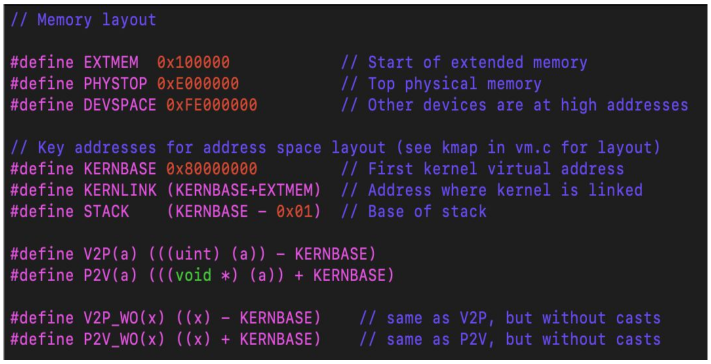
Chris Hung Huynh, SID: 862024281

Edwin Leon, SID: 862132870

Lab 3 Report

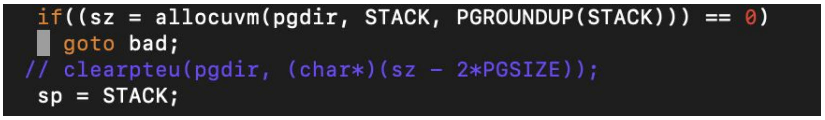
In this lab, we are asked to change the memory layout by placing the stack right under the kernel base. In addition, we had to grow the stack backwards by adding a page fault case in the trap.c file. We started the lab by moving the stack, so that it would be placed right after the kernel base. The steps we took to change the layout by moving the stack were:

1. First we added "STACK" in the memlayout.h file, where STACK would be the address below the KERBASE. (STACK = KERNBASE(0x80000000) - 0x01)



```
// Memory layout

#define EXTMEM  0x100000            // Start of extended memory
#define PHYSTOP 0xE000000           // Top physical memory
#define DEVSPACE 0xFE000000         // Other devices are at high addresses

// Key addresses for address space layout (see kmap in vm.c for layout)
#define KERNBASE 0x80000000         // First kernel virtual address
#define KERNLINK (KERNBASE+EXTMEM)  // Address where kernel is linked
#define STACK    (KERNBASE - 0x01)  // Base of stack

#define V2P(a) (((uint) (a)) - KERNBASE)
#define P2V(a) (((void *) (a)) + KERNBASE)

#define V2P_WO(x) ((x) - KERNBASE)    // same as V2P, but without casts
#define P2V_WO(x) ((x) + KERNBASE)    // same as P2V, but without casts
```

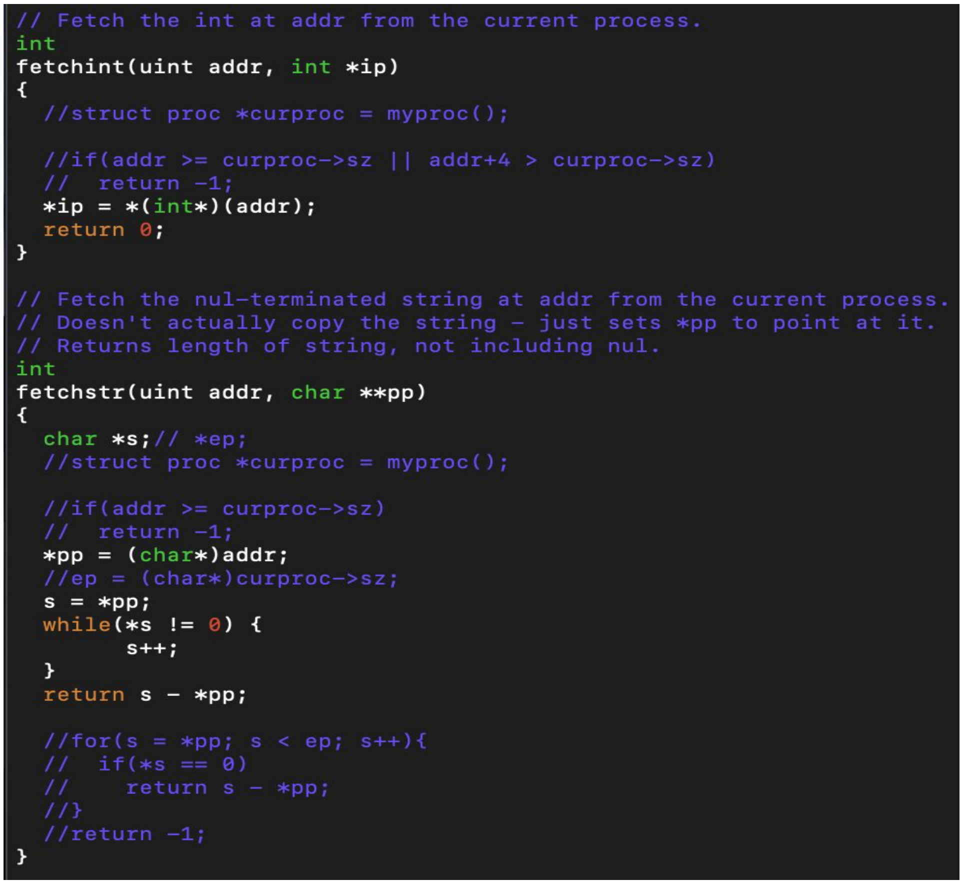
2. Then we modified the code that created the stack in the exec.c file so that it would be placed right under the kernel base.
   a. We did this by changing the second and third parameter. Second parameter being STACK, and the third parameter being PGROUNDUP(STACK).
   b. We then assign STACK to "sp".



