Lab 3 Report

Steven Nguyen - snguy057

Brittney Mun - bmun001

Changes

proc.h

Removed status, runtime, start time, and priority variables

- Line 52: Created uint pages, deleted 4 variables.

memlayout.h

Defined top of stack address.

- Line 12: Removed char pointer declaration.
- Line 18: Defined memory address for top of stack.

proc.c

Removed lab 2 content due to programming issues.

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.
- Line 115, 116: Syscall initialization.
- Line 140, 141: Syscall call.

```
24    if(addr >= TOPSTACK || addr+4 > TOPSTACK)
42    if(addr >= TOPSTACK)
43      return -1;
44    *pp = (char*)addr;
45    ep = (char*)TOPSTACK;
74    if(size < 0 || (uint)i >= TOPSTACK || (uint)i+size > TOPSTACK)
115  extern int sys_shm_open(void);
116  extern int sys_shm_close(void);
140  [SYS_shm_open] sys_shm_open,
141  [SYS_shm_close] sys_shm_close
```

vm.c

Copies over information into virtual memory.

- Lines 340 - 350: 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.

```
63      // CS153 Lab 3: Allocate page at the top of the stack
64      sz = PGROUNDUP(sz);
65      if((sp = allocuvm(pgdir, TOPSTACK - PGSIZE, TOPSTACK)) == 0)
66        goto bad;
67      sp = TOPSTACK;
101     // CS153 Lab 3: set number of pages to 1
102     curproc->pages = 1;
103     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.

```
103  // This table defines the kernel's mappings, which are present in
104  // every process's page table.
105  static struct kmap {
106    void *virt;
107    uint phys_start;
108    uint phys_end;
109    int perm;
110  } kmap[] = {
111   { (void*)KERNBASE, 0,             EXTMEM,    PTE_W}, // I/O space
112   { (void*)KERNLINK, V2P(KERNLINK), V2P(data), 0},     // kern text+rodata
113   { (void*)data,     V2P(data),     PHYSTOP,   PTE_W}, // kern data+memory
114   { (void*)DEVSPACE, DEVSPACE,      0,         PTE_W}, // more devices
115  };
```