Mark Gameng

CS 450 – Duan Yue

Lab 4 – Memory Management

In exec.c we change the allocation to the one we want. Kernbase at top of stack with 2 pages. This makes the stack pointer to be Kernbase – 1. Thus, the user stack and page guard goes up to Kerbase – 2 * pg size. There's also a need to track the number of user stack pages so I added a new property called pages.

```
65  // lab 4
66  sz = PGROUNDUP(sz);
67  // pg 12 of pp
68  sp = KERNBASE - 1;
69  if((allocuvm(pgdir, PGROUNDDOWN(sp), sp)) == 0) // sz no longer here, pg 5 -> pg 12
70    goto bad;
71  clearpteu(pgdir, (char*)(sp - 2*PGSIZE)); // page guard, need 2?
72  //sp = sz;
```

```
 99  // Commit to the user image.
100  oldpgdir = curproc->pgdir;
101  curproc->pgdir = pgdir;
102  curproc->sz = sz;
103  curproc->tf->eip = elf.entry;  // main
104  curproc->tf->esp = sp;
105  curproc -> pages = 1; // keep track of # user stack pages
```

So, in syscall.c, changes need to be made to fetchint, fetchstr, and argptr as there is a new boundary which is top of user stack. So need to change curproc -> sz.

```
17  int
18  fetchint(uint addr, int *ip)
19  {
20    struct proc *curproc = myproc();
21    // new boundary is top of user stack which would be kernbase or kernbase - 1??
22    if(addr >= KERNBASE - 1 || addr+4 > KERNBASE - 1)
23      return -1;
24    *ip = *(int*)(addr);
25    return 0;
26  }
```

```
31  int
32  fetchstr(uint addr, char **pp)
33  {
34    char *s, *ep;
35    struct proc *curproc = myproc();
36
37    if(addr >= KERNBASE - 1)
38      return -1;
39    *pp = (char*)addr;
40    ep = (char*)(KERNBASE - 1);
41    for(s = *pp; s < ep; s++){
42      if(*s == 0)
43        return s - *pp;
44    }
45    return -1;
46  }
```

```
58  int
59  argptr(int n, char **pp, int size)
60  {
61    int i;
62    struct proc *curproc = myproc();
63
64    if(argint(n, &i) < 0)
65      return -1;
66    if(size < 0 || (uint)i >= KERNBASE - 1 || (uint)i+size > KERNBASE - 1)
67      return -1;
68    *pp = (char*)i;
69    return 0;
70  }
```

In vm.c, need to update copyuvm. As previously the virtual memory is from 0 to curproc -> sz. With the changes we have made, that memory now contains the code and the heap. So needs to update that with the correct one.

```
326   struct proc *curproc = myproc();
327   for(i = PGROUNDDOWN(KERNBASE -1); i > PGROUNDDOWN(KERNBASE - 1) - ((curproc->pages) * PGSIZE); i -= PGSIZE){
328   // instead of 0 to curproc -> sz, its kernbase to 0 followed by page guard
329     if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
330       panic("copyuvm: pte should exist");
331     if(!(*pte & PTE_P))
332       panic("copyuvm: page not present");
333     pa = PTE_ADDR(*pte);
334     flags = PTE_FLAGS(*pte);
335     if((mem = kalloc()) == 0)
336       goto bad;
337     memmove(mem, (char*)P2V(pa), PGSIZE);
338     if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0) {
339       kfree(mem);
340       goto bad;
341     }
```

In trap.c need to add a case for page fault, in which it allocates a new page only if the bad address is from the page right below the stack

```
case T_PGFLT: {
  // check what address caused it and allocate new page ONLY if bad address is from the page right below stack
  uint addr_ = rcr2();
  struct proc *curproc = myproc();
  if(allocuvm(curproc -> pgdir, PGROUNDDOWN(addr_), PGROUNDDOWN(addr_) + PGSIZE) == 0){
    exit();
  }
  //clearpteu(curproc -> pgdir, (char*)((KERNBASE - 1 - curproc -> pages * PGSIZE) - PGSIZE));
  curproc -> pages = curproc -> pages + 1;
  cprintf("increased stack size\n");

  break; } // need brackets - stackoverflow.com/questions/27006986/what-is-wrong-with-my-switch-statement
```

Testing

For testing, I just used the test program given by the TA on the slides.

```
int main(int argc, char *argv[]){
    int v = argc;
    printf(1, "&p\n", &v);
    exit();
}
```

```
$ test
7FFFFFCC
$ test 1 2
7FFFFFBC
$ test 1 1 1 1 1 1 1
7FFFFF8C
$
```

```
// Prevent this function from being optimized, which might give it closed form
#pragma GCC push_options
#pragma GCC optimize ("O0")
static int
recurse(int n)
{
  if(n == 0)
    return 0;
  return n + recurse(n - 1);
}
#pragma GCC pop_options

int
main(int argc, char *argv[])
{
  int n, m;

  if(argc != 2){
    printf(1, "Usage: %s levels\n", argv[0]);
    exit();
  }

  n = atoi(argv[1]);
  printf(1, "Lab 3: Recursing %d levels\n", n);
  m = recurse(n);
  printf(1, "Lab 3: Yielded a value of %d\n", m);
  exit();
}
```

```
$ test1 555
Lab 3: Recursing 555 levels
increased stack size
increased stack size
increased stack size
increased stack size
Lab 3: Yielded a value of 154290
$
```