

# Lab 3 Report

Celyna Su (SID: 862037643) & Wei Xin Koh (SID: 862191103)

For Lab 3, we have implemented the following changes to the code:

1. Introduce `USERSTACKBASE` as in `memlayout.h`

```
10 | #define USERSTACKBASE (KERNBASE - 1) // Start of the user stack
```

2. Define `copyuvm(pde_t*, uint, uint)` in `defs.h`

```
183 | pde_t* copyuvm(pde_t*, uint, uint);
```

3. Add `stackSize` to `copyuvm` in `vm.c`. Iterate over virtual address range from top of `USERSTACKBASE` to end of user stack and copies over user stack page by page (`kmallocs` a page for each one, `memmove`s to create a copy from the parent, and then `mappages()` to add it to page table).

```
360 | pde_t*
361 | copyuvm(pde_t *pgdir, uint sz, uint stackSize)
362 | {
363 |     pde_t *d;
364 |     pte_t *pte;
365 |     uint pa, i, flags;
366 |     char *mem;
367 |
368 |     if ((d = setupkvm()) == 0)
369 |         return 0;
370 |     for (i = 0; i < sz; i += PGSIZE) {
371 |         if ((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
372 |             panic("copyuvm: pte should exist");
373 |         if (!(*pte & PTE_P))
374 |             panic("copyuvm: page not present");
375 |         pa = PTE_ADDR(*pte);
376 |         flags = PTE_FLAGS(*pte);
377 |         if ((mem = kalloc()) == 0)
378 |             goto bad;
379 |         memmove(mem, (char*)P2V(pa), PGSIZE);
380 |         if (mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0)
381 |             goto bad;
382 |     }
383 |
384 |     for (i = USERSTACKBASE - PGSIZE + 1; stackSize > 0; i -= PGSIZE, stackSize--) {
385 |         if ((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
386 |             panic("copyuvm: pte should exist");
387 |         if (!(*pte & PTE_P))
388 |             panic("copyuvm: page not present");
389 |         pa = PTE_ADDR(*pte);
390 |         flags = PTE_FLAGS(*pte);
391 |         if ((mem = kalloc()) == 0)
392 |             goto bad;
393 |         memmove(mem, (char*)P2V(pa), PGSIZE);
394 |         if (mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0)
395 |             goto bad;
396 |     }
397 |
398 |     return d;
399 |
400 | bad:
401 |     freevm(d);
402 |     return 0;
403 | }
```

4. Add `stackSize` to `proc` structure in `proc.h`

```
42 | uint stackSize; // Size of the stack for the process
```

5. Pass in current process's `stackSize` to `copyvm` function call in `proc.c`, in `fork()`

```
192 // Copy process state from proc.
193 if ((np->pgdir = copyvm(curproc->pgdir, curproc->sz, curproc->stackSize)) == 0) {
194     kfree(np->kstack);
195     np->kstack = 0;
196     np->state = UNUSED;
197     return -1;
198 }
```

6. Initialise current process's `stackSize` to 1 in `exec.c`. Allocate user stack using `szStack`.

```
63 // Allocate two pages at the next page boundary.
64 // Make the first inaccessible. Use the second as the user stack.
65 sz = PGROUNDUP(sz);
66 // if((sz = allocvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
67 //     goto bad;
68 // clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
69 // sp = sz;
70 uint szStack = KERNBASE - PGSIZE;
71 if ((szStack = allocvm(pgdir, szStack, szStack + 8)) == 0)
72     goto bad;
73 sp = USERSTACKBASE;
```

```
100 // Commit to the user image.
101 oldpgdir = curproc->pgdir;
102 curproc->stackSize = 1;
103 curproc->pgdir = pgdir;
104 curproc->sz = sz;
105 curproc->tf->eip = elf.entry; // main
106 curproc->tf->esp = sp;
107 switchvm(curproc);
108 freevm(oldpgdir);
109 return 0;
```

7. Check addresses against `USERSTACKBASE` in `syscall.c`

```
16 // Fetch the int at addr from the current process.
17 int
18 fetchint(uint addr, int *ip)
19 {
20     // struct proc *curproc = myproc();
21
22     if(addr >= USERSTACKBASE || addr+4 > USERSTACKBASE)
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 >= USERSTACKBASE)
38         return -1;
39     *pp = (char*)addr;
40     ep = (char*)USERSTACKBASE;
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 >= USERSTACKBASE || (uint)i+size > USERSTACKBASE)
67         return -1;
68     *pp = (char*)i;
69     return 0;
70 }
```

8. Implement `T_PGFLT` trap in `trap.c` as one of the switch cases in `trap()`, to check whether there is a page fault caused by an access to the page right under the current top of the stack. If so, allocate and map the page, else go to the default handler and print error message in `allocuvmm`.

```
81 | case T_PGFLT:
82 |     ;
83 |     uint faultAddr = rcr2();
84 |     uint numPages = myproc()->stackSize + 1;
85 |
86 |     if (faultAddr >= USERSTACKBASE - ((PGSIZE * numPages) + 1)) {
87 |         if (allocuvmm(myproc()->pgdir, PGROUNDDOWN(faultAddr), PGROUNDDOWN(faultAddr) + 8) == 0){
88 |             cprintf("There was an error in allocuvmm\n");
89 |             break;
90 |         }
91 |
92 |         myproc()->stackSize += 1;
93 |         cprintf("Successfully increased the size of the stack\n");
94 |         break;
95 |     }
```

9. Test file: lab3.c

```
1 | #include "types.h"
2 | #include "user.h"
3 |
4 | // Prevent this function from being optimized, which might give it closed form
5 | #pragma GCC push_options
6 | #pragma GCC optimize ("O0")
7 | static int
8 | recurse(int n)
9 | {
10 |     if(n == 0)
11 |         return 0;
12 |     return n + recurse(n - 1);
13 | }
14 | #pragma GCC pop_options
15 |
16 | int
17 | main(int argc, char *argv[])
18 | {
19 |     int n, m;
20 |
21 |     if(argc != 2){
22 |         printf(1, "Usage: %s levels\n", argv[0]);
23 |         exit();
24 |     }
25 |
26 |     n = atoi(argv[1]);
27 |     printf(1, "Lab 3: Recursing %d levels\n", n);
28 |     m = recurse(n);
29 |     printf(1, "Lab 3: Yielded a value of %d\n", m);
30 |     exit();
31 | }
```

[illegible]