Lab Report: Scheduler

Win Thant Tin Han SID: 862258943

12/10/2023

1 Changes and Actions Taken

Here is a general summary of the steps I took as I worked on this lab:

- 1. 'exec.c' – modified exec(), created TOPBASE user stack, which is (KERNBASE-1).
- 2. 'memlayout.h' – added defintion of TOPBASE
- 3. 'proc.h' added variable under 'struct proc', called stack _alloc, which has the page stacks
- 4. 'proc.c' worked on fork(void) function, to copy the page stacks
- 5. 'Makefile' added test file.
- 6. 'syscall.c' modified fetchint, fetchstr, argptr.
- 7. 'trap.c' added a case to handle page faults (part 2/extra credit)
- 8. 'vm.c' modified copyuvm

2 Part 1: Memory Layout Change

2.1 exec.c

In the `exec.c` file, the `exec()` function was modified to create a TOPBASE user stack, positioned at `KERNBASE-1`. In other words, TOPBASE points to one address below the KERNBASE. This involved changing the parameters passed to the `allocuvm` function to allocate and map the stack.
After figuring out in '`exec.c`' where xv6 allocates and initializes the user stack; then, we have to think about how we can change that to use a page at the high-end of the xv6 user address space, instead of one between the code and heap.

2.2 memlayout.h

The `memlayout.h` file was updated to include a definition for `TOPBASE`, representing the top of the user address space.

2.3 proc.h

Within `proc.h`, a new variable called `stack_alloc` was added under the `struct proc`. This variable keeps track of the page stacks, and makes it so that the xV6 uses the newly created user stack.

2.4 proc.c

The `fork(void)` function in `proc.c` was modified to handle the copying of page stacks for child processes. This ensures that the child process inherits the correct stack information from the parent.

2.5 Makefile

The `Makefile` was updated to include the test file associated with the changes made to the XV6 memory layout.

2.6 syscall.c

In `syscall.c`, modifications were made to functions such as `fetchint`, `fetchstr`, and `argptr` to account for the changes in stack location.
The modifications are made so that we can use the top address of our user stack memory (TOPBASE).

2.7 vm.c

The `vm.c` file was updated, specifically the `copyuvm` function, to accommodate the changes in the stack location. We modified the function so that the memory from 0 to $curproc() \rightarrow sz$ contains the code and the heap. A one page stack grows from the KERNBASE towards 0 and followed by a page guard.

3 Part 2: Stack Growing

In this part of the lab, the idea is to be able to grow the stack backwards when needed. If a stack grows beyond the pages that are allocated, it will throw a page fault for accessing something unmapped. To handle the page faults, a case is needed in 'trap.c' file. Specifically, the trap is `T_PGFLT`, which is not handled by the base code in the trap handler.

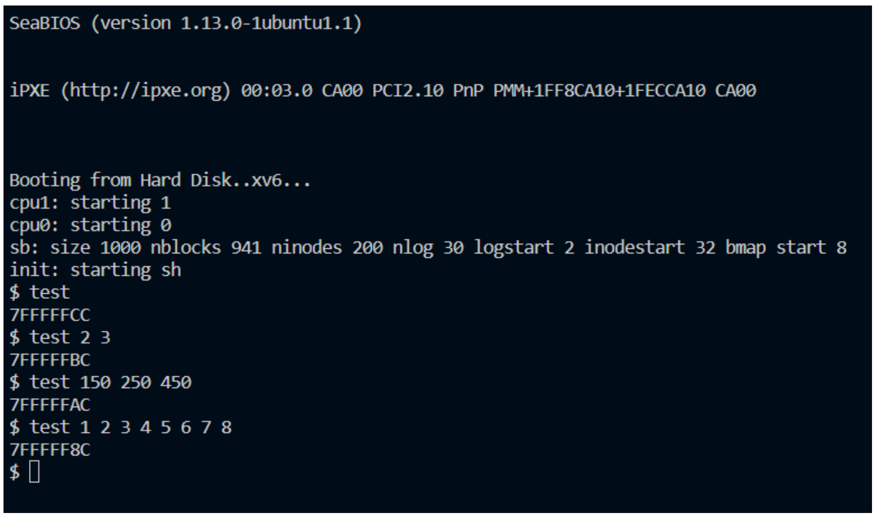
3.1 trap.c

A case for handling page faults (trap number 14) was added to the `trap.c` file. This modification checks the address causing the page fault and dynamically grows the stack when necessary.
It checks if page fault is caused by access to unmapped page. If it is, we allocate and map the page. Otherwise, we go to default handler, and do a kernel panic as before.

4 Results and What I learned

From this lab, I learned the basics about memory management, and how i can modify virtual address space (layout) management, such as implementing stack growing and adding stack to top of user address space. The output for test.c is:

Part1:



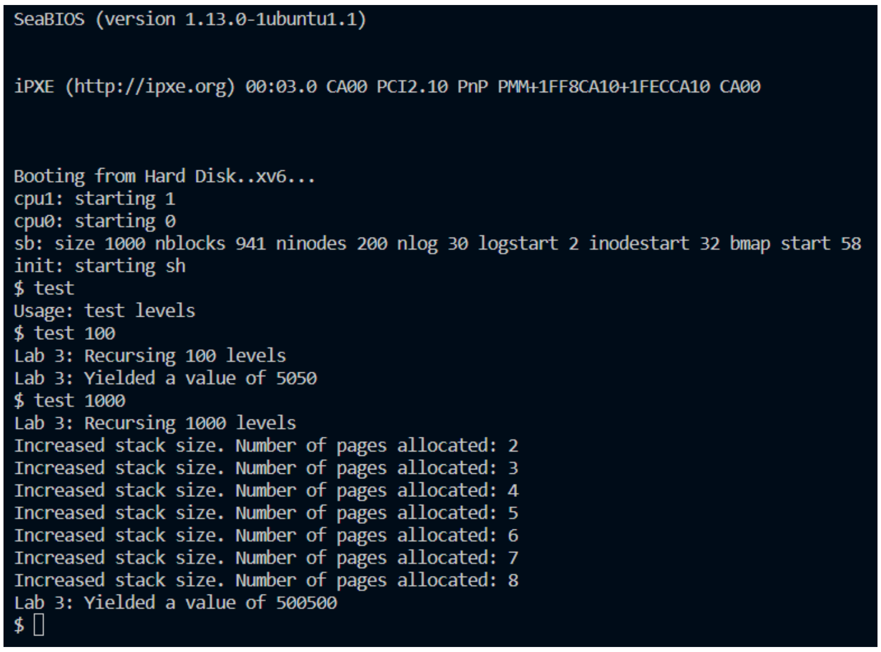
```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 8
init: starting sh
$ test
7FFFFFCC
$ test 2 3
7FFFFFBC
$ test 150 250 450
7FFFFFAC
$ test 1 2 3 4 5 6 7 8
7FFFFF8C
$ 
```

Part2:



```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ test
Usage: test levels
$ test 100
Lab 3: Recursing 100 levels
Lab 3: Yielded a value of 5050
$ test 1000
Lab 3: Recursing 1000 levels
Increased stack size. Number of pages allocated: 2
Increased stack size. Number of pages allocated: 3
Increased stack size. Number of pages allocated: 4
Increased stack size. Number of pages allocated: 5
Increased stack size. Number of pages allocated: 6
Increased stack size. Number of pages allocated: 7
Increased stack size. Number of pages allocated: 8
Lab 3: Yielded a value of 500500
$
```