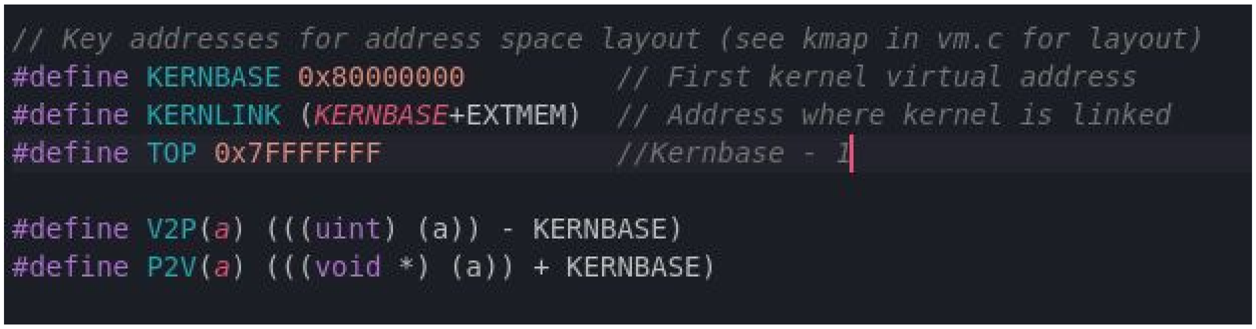
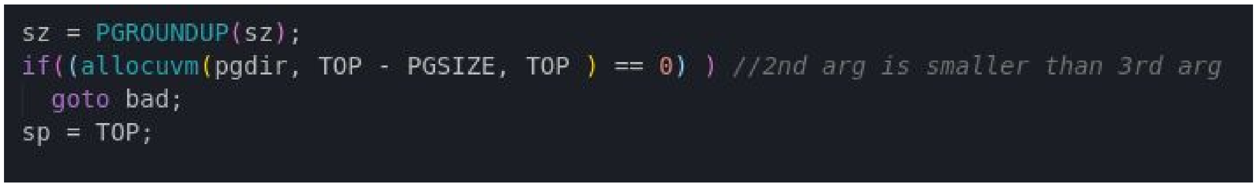
Phillip Nguyen 862050241
Fangsheng Chao 862016183

First we went into memlayout.h and added a define for the location for the new stack which is right under KERNBASE.



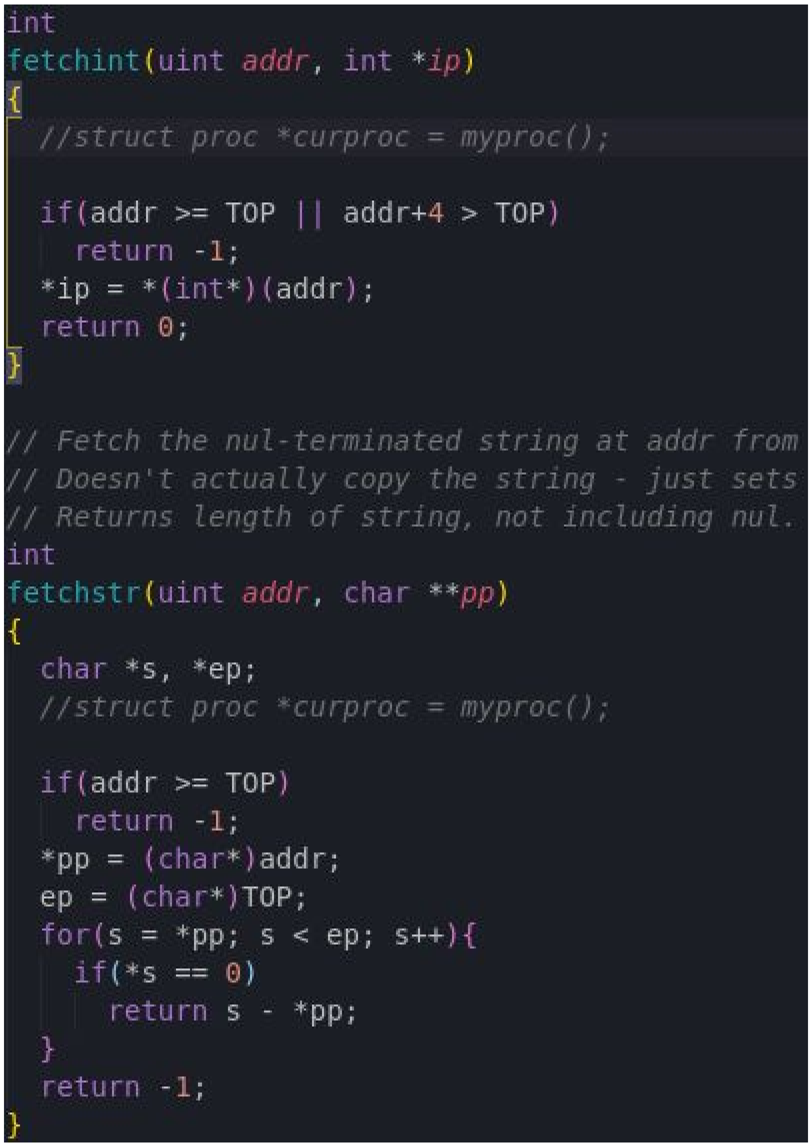
```
// Key addresses for address space layout (see kmap in vm.c for layout)
#define KERNBASE 0x80000000         // First kernel virtual address
#define KERNLINK (KERNBASE+EXTMEM)  // Address where kernel is linked
#define TOP 0x7FFFFFFF              //Kernbase - 1

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

We then went into exec.c and modified how we use allocuvm. We now need to move the stack to where the new top which is called TOP. We changed the parameters around to move the stack. We also initialized the stack pointer to TOP



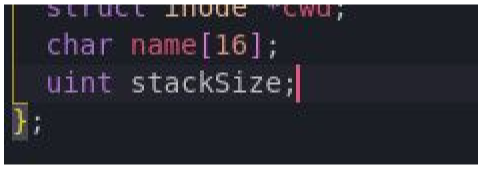
```
sz = PGROUNDUP(sz);
if((allocuvm(pgdir, TOP - PGSIZE, TOP ) == 0) ) //2nd arg is smaller than 3rd arg
  goto bad;
sp = TOP;
```

Now we have to clean up code in syscall.c, we have to change the checks for sz to the new TOP. Since this is the new location of the stack.



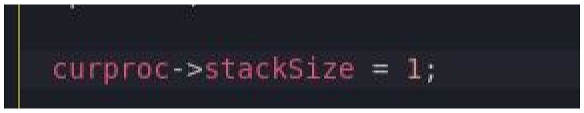
```c
int
fetchint(uint addr, int *ip)
{
  //struct proc *curproc = myproc();

