Anthony Mejia - SID: **861152139**
Christian Campos - SID: **862080812**

Lab 3: Memory Management

Part 1: Changing memory layout

The main goal for this part is to arrange the xv6 project address space to look more like Linux as shown in lab 3 instruction. The only changes we're making is the xv6 user memory layout. The xv6 user memory layout starts at 0 and goes up to KERNBASE.

For our memory layout approach, we created a second user stack named KERNBASE-1 in the exec.c file.This second user stack; KERNBASE-1 points to one address lower than the KERNBASE. Then, we created an int stackPage in the proc.h file to calculate the amount of pages to allocate for this new created user stack, KERNBASE-1. In doing so , we made the KERNBASE inaccessible and forced xv6 to use our user stack.

To make xv6 to recognize our changes, we modified the syscall.c in the following functions: fetchint(uint addr, int *ip) ,fetchstr(uint addr, char **pp), and argptr(int n, char **pp, int size). These changes are needed to use the top address of our user stack memory ; KERNBASE-1. In addition, we added another loop in the copyuvm function in vm.c to copy our stack memory separately from the original xv6 code memory. Finally, we change the function fork(void) in proc.c to copy the pageStacks field from parent process to child process.

Part 2: Implement stack growth

The main goal for this part is to automatically grow the stack backwards when needed. When the stack grows beyond its allocated pages , it will cause a page fault since it's accessing an unmapped page. The page fault is due to T_PGFLT not being handled in the trap handler in trap.c. This means it goes to handling unknown traps and causes a kernel panic.

To handle the page faults, we add a case in the trap.c file. For our approach , we modified thetrap(struct trapframe *tf) in the trap.c. We added a T_PGFLT case in the switch case in the thetrap(struct trapframe *tf) function. T_PGFLT case checks if the page is caused by an access to the page right under the current top of the stack. If this is the case, we allocate and map the page and we're done as shown below. Also, If the page fault is caused by a different address , then the default handler handles the page fault and causes a kernel panic like we did before.

The following screenshots below are our modification of the code we made in lab 3

proc.h:

```
int stackPages;              // Amount of pages to allocate for this stack - lab 3
```

proc.c:

```
np->stackPages = curproc->stackPages;
```

syscall.c:

```
int
fetchint(uint addr, int *ip)
{
  if(addr >= (KERNBASE-1) || addr+4 > (KERNBASE-1))
    return -1;
  *ip = *(int*)(addr);
  return 0;
}
```

```
int
fetchstr(uint addr, char **pp)
{
  char *s, *ep;

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

```
int
argptr(int n, char **pp, int size)
{
  int i;

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

exec.c:

```
// Added for lab 3
sz = PGROUNDUP(sz);
sp = KERNBASE-1;
if((allocuvm(pgdir, sp - PGSIZE, sp)) == 0)
  goto bad;

curproc->stackPages = 1;
cprintf("Initial number of pages by the process: %d\n", curproc->stackPages);
```

vm.c:

```
  uint t = KERNBASE-1;
  t = PGROUNDDOWN(t);

  for(i = t; i > t - (curproc->stackPages) * PGSIZE; 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);
```

trap.c:

```
// Added for lab3
case T_PGFLT: ;
  uint va = rcr2();
  if (va >KERNBASE-1){
        cprintf("from trap access > KERNBASE");
        exit();
      }
  va = PGROUNDDOWN(va);
  if (allocuvm(myproc()->pgdir, va, va + PGSIZE) == 0) {
    cprintf("case T_PGFLT from trap.c: allocuvm failed. Number of current allocated pages: %d\n", myproc()->stackPages);
    exit();
  }
  myproc()->stackPages++;
  cprintf("case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: %d\n", myproc()->stackPages);
  break;
```

Lab 3 Test Code:

```
#include "types.h"
#include "user.h"

// 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();
}
```

To run our test file; lab3.c
Use make clean qemu-nox inside the xv6 directory.
For our test program, we entered lab3 100, lab3 500 and lab3 1000 to run our test code program as shown below

Test output:

```
$ lab3 100
Initial number of pages by the process: 1
Lab 3: Recursing 100 levels
Lab 3: Yielded a value of 5050
$ lab3 500
Initial number of pages by the process: 1
Lab 3: Recursing 500 levels
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 2
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 3
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 4
Lab 3: Yielded a value of 125250
$ lab3 1000
Initial number of pages by the process: 1
Lab 3: Recursing 1000 levels
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 2
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 3
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 4
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 5
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 6
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 7
case T_PGFLT from trap.c: allocuvm succeeded. Number of pages allocated: 8
Lab 3: Yielded a value of 500500
$
```