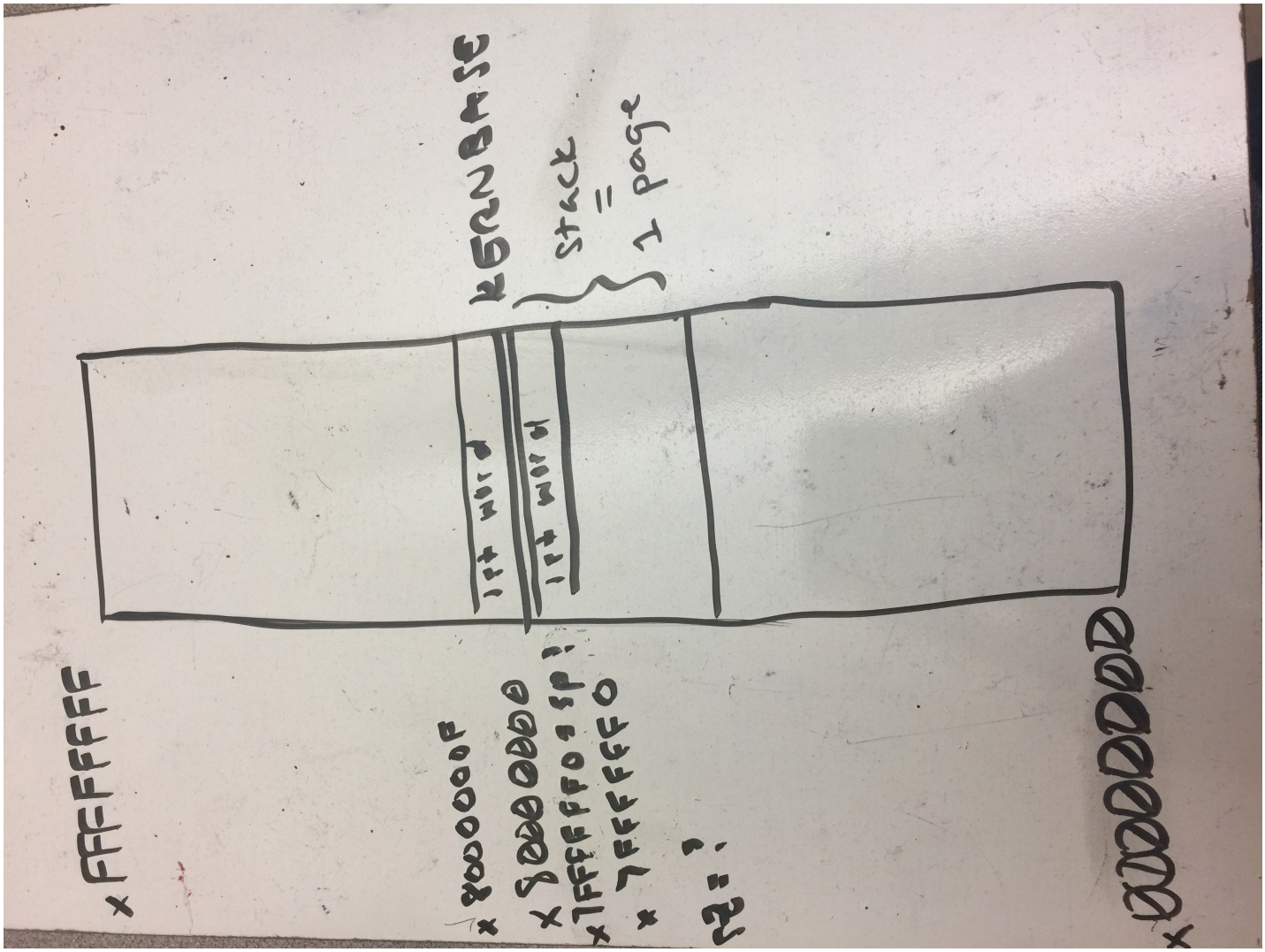
! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

question @66 77 views

Getting Started with Lab 3

So I'm trying to understand the beginning of lab 3.
For allocuvm, it says to change the virtual address of the first page we are mapping – this needs to change to point to the top page of the user part of memory (right under `KERNBASE`), so would that be x7FFFFFFF (`KERNBASE` -1)?
Then, we have to change the virtual address of the last page we are mapping. "For us, we are creating a stack with only a single page, so this can another address in the same page, slightly bigger than the first address." By bigger, would that mean `KERNBASE` -1 - x, such that x < PGSIZE?
Now the part I mostly don't understand is why we have to change sp. Don't we change sz (from allocuvm) such that we wouldn't need to change sp? I understand allocuvm allocates pages, but doesn't it also get assigned to sz?
The lab says we have to change sp to the address of the top word in the stack page. "Note that `KERNBASE` is the first word in the kernel address space, so this is the word right under that." Would that address be x7FFFFFFF? Or x7FFFFFF0? The note about how it's a word confused me.



lab3

~ An instructor (Bashar Romanous) thinks this is a good question ~

Updated 9 months ago by Amanda Jade Cao (Anon. Helix to classmates)

the instructors' answer, *where instructors collectively construct a single answer*

