`.
- Handle the parent-to-child memory copy in `fork()` correctly.
- Handle the case in which a process passes a valid address from `sbrk()` to a system call such as `read` or `write`, but the memory for that address has not yet been allocated.
- Handle out-of-memory correctly: if `kalloc()` fails in the page fault handler, kill the current process.
- Handle faults on the invalid page below the user stack.

Your solution is acceptable if your kernel passes `lazytests` and `usertests`:

```
$ lazytests
lazytests starting
running test lazy alloc
test lazy alloc: OK
running test lazy unmap...
usertrap(): ...
test lazy unmap: OK
running test out of memory
usertrap(): ...
test out of memory: OK
ALL TESTS PASSED
$ usertests
...
ALL TESTS PASSED
$
```

Submit the lab

This completes the lab. Make sure you pass all of the make grade tests. If this lab had questions, don't forget to write up your answers to the questions in `answers-lab-name.txt`. Commit your changes (including adding `answers-lab-name.txt`) and type `make handin` in the lab directory to hand in your lab.

Time spent

Create a new file, `time.txt`, and put in it a single integer, the number of hours you spent on the lab. Don't forget to `git add` and `git commit` the file.

Submit

You will turn in your assignments using the google classroom.

After committing your final changes to the lab, type `make tarball` to create archive which you'll submit to classroom.

```
$ git commit -am "ready to submit my lab"
[util c2e3c8b] ready to submit my lab
 2 files changed, 18 insertions(+), 2 deletions(-)

$ make tarball
git archive --format=tar HEAD | gzip > lab-lazy-handin.tar.gz
$
```

If you run `make tarball` and you have either uncommitted changes or untracked files, you will see output similar to the following:

```
M hello.c
?? bar.c
?? foo.pyc
Untracked files will not be handed in. Continue? [y/N]
```

Inspect the above lines and make sure all files that your lab solution needs are tracked i.e. not listed in a line that begins with `??`. You can cause git to track a new file that you create using `git add filename`.

If `make tarball` does not work properly, try fixing the problem with the Git commands.

- Please run ``make grade`` to ensure that your code passes all of the tests
- Commit any modified source code before running ``make tarball``

Optional challenge exercises

- Make lazy page allocation work with your simple copyin from the previous lab.