E0253 – OPERATING SYSTEMS: ASSIGNMENT 0

NABARUN SARKAR[1]

CONTENTS

LIST OF TABLES

[1] *Department of Computer Science and Automation, IIsc, Bangalore, India*

1 INTRODUCTION

1.1 Objective of this assignment

In this project, we have to download the Linux kernel sources of version 5.0 and perform following operations:

1.1.1 *Introduce a no-argument system call that returns a long (value 0)*

1.1.2 *Understand how fork() system call works*

1.1.3 *Implement a user-space micro-benchmark (in C) to measure the timing of these system calls*

1.2 Submission requirements

Description: We have to submit a tarball consisting of three things:

1.2.1 *The kernel patch for introducing the my precious syscall*

1.2.2 *User-space program(s) used to measure the timing of system calls and*

1.2.3 *Report in PDF format*

---

2 PART B

2.1 Adding custom system call

Description: Using the git[6] version control software, and creating a new branch based on a version 5.0 of Linux. Then making required modifications and compiling the source code to generate patch file.

2.1.1 *Downloading Linux kernel v5.0 using git*

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
$ cd linux-stable/
$ git checkout v5.0
$ git status
$ git checkout -b e0253nabaruns
```

Switched to a new branch 'e0253nabaruns'

```
$ git status
$ mkdir hello
$ cd hello
$ vim hello.c
```

2.1.2 *Defining a new system call[4]:*

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

asmlinkage long my_precious(void)
{
        printk("Hello world!\n");
        return 0;
}
```

2.1.3 *Create a Makefile in this hello directory*

```
$ vim Makefile
```

Add the following line to it:

```
obj-y := hello.o
```

2.1.4 *Adding hello/ to the kernel's Makefile*

```
$ cd ..
$ vim Makefile
```

Add 'hello/' to the end of the line:

```
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello/
```

2.1.5 *Add the new system call to the system call table:*

```
$ cd arch/x86/entry/syscalls/
$ vim syscall_64.tbl
```

Add a new line:

```
548     64      hello           my_precious
```

2.1.6 *Add new system call to the system call header file:*

```
$ cd ..
$ cd ..
$ cd ..
$ cd ..
$ cd include/linux/
$ vim syscalls.h
```

Add the following line to the end of the document before the endif statement:

```
asmlinkage long sys_hello(void);
```

2.1.7 *Compile the kernel:*

```
$ cd ..
$ cd ..
$ sudo make menuconfig
```

configuration written to .config

```
$ sudo make -j2
```

2.1.8 *Install / update Kernel:*

```
$ sudo make modules_install install
$ sudo shutdown -r now
```

After restarting:

```
$ uname -r
```

5.0.0+

```
$ dmesg
[   160.983395] Hello world
```

2.2 Generating patch file

Description: Using the git version control software to generate patch file.[5][2]

2.2.1 *Steps:*

```
$ git status
```

Output:

```
On branch e0253nabaruns
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   Makefile
  modified:   arch/x86/entry/syscalls/syscall\_64.tbl
  modified:   include/linux/syscalls.h

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  hello/

no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git diff
```

Output:

```
diff --git a/Makefile b/Makefile
index d5713e7..5a85daa 100644
--- a/Makefile
+++ b/Makefile
@@ -958,7 +958,7 @@ endif
 PHONY += prepareo

 ifeq ($(KBUILD_EXTMOD),)
-core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/
+core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/

 vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                      $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
diff --git a/arch/x86/entry/syscalls/syscall_64.tbl b/arch/x86/entry/syscalls/sysc
index f0b1709..a31a90d 100644
--- a/arch/x86/entry/syscalls/syscall_64.tbl
+++ b/arch/x86/entry/syscalls/syscall_64.tbl
@@ -386,3 +386,4 @@
 545     x32     execveat                __x32_compat_sys_execveat/ptregs
 546     x32     preadv2                 __x32_compat_sys_preadv64v2
 547     x32     pwritev2                __x32_compat_sys_pwritev64v2
+548      64     hello                   my_precious
diff --git a/include/linux/syscalls.h b/include/linux/syscalls.h
index 257cccb..b90bb32 100644
--- a/include/linux/syscalls.h
+++ b/include/linux/syscalls.h
@@ -1314,5 +1314,6 @@ static inline unsigned int ksys_personality(unsigned int per

        return old;
 }
+asmlinkage long my_precious(void);

 #endif
```

2.2.2 *Finally, generating the patch file:*

```
$ git format-patch master --stdout > assignmento_nabarun.patch
```