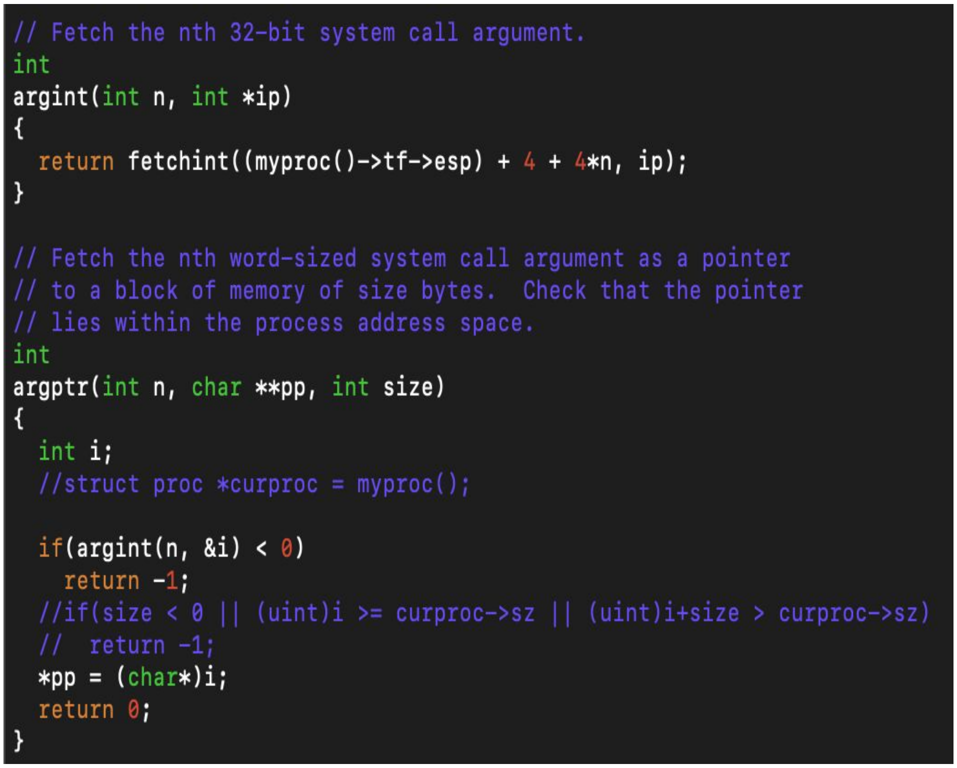
```
  if((sz = allocuvm(pgdir, STACK, PGROUNDUP(STACK))) == 0)
    goto bad;
// clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
  sp = STACK;
```

3. Next, we removed all the checks that used "sz" in the syscall.c file.
   a. Removed the check in the fetching function.
   b. Removed the check in the fetchstr function and modified it so that it would keep its functionality.

c. Removed the check in the argptr function.