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

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

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

We then added a variable in proc.h to keep track of how many pages over stack takes up



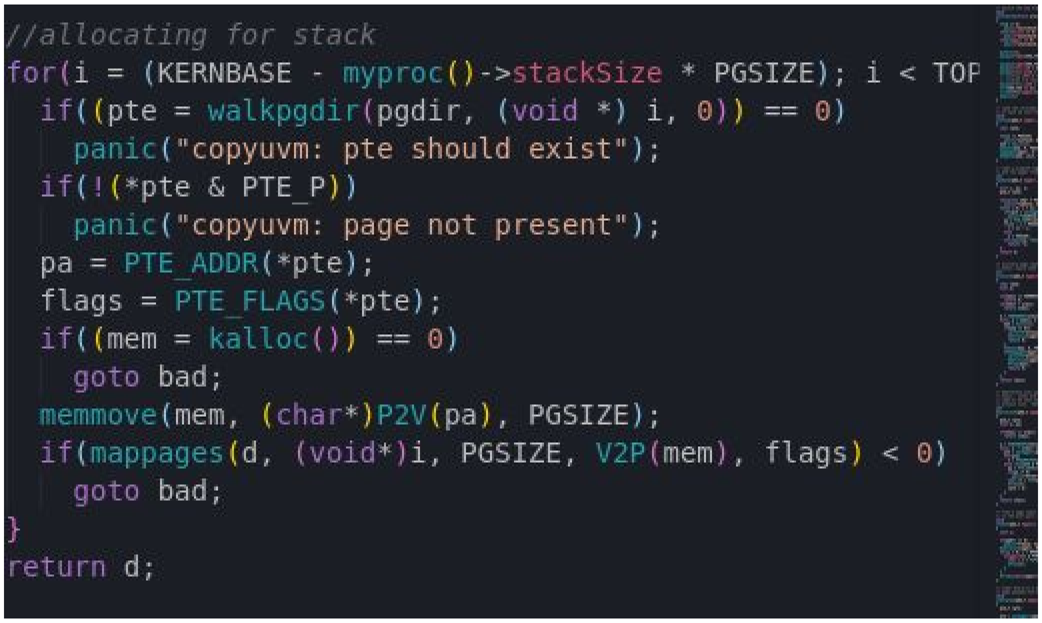
```c
  struct inode *cwd;
  char name[16];
  uint stackSize;
};
```

