```
// Allocate two pages at the next page boundary.
// Make the first inaccessible.  Use the second as the user sta
sz = PGROUNDUP(sz);
if((allocuvm(pgdir, STKBASE-PGSIZE,  STKBASE) == 0))
  goto bad;
clearpteu(pgdir, (char*)(STKBASE-PGSIZE));
sp = STKBASE;
//sp = sz;
```

Changed the start of the stack and cleared the buffer (in exec.c)

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

Changing checks since it used to compare to our old sz (in sysfile.c)

```
for(i = cur->stkloc; i <= STKBASE ; i +=PGSIZE){
    if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
      panic("copyuvm: pte should exist");
    if(!(*pte & PTE_P))
      panic("copyuvm: page not present2");
    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){
        kfree(mem);
      goto bad;
  }
}
```

Changed copyuvm to now go over the new stack (in vm.c)

```
    break;
  case T_PGFLT:
    if((rcr2() < STKBASE)){
        allocuvm(myproc()->pgdir, STKBASE- ((myproc()->numpages+1)*PGSIZE),STKBASE
- ((myproc() -> numpages)*PGSIZE));
        myproc()->numpages++;
}
break;
```

Made a new page fault test

```
// 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 numpages;
  uint stkloc;
;
```

Updated proc to hold the number of pages and the location of the stack (in proc.h)

```
// Commit to the user image.
oldpgdir = curproc->pgdir;
//LAB3 added ---------------------------------------------------
curproc->pgdir = pgdir;
curproc->sz = sz;
curproc->tf->eip = elf.entry;  // main
curproc->tf->esp = sp;

curproc->stkloc = (uint)PGROUNDDOWN(sp);
curproc->numpages = 1;
switchuvm(curproc);
freevm(oldpgdir);
return 0;
```

Updated numpages and added the stack location to proc