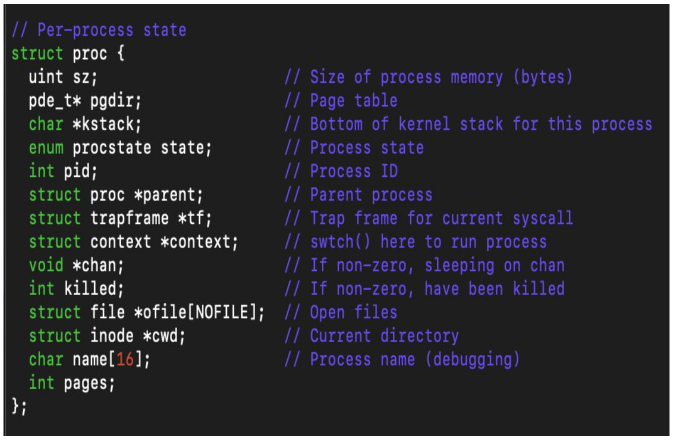


```
// Fetch the int at addr from the current process.
int
fetchint(uint addr, int *ip)
{
  //struct proc *curproc = myproc();

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

// Fetch the nul-terminated string at addr from the current process.
// Doesn't actually copy the string - just sets *pp to point at it.
// Returns length of string, not including nul.
int
fetchstr(uint addr, char **pp)
{
  char *s;// *ep;
  //struct proc *curproc = myproc();

  //if(addr >= curproc->sz)
  //  return -1;
  *pp = (char*)addr;
  //ep = (char*)curproc->sz;
  s = *pp;
  while(*s != 0) {
        s++;
  }
  return s - *pp;

  //for(s = *pp; s < ep; s++){
  //  if(*s == 0)
  //    return s - *pp;
  //}
  //return -1;
}
```