And then initalize it in exec.c to 1



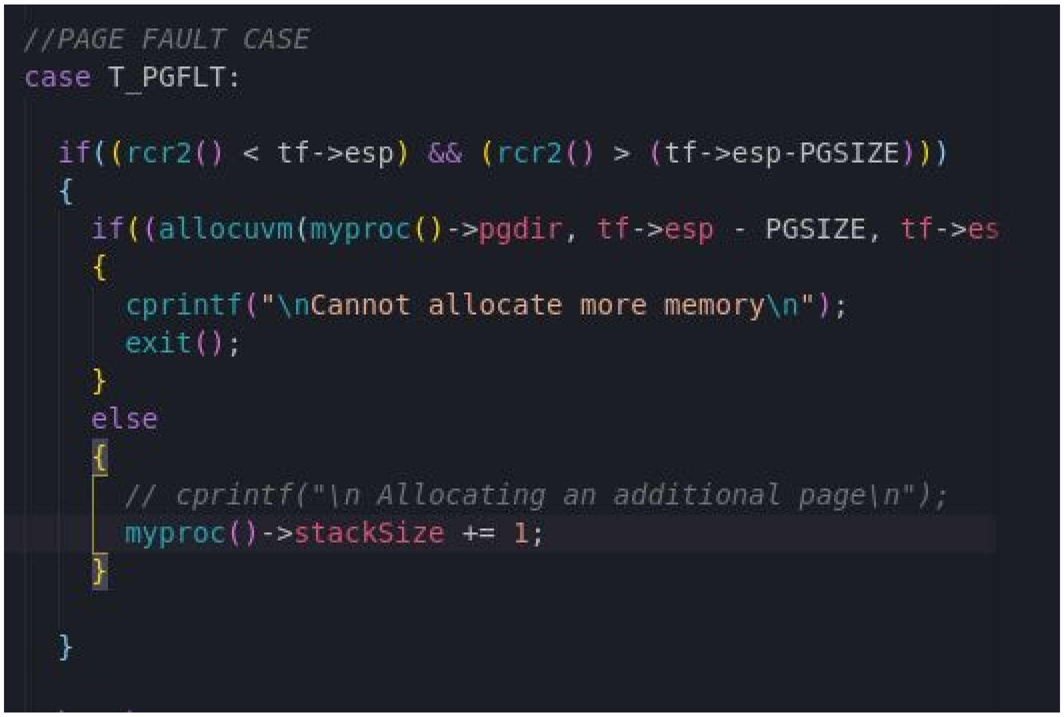
```c
  curproc->stackSize = 1;
```

We then go into vm.c and add an additional for loop for copyuvm to copy the space of the new stack



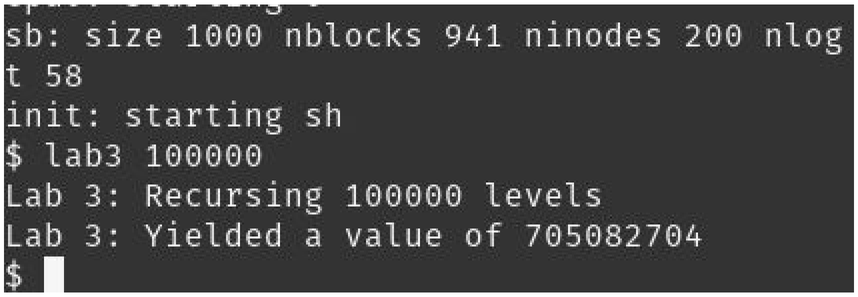
```
//allocating for stack
for(i = (KERNBASE - myproc()->stackSize * PGSIZE); i < TOP
  if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
    panic("copyuvm: pte should exist");
  if(!(*pte & PTE_P))
    panic("copyuvm: page not present");
  pa = PTE_ADDR(*pte);
  flags = PTE_FLAGS(*pte);
  if((mem = kalloc()) == 0)
    goto bad;
  memmove(mem, (char*)P2V(pa), PGSIZE);
  if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0)
    goto bad;
}
return d;
```

Now we have to grow the stack, we added the variable earlier. We now add a check inside trap.c for page faults. This makes sure we have enough memory to allocate when we grow the stack. We also increment the size variable for our stack.



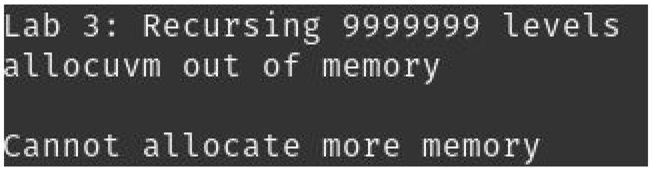
```
//PAGE FAULT CASE
case T_PGFLT:

  if((rcr2() < tf->esp) && (rcr2() > (tf->esp-PGSIZE)))
  {
    if((allocuvm(myproc()->pgdir, tf->esp - PGSIZE, tf->es
    {
      cprintf("\nCannot allocate more memory\n");
      exit();
    }
    else
    {
      // cprintf("\n Allocating an additional page\n");
      myproc()->stackSize += 1;
    }

  }
```

Here is an example output with the given test bench



```
sb: size 1000 nblocks 941 ninodes 200 nlog
t 58
init: starting sh
$ lab3 100000
Lab 3: Recursing 100000 levels
Lab 3: Yielded a value of 705082704
$ 
```

Here is an example output for when you run out of memory



```
Lab 3: Recursing 9999999 levels
allocuvm out of memory

Cannot allocate more memory
```