**Final Project: Multi-Level Page Tables**

**Chris Limson, *climson@gmu.edu***

**Saturday, May 14, 2022**

GMU CYSE 570 Fundamentals of Operating Systems

Prof. Mohamed Gebril - Venkat Sachith Rachapudi, TA

**I. Demonstration**

The following unlisted YouTube video is the presentation and demonstration of all work:

*<https://youtu.be/o3NU-g7JhCo>*

**II. Objective**

Implement multi-level page tables of the Xv6 operating system, which will require revising the virtual memory system to use 4-level pages tables. Xv6 should only be able to run on 64-bit processors.

**III. Challenges**

Completing the project without any group, I had to fully understand the concepts of page tables, and learn how to manipulate them. The C and assembly code revisions necessary to upgrade Xv6 were determined, implemented and debugged. And finally, the differences between the 32 and 64-bit operating systems were demonstrated.

**IV. Research**

Documentation from the official Xv6 book describes how the kernel provides each process with its own private address space and memory by using page tables. Page tables determine how virtual memory maps to physical memory. Because addresses can be referenced with twice as many 64-bit registers that are twice as large as 32-bit registers, page tables must be expanded to four levels made for the purpose of accommodating more addresses.

**V. Architecture**

GCC preprocessor definitions to specify 32 or 64-bits OS compilations were used. With conditional directives, I was able to delineate differences while minimizing code duplication. The preprocessor conditional of the definition was also used to customize the shell prompt to include my signature and timestamp. A Makefile variable was made to specify the QEMU emulator architecture and configure it for additional experimentation.

Assembly code for the kernel and C code for virtual memory were the fundamental components that needed to be changed. This affected references to a number of other components and needed updates to remain consistent on type sizes and register names.

entry64.S defines an increase to four page tables for the multiboot entry.

initcode64.S uses movl to $rax instead of the 32-bit pushl $eax.

kernel64.ld defines the fundamental format and architecture elf64-x86-64 to be used by the linker, instead of 32-bit elf32-i386.

swtch64.S and trapasm64.S replace the 32-bit architecture's long registers with twice as many 64-bit standard integer registers.

vectors64.pl is a script that generates the quad vector, versus vector32.pl's long, based on the entry point.

The 64-bit type.h defines the unsigned integer pointer to unsigned long values instead of the 32-bit unsigned integer.

elf.h makes sure to use the new type uint64, updated from uint.

64-bit x85.c doubles the size of the physical descriptors as that in 32-bit, and twice as many long registers as integer registers.

memlayout.h increased the kernel and device base virtual addresses to twice the length of 32-bit.

**VI. Environment**

The entire project was performed on physical Intel Core i7-4790K hardware with Windows 10. It ran Linux VMs with 2 CPUs each on VirtualBox. The ISOs used were 64-bit Ubuntu 20.04 and 32-bit Ubuntu 16.04..

**VII. Procedure, Results**

Assume tester has very basic Linux operation, like installing Ubuntu, copying files to working directory and editing files.

Install the development tools:

sudo apt install gcc make qemu

The unzipped project may still need to change permissions of Perl scripts:

chmod u+x sign.pl vectors32.pl vectors64.pl

**Task A. 32-bit Xv6**

1. In the Ubuntu 20.04 64-bit environment, verify that our platform is 64-bit with x86\_64

uname -a

2. In the decompressed directory with C and assembly code files, open Makefile. In the first two lines, ensure that the Xv6 operating system code architecture is 32-bit and the QEMU emulated architecture is 32-bit

OS = 32

QA = 32

Inspect the rest of Makefile, confirming that only 32-bit-related assembly files will be compiled and 32-bit QEMU will be run.

This is close to the original version of Xv6 with which we started in January 2022 from Stony Brook University. It is just simplified for readability and presentation of essential components.

3. Start from a clean sandbox, then run QEMU with make. All necessary files will be compiled.

make clean

make qemu

4. Test basic OS commands. Observe that there is no PATH variable, so be sure to explicitly identify locations.

ls

mkdir sub

cd sub

ls

../ls ..

To quit:

<ctrl-a> c q <enter>

5. Set parameters for the same 32-bit OS on 64-bit QEMU architecture.

OS = 32

QA = 64

6. Running will not need to compile the OS because its files still exist.

make qemu

7. Test with similar commands as in step 4.

**Task B. 64-bit Xv6**

8. Set parameters.

OS = 64

QA = 64

9. Clean previous compile. Compile incrementally just to illustrate the process, and run.

make clean

make bootblock

make kernel

make xv6.img

make fs.img

make qemu

10. The successful test results are transparent, which was our primary goal. **Our 64-bit Xv6 runs on 64-bit emulated architecture** with no differences.

11. But ensure that 64-bit Xv6 will *only* run on 64-bit QEMU and not 32-bit.

OS = 64

QA = 32

12. **64-bit Xv6 is only able to run on 64-bit processors**.

make qemu

**Task C. 64-bit Xv6 on 32-bit VM hardware**

13. On the Ubuntu 16.04 32-bit environment, verify that our platform is 32-bit with i686

uname -a

QEMU installation actually provides an available qemu-system-x86\_64 binary, regardless of VM 32-bit architecture.

14. Our nominal parameters.

OS = 64

QA = 64

15. Clean, run.

make clean

make qemu

16. Test, regardless of VM 32-bit architecture.

17. Experiment.

OS = 64

QA = 32

18. Verify that again, it is not backwards compatible.

make qemu

**VIII. Conclusion**

As intuition suggests, a 32-bit OS design does indeed run on an emulator of 64-bit architecture, but not the inverse -- 64-bit Xv6 does not run on 32-bit QEMU.

And with its success of step 16, we can conclude that QEMU software simulates some underlying hardware differences, but not all as the emulator does throw warnings about CPUs.

It is interesting that QEMU emulates the x86\_64 64-bit architecture even on a 32-bit VM. I could therefore conceivably test a 64-bit operating system on a 64-bit emulator running on a physical bare-metal 32-bit machine -- if one were available. The mainstream of all computers have migrated to 64-bit hardware since 2003.

**IX. References**

Operating Systems Labs, Nima Honarmand, Stony Brook University

*https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa17/cse306/labs.html*

Xv6 Documentation, MIT

*https://pdos.csail.mit.edu/6.828/2021/xv6/book-riscv-rev2.pdf*

64-bit Port of Xv6, Brian Swetland, GitHub

*https://github.com/swetland/xv6*