Very good question!

consider the code in exec.c:

```
// Load program into memory.
 sz = 0;
 for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){
   if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
     goto bad;
```

```
    if(ph.type != ELF_PROG_LOAD)
      continue;
    if(ph.memsz < ph.filesz)
      goto bad;
    if(ph.vaddr + ph.memsz < ph.vaddr)
      goto bad;
    if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
      goto bad;
    if(ph.vaddr % PGSIZE != 0)
      goto bad;
    if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)
      goto bad;
  }
  iunlockput(ip);
  end_op();
  ip = 0;
```

by the end of this code, this is what your process memory looks like:

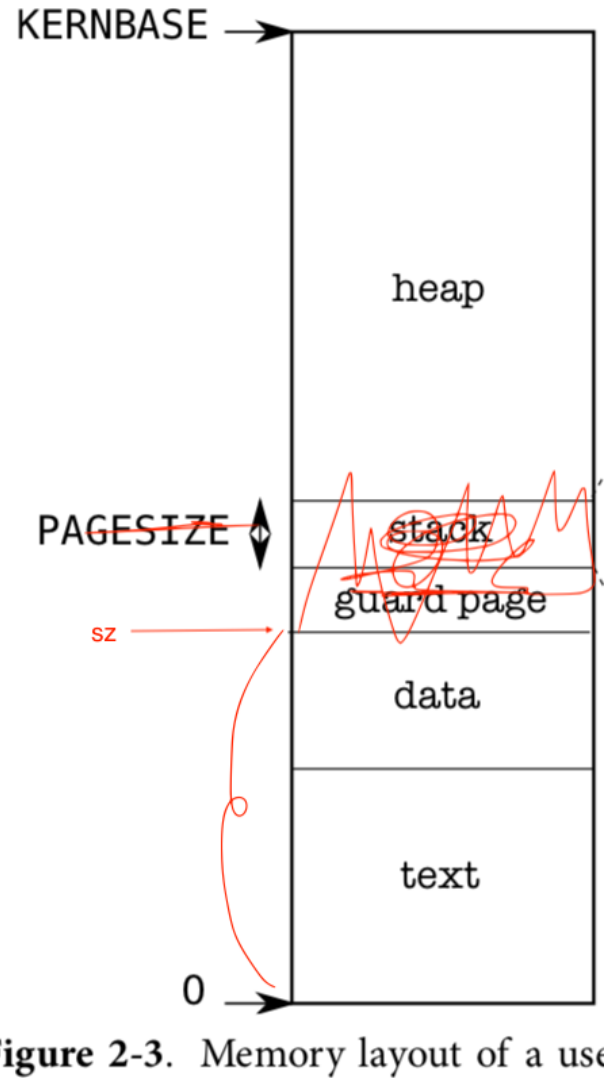


Figure 2-3. Memory layout of a use

where sz refers to the top of the code (text and const data) section.
and After these lines in the original code:

```
  // Allocate two pages at the next page boundary.
  // Make the first inaccessible.  Use the second as the user stack.
  sz = PGROUNDUP(sz);
  if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
    goto bad;
  clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
  sp = sz;
```

two pages are allocated on top of previous sz and the new size of prices memory is assigned to sz again.
sz now is both the size of the process memory (text and const data + one guard page and 1 stack page) and the address of the top of the stack in the same time.