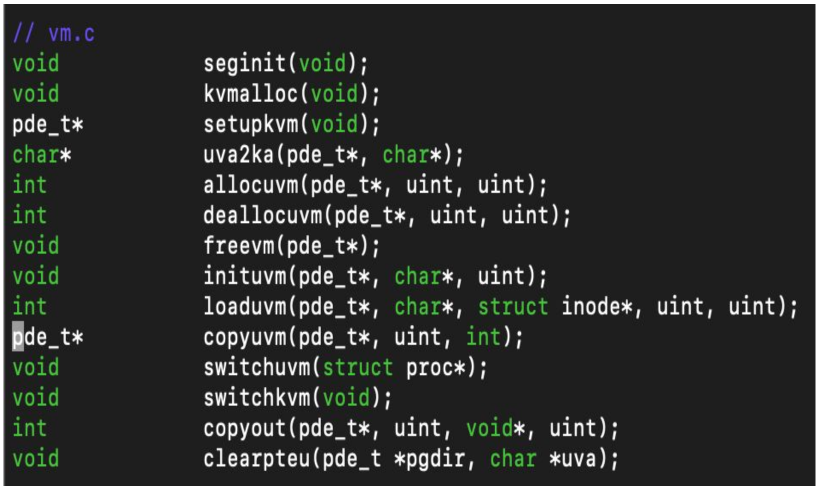


```
// Fetch the nth 32-bit system call argument.
int
argint(int n, int *ip)
{
  return fetchint((myproc()->tf->esp) + 4 + 4*n, ip);
}

// Fetch the nth word-sized system call argument as a pointer
// to a block of memory of size bytes.  Check that the pointer
// lies within the process address space.
int
argptr(int n, char **pp, int size)
{
  int i;
  //struct proc *curproc = myproc();

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

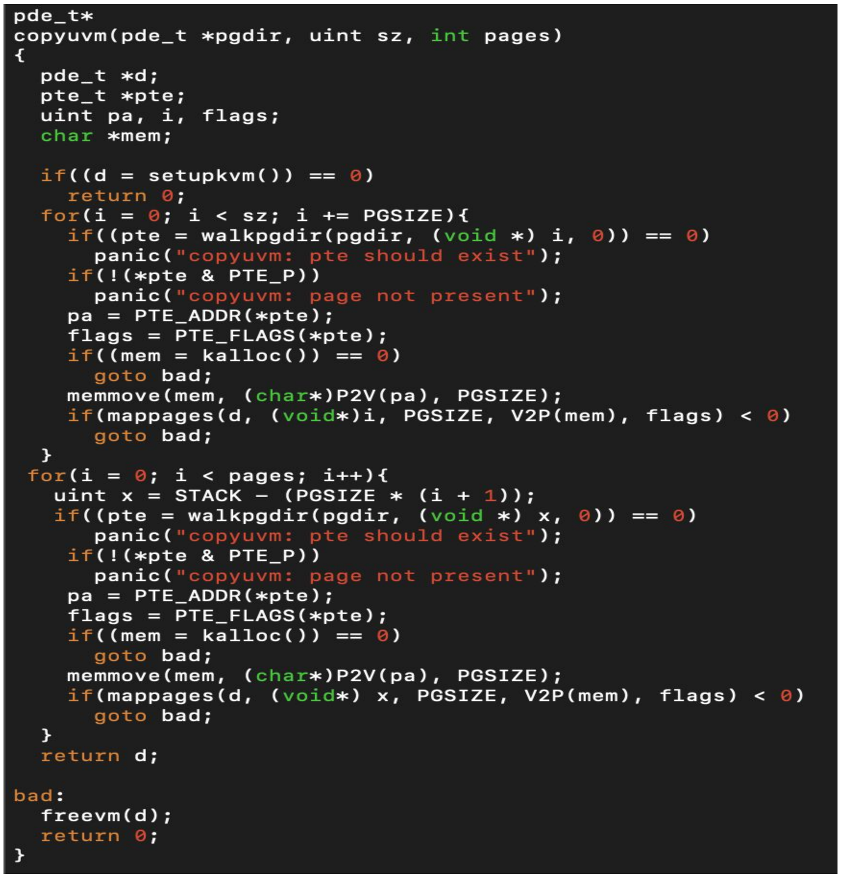
4. We then modified the copyuvm function in the vm.c file so that it would take into account the new stack.
   a. We first added a “int pages” in the proc structure in the proc.h file.
   b. Then, we modified the copyuvm function so that it would take another parameter which would be an “int pages”. To do this we had to modify the defs.h and the vm.c file.
   c. Next, we added another for-loop which would iterate over all the pages and the new virtual address range of the stack.
      i. To get the virtual address we created a variable called “x” that would be assigned the result of the STACK(stack base) - (PGSIZE(page size) * (#of pages + 1).
      ii. The rest of the for-loop had the same functionality as the first for-loop.



```c
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int pages;
};
```

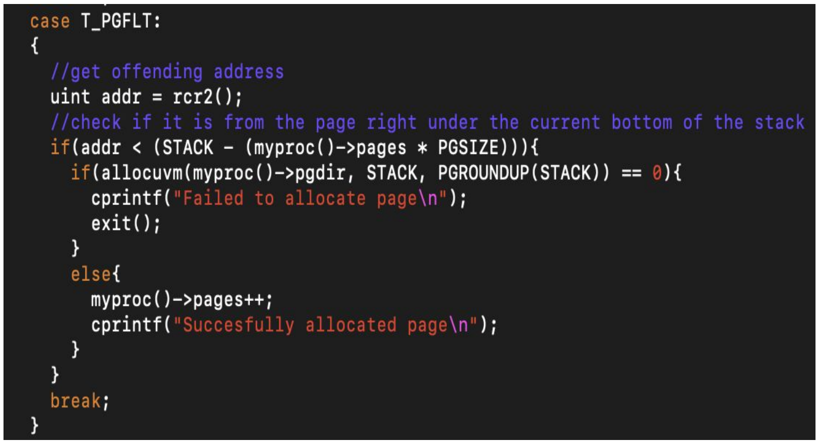


```c
// vm.c
void            seginit(void);
void            kvmalloc(void);
pde_t*          setupkvm(void);
char*           uva2ka(pde_t*, char*);
int             allocuvm(pde_t*, uint, uint);
int             deallocuvm(pde_t*, uint, uint);
void            freevm(pde_t*);
void            inituvm(pde_t*, char*, uint);
int             loaduvm(pde_t*, char*, struct inode*, uint, uint);
pde_t*          copyuvm(pde_t*, uint, int);
void            switchuvm(struct proc*);
void            switchkvm(void);
int             copyout(pde_t*, uint, void*, uint);
void            clearpteu(pde_t *pgdir, char *uva);
```



