Lab 3 Report

Steven Nguyen - snguy057

Brittney Mun - bmun001

Changes

proc.h

- Line 52: Created uint pages

```
52    uint pages;                   // CS153 Lab 3: Number of pages
```

memlayout.h

Defined top of stack address

- Line 18: Defined memory address for top of stack.

```
17  // Lab 3: Top of Stack Address
18  #define TOPSTACK (KERNBASE-4)       // The first address is one word
19                                      // under the kenal base
```

proc.c

Copied pages to child processes

- Line 204: Copied pages to child process as well.

```
203    // CS153 Lab 3: copy number of pages
204    np->pages = curproc->pages;
```

syscall.c

Implemented Lab 3 by replacing cur-proc->sz with TOPSTACK throughout the file.

- Line 24, 42, 45, 74: Use TOPSTACK instead of curproc->sz.

```c
int
fetchint(uint addr, int *ip)
{
  // CS153 Lab 3: Change checks to top of the stack rather than sz
  //     Removes need to myproc as we use the top of stack
  // struct proc *curproc = myproc();

  if(addr >= TOPSTACK || addr+4 > TOPSTACK)
    return -1;
  *ip = *(int*)(addr);
  return 0;
}

int
fetchstr(uint addr, char **pp)
{
  char *s, *ep;

  // CS153 Lab 3: Change checks to top of the stack rather than sz
  //     Removes need to myproc as we use the top of stack
  // struct proc *curproc = myproc();

  if(addr >= TOPSTACK)
    return -1;
  *pp = (char*)addr;
  ep = (char*)TOPSTACK;
  for(s = *pp; s < ep; s++){
    if(*s == 0)
      return s - *pp;
  }
  return -1;
}
```

```
  // CS153 Lab 3: Change checks to top of the stack rather than sz
  //     Removes need to myproc as we use the top of stack
  // struct proc *curproc = myproc();

  if(argint(n, &i) < 0)
    return -1;
  if(size < 0 || (uint)i >= TOPSTACK || (uint)i+size > TOPSTACK)
    return -1;
  *pp = (char*)i;
  return 0;
}
```

vm.c

Copies over information into virtual memory.

- Lines 340 - 352: Loop in order to copy information onto the new stack.

```
339   // CS153 Lab 3: Loop to copy the new stack
340   for(i = PGROUNDUP(TOPSTACK - (PGSIZE * myproc()->pages)); i < TOPSTACK; i += PGSIZE) {
341     if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
342       panic("copyuvm: pte should exist");
343     if(!(*pte & PTE_P))
344       panic("copyuvm: page not present");
345     pa = PTE_ADDR(*pte);
346     flags = PTE_FLAGS(*pte);
347     if((mem = kalloc()) == 0)
348       goto bad;
349     memmove(mem, (char*)P2V(pa), PGSIZE);
350     if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0)
351       goto bad;
352   }
```

exec.c

Allocates all pages starting from the top of the stack. Replaces all sz with TOPSTACK.

- Line 65, 67: Replace sz with TOPSTACK.
- Line 102: Set number of pages to 1.

```
// CS153 Lab 3: Allocate page at the top of the stack
sz = PGROUNDUP(sz);
if((sp = allocuvm(pgdir, TOPSTACK - PGSIZE, TOPSTACK)) == 0)
  goto bad;
sp = TOPSTACK;
 // CS153 Lab 3: set number of pages to 1
 curproc->pages = 1;
 cprintf("Initial number of pages by the process: %d\n", curproc->pages); //cs153 - Lab3
```

trap.c

Implemeneted page faults

- Lines 81 - 93: Implemented page faults.

```
81  // CS153 Lab 3: Page Fault
82  case T_PGFLT:
83    // Check if the address in rcr2 is in the next page
84    if ((rcr2() < PGROUNDUP(TOPSTACK - myproc()->pages * PGSIZE)) && (rcr2() > PGROUNDUP(TOPSTACK - (myproc()->pages+1)*PGSIZE))) {
85      myproc()->pages++;
86      if (allocuvm(myproc()->pgdir, (TOPSTACK - myproc()->pages * PGSIZE), (TOPSTACK - (myproc()->pages - 1) * PGSIZE)) == 0) {
87        cprintf("case T_PGFLT from trap.c: allocuvm failed. Number of current allocated pages: %d\n", myproc()->pages);
88        exit();
89      }
90       cprintf("case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: %d\n", myproc()->pages);
91    }
92    break;
```

Extra Credit

We were unable to implement a stack that grows into the heap because this would cause the stack to overlap the heap. As a result, the stack would overwrite information contained within the heap. The heap is allocated within vm.c and shown below. Within the I/O space is the heap. By growing the stack towards the heap, we effectively reach outside of the size limitations of the stack and cause an overflow, which would break the kernel and trigger an overflow error.

```c
// This table defines the kernel's mappings, which are present in
// every process's page table.
static struct kmap {
  void *virt;
  uint phys_start;
  uint phys_end;
  int perm;
} kmap[] = {
 { (void*)KERNBASE, 0,             EXTMEM,    PTE_W}, // I/O space
 { (void*)KERNLINK, V2P(KERNLINK), V2P(data), 0},     // kern text+rodata
 { (void*)data,     V2P(data),     PHYSTOP,   PTE_W}, // kern data+memory
 { (void*)DEVSPACE, DEVSPACE,      0,         PTE_W}, // more devices
};
```