OS support on IMS report #2

Yusen Liu
liu797@wisc.edu

Jan. 14 ~Jan. 19

Abstract

After I fixed the error in wcpy_from_user() system call, I continued to make the 64-bit RISC-V Linux run on QEMU not only on my own virtual machine, but also on manaslu (with the shell script executed). I am currently reading the evaluation section of the MEG paper and have started learning Digital Design and Computer Architecture by Professor Onur Mutlu.

1 Methodology

1.1 wcpy_from_user() system call fixed

After a series of debugging and testing, I rechecked the manual page of `strncpy_from_user()`, and the API was defined as: `long strncpy_from_user (char * dst, const char __user * src, long count)`. Since the function aims to copy a NULL-terminated string from user space to kernel space, we need to specify the type of `src` to be `const char __user*` instead of `char*`. Hence, the code shoule be as follows:

```
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE1(wcpy_from_user,
                char __user*, src)
{
  char buf[256];
  long cpy_result = strncpy_from_user(buf, src, sizeof(buf)-1);

  if(cpy_result < 0){
    printk(KERN_INFO "return value of syscall: %ld\n", cpy_result);
    return -EFAULT;
```

```
  }

  printk(KERN_INFO "copied from user: %s\n", buf);
  return 0;
}
```

1.1.1 Functionality test of wcpy_from_user()

After recompiling the kernel, we need to test the functionality of `wcpy_from_user()`. The testing code is as follows:

```
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SYS_wcpy_from_user 441

int main(int argc, char *argv[])
{
  if(argc <= 1)
    exit(EXIT_FAILURE);

  long ret_val = syscall(SYS_wcpy_from_user, argv[1]);

  if(ret_val < 0)
    perror("syscall failed.\n");
  else
    printf("syscall succeed.\n");

  return 0;
}
```

After executing `./test 'hello world'`, the output was: syscall succeed. I then executed `$ dmesg` to check the kernel log buffer and it showed:

```
[ 2178.495202] copied from user: hello world
```

1.2 Running 64-bit RISC-V Linux on QEMU

I have successfully made the 64-bit RISC-V Linux on QEMU run on my virtual machine. However, some problems occurred when I tried to automated the build but I solved them eventually.

The first problem I noticed was that, after I rebooted my virtual machine, the mounted file system did not exist anymore, thus I should re-run the script to build it again. Since building the file system requires root permission, if I would like to operate on the virtual machine (manaslu), it would be fine if it does not reboot that often. Since it is scheduled to reboot on the first Monday of the month at 03:00, I believe it would be fine.

The second problem was, the script that I wrote for mounting the file system did not work well when I was testing on my own virtual machine. When I tried to run the RISC-V Linux on QEMU, I executed the following command:

```
qemu-system-riscv64 -M virt -m 256M -nographic \
-bios opensbi/build/platform/generic/firmware/fw_jump.bin \
-kernel ./linux/arch/riscv/boot/Image \
-drive file=./rootfs.img,format=raw,id=hd0 \
-device virtio-blk-device,drive=hd0 -append "root=/dev/vda rw \
console=ttyS0"
```

The result was: qemu-system-riscv64: -device virtio-blk-device,drive=hd0: Could not reopen file: Permission denied. The only solution was run the command above in root mode, i.e. with `sudo`.

I tried to re-execute the command line by line by typing into the terminal, and it worked well. Hence, there must be some problems in the script. After debugging, I noticed that creating the file `rcS` in the directory `working_dir/rootfs/etc/init.d/` requires root permission, and the script that I wrote was incorrect. The correct script is shown below:

```
#!/bin/bash

sudo tee -a rcS > /dev/null <<EOF
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
/sbin/mdev -s
EOF
```

I put the full build process online, which can be found at: https://www.notion.so/running-64-bit-RISC-V-Linux-on-QEMU-3e7f44aaa41a4087999f1768d4853af4.

1.3 MEG

MEG is a configurable, cycle-exact, RISC-V based full system emulation infrastructure using FPGA and HBM. Since this is the first paper I have read that covers so much details of hardware, I picked a different approach. In order to better understand each sentence, I chose to retype the whole paper sentence by sentence, and reorganized the paper into a much more readable format. Although this approach seems clumsy, I believe it is the best approach for the person who has limited background of hardware to read the paper mainly focusing on hardware design. Thanks to this approach, I got the main idea of the paper even if I have not finished reading yet.

The link to my work can be found at: https://www.notion.so/MEG-b4c0bcdf66e043eda1add08800b992eb.

1.4 Online lecture

As discussed in Section 1.3, I went into every details of the paper about MEG. During the process, I accidentally noticed a free online lecture by Professor Onur Mutlu, which could be accessed on YouTube. I could not help going through the first two lectures and found it fantastically interesting. Especially in his first lecture, where he talked about the design of hardware requires balancing the trade-offs and considering the power budget as well as area budget. That helped me better understand the background of MEG since MEG also covered the power budget for at least 4 times. Hence I made a decision to finish the lectures as soon as possible.

2 Discussion

Unlike adding system calls and building Ubuntu on a virtual machine, building a RISC-V Linux running on QEMU requires much more time. After cycles of testing-and-failure, I successfully made it run. However, I went through the system calls provided but did not find a way to write programs in it except running assembly language. This is the main problem that gets me stuck. The problem also motivates me to learn more about RISC-V instruction set.

In this second report, I did not cover too much details about how to build the system. The reason is, I think I should spend more time on MEG since fully understanding of it and finding the OS support are what I should focus on. I firmly believe that, in order to provide much powerful operating systems for various of hardware, I have to learn more details about computer architecture. Unlike the machine learning courses that I found quite boring, courses by Professor Onur Mutlu is much more interesting. Although I am not allowed to take CS 552 because the lecturer did not response to my emails about waiving CS 352 - the prerequisite of CS 552 at UW-Madison, I have found the best alternative to it.

3 Future plan

I would keep reading the paper and manage to write a paper review before the next meeting. After that, I would go on watching the lectures Digital Design and Computer Architecture in my spare time.