```
pde_t*
copyuvm(pde_t *pgdir, uint sz, int pages)
{
  pde_t *d;
  pte_t *pte;
  uint pa, i, flags;
  char *mem;

  if((d = setupkvm()) == 0)
    return 0;
  for(i = 0; i < sz; i += PGSIZE){
    if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
      panic("copyuvm: pte should exist");
    if(!(*pte & PTE_P))
      panic("copyuvm: page not present");
    pa = PTE_ADDR(*pte);
    flags = PTE_FLAGS(*pte);
    if((mem = kalloc()) == 0)
      goto bad;
    memmove(mem, (char*)P2V(pa), PGSIZE);
    if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0)
      goto bad;
  }
 for(i = 0; i < pages; i++){
   uint x = STACK - (PGSIZE * (i + 1));
   if((pte = walkpgdir(pgdir, (void *) x, 0)) == 0)
      panic("copyuvm: pte should exist");
    if(!(*pte & PTE_P))
      panic("copyuvm: page not present");
    pa = PTE_ADDR(*pte);
    flags = PTE_FLAGS(*pte);
    if((mem = kalloc()) == 0)
      goto bad;
    memmove(mem, (char*)P2V(pa), PGSIZE);
    if(mappages(d, (void*) x, PGSIZE, V2P(mem), flags) < 0)
      goto bad;
  }
  return d;

bad:
  freevm(d);
  return 0;
}
```

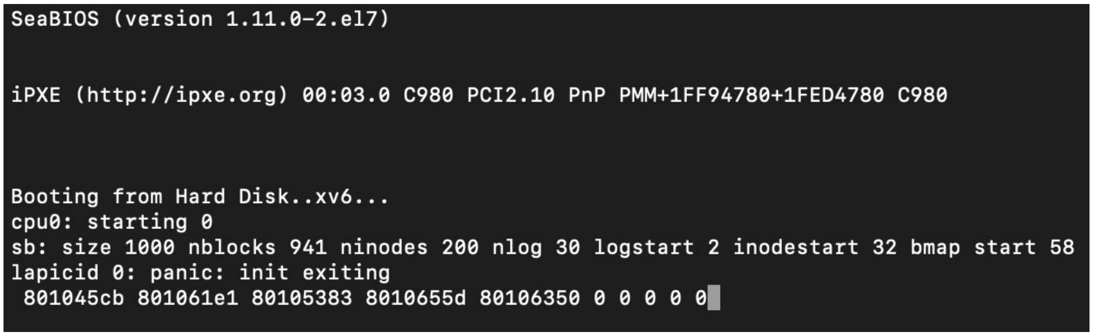
5. Finally, we had to grow the stack and this was done by adding a case for a page fault in the trap.c file.
   a. In the trap function we added a case called “T_PGFLT”
      i. First, we created a variable “uint addr” which would be assigned the offending address that would be obtained using the rcr2 function.
      ii. Then we checked if the “addr” was less than the current bottom of the stack.
      iii. If the above is true, then it would allocate a new page using allocuvm, and it would increase the page counter by 1.



```
case T_PGFLT:
{
  //get offending address
  uint addr = rcr2();
  //check if it is from the page right under the current bottom of the stack
  if(addr < (STACK - (myproc()->pages * PGSIZE))){
    if(allocuvm(myproc()->pgdir, STACK, PGROUNDUP(STACK)) == 0){
      cprintf("Failed to allocate page\n");
      exit();
    }
    else{
      myproc()->pages++;
      cprintf("Succesfully allocated page\n");
    }
  }
  break;
}
```

Errors:

- For the lab we attempted to run the code on qemu but we received a panic error. We were not able to figure out how to fix the issue so we attempted to write the code for the lab to the best of our ability. We attempted to debug it using gdb, but it gave us a message saying that the panic was at console.c at line 123 which is a file that we did not modify.



```
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
lapicid 0: panic: init exiting
 801045cb 801061e1 80105383 8010655d 80106350 0 0 0 0 0
```