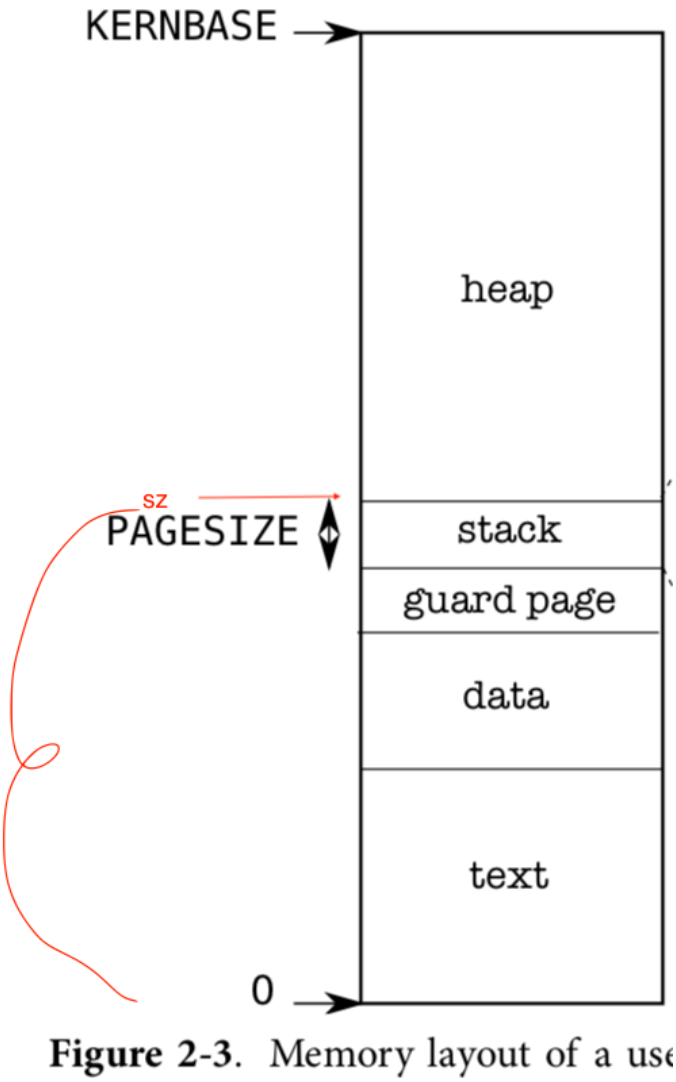
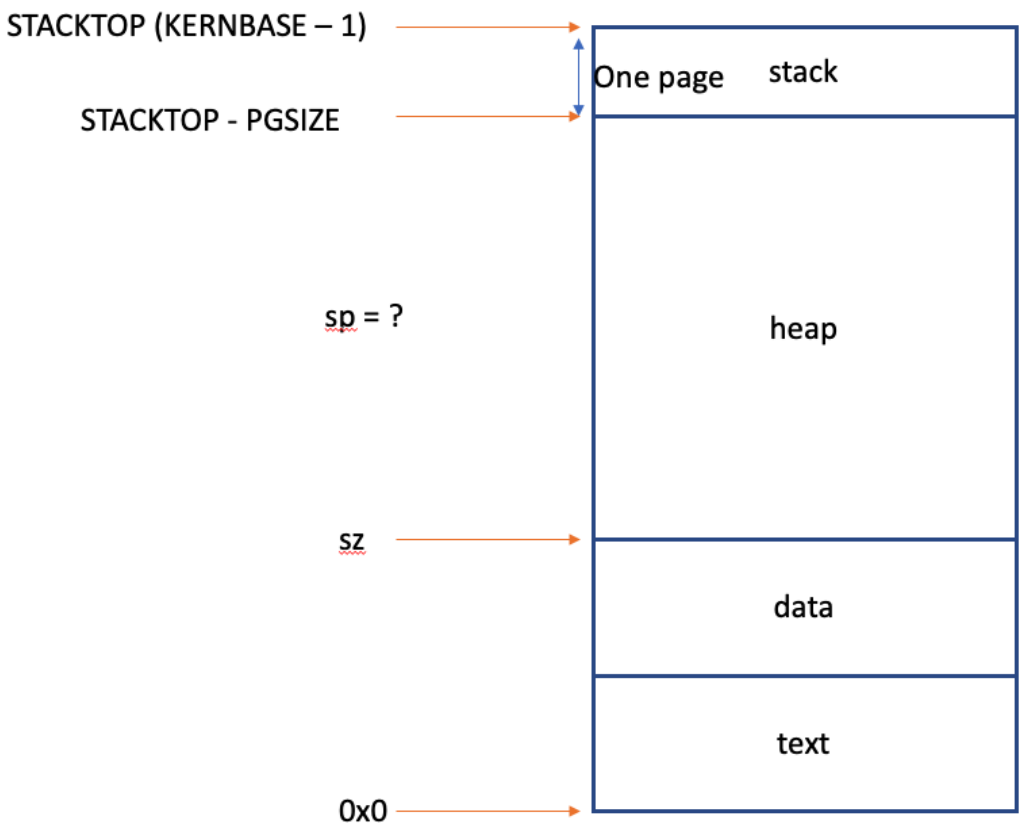


Figure 2-3. Memory layout of a use

the variable sz is serving both as an address to tell you where the top of the stack is (hence sp =sz) and as a number to tell you the entire memory size of the process.

since we are making the stack start somewhere below kernbase such as STACKTOP = KERNBASE -1 and putting the heap between the stack and the (text + const data) section:



why should sp be set to sz anymore? you should set sp to point to the top of the new stack and create a new field in proc to keep track of the number of new pages you are allocating and leave sz to point to the size of the (text + data) section of the code (till you finish part 2 the stack is only page page since trap 14 for page fault handling is not complete, that is part 2 of lab3 which Is to write that part of the code where your stack can allocate more pages as needed, see lab3 notes on ilearn regarding lab3.c)

note: it helps to read the prototype of allocuvm to understand what it returns and takes as inputs.

```
// Allocate page tables and physical memory to grow process from oldsz to
// newsz, which need not be page aligned.  Returns new size or 0 on error.
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
```

Note: you don't need the guard page anymore, just allocate one page for stack

you will be changing few lines in exec.c to allocate one page of stack in the new location and initiate the new stack counter. modify copyuvm to copy the new stack separately since it is no longer on continues space with (text + data) and finally reconsider the safety checks for the 3 helper functions under syscall.c

Updated 9 months ago by Bashar Romanous

followup discussions *for lingering questions and comments*

Resolved Unresolved

Anonymous Beaker 9 months ago
o_o
good comment | 0

Anonymous Comp 9 months ago o_o
good comment | 0