Verify the patch file for any error:

```
$ ./scripts/checkpatch.pl assignmento_nabarun.patch
```

2.3 Applying patch file[3]

```
$ cd linux-5.0/
$ patch -p1 < ../assignmento_nabarun.patch
```

3 PART C

3.1 User-space program used to measure the timing of system calls

Description: For this part of the assignment, a C code was written. We are required to measure the execution time, in terms of CPU cycles, of both the system calls. For this task, you are required to learn how to use the processor's hardware performance counter (the time stamp counter) to measure time.[1]

```c
#include <linux/kernel.h>
#include <sys/syscall.h>
#include  <sys/types.h>
#ifdef _WIN32
#include <intrin.h>
#else
#include <x86intrin.h>
#endif
#include <stdint.h>
#include <stdio.h>
#include <sys/wait.h>


inline
uint64_t readTSC() {
        uint64_t tsc = __rdtsc();
        return tsc;
}

int main()
{
        pid_t  pid, wpid;
        unsigned long long a,b;
        int status;

        //Measuring custom sys call
        a = readTSC();
        long int ret_val = syscall(548);
        b = readTSC();
        printf("System call my_precious returned %ld in %llu\n", ret_val, b-a);

        // Measuring fork() sys call
        a = readTSC();
        pid = fork();

        if(pid < 0){
                printf("Failed to fork process.\n");
                exit(1);
        }
        else if(pid == 0){
                //child process
                b = readTSC();
                printf("Fork created child pid %ld in %llu\n",(long) pid, b-a);
                exit(0);
        }
        else{
            //parent process
                b = readTSC();
                wpid = wait(&status);

                if(WIFEXITED(status)){
                        printf("Fork returned parent pid %ld in %llu\n",(long) pid, b-a);
                }
        }
```

```
55          return 0;
56  }
```

Algorithm 1: The user space code to measure performance

3.1.1 *Output like:*

```
1  System call my_precious returned 0 in 9082
2  Fork created child pid 0 in 441178
3  Fork returned parent pid 3106 in 308194
```

4 PART D

4.1 Measure the execution time

Description: Reporting the measurements based on multiple runs (minimum 10) along with the standard deviation.

4.1.1 *What causes the variations?*

4.1.2 *How would we choose a representative value to indicate execution cycles required for a system call?*

4.2 Measurements

The output is listed in the Table 1.

Table 1: The output of iterations

Iteration	my_precious	fork_child	fork_parent
1	9,082.00	4,41,178.00	3,08,194.00
2	8,491.00	3,72,276.00	2,08,966.00
3	12,096.00	3,31,146.00	1,76,359.00
4	11,904.00	16,37,624.00	2,12,812.00
5	8,368.00	3,64,679.00	2,77,028.00
6	8,038.00	2,64,381.00	1,38,237.00
7	9,738.00	2,95,363.00	1,45,741.00
8	12,284.00	3,34,424.00	2,55,372.00
9	8,099.00	8,77,629.00	1,36,247.00
10	8,275.00	3,06,909.00	1,54,925.00
Mean	**9,637.50**	**5,22,560.90**	**2,01,388.10**
Median	**8,786.50**	**3,49,551.50**	**1,92,662.50**
Std. Dev.	**1,770.49**	**4,29,520.55**	**61,774.13**

As we can see from the Table 1, we do see that the cycles taken by system call my_precious is very low. Whereas the parent process of fork() call is comparatively faster than that of the child.

The median of parent process is lesser with lower standard deviation.

Also, RDTSC returns the cycle count with such variations, probably due to:

- the Time Stamp Counter might not be synchronized between the cores.
- During the execution of the code, it might be possible that the Operating System might have done timer interrupt.
- Another feature that might be another reason that may lead to such cause can be Pipelining.
- The issue may be a result of caching operation that can vary in speed.

REFERENCES

[1] Adam Aviv. Lec. 14: exec()/fork()/wait() cycles for process management, 2016. URL https://www.usna.edu/Users/cs/aviv/classes/ic221/s16/lec/14/lec.html. [Online;].

[2] Nick Desaulniers. Submitting your first patch to the linux kernel and responding to feedback, 2017. URL http://nickdesaulniers.github.io/blog/2017/05/16/submitting-your-first-patch-to-the-linux-kernel-and-responding-to-feedback/. [Online;].

[3] Jesper Juhl. Applying patches to the linux kernel, 2005. URL https://01.org/linuxgraphics/gfx-docs/drm/process/applying-patches.html. [Online;].

[4] Anubhav Shrimal. Adding a hello world system call to linux kernel, 2018. URL https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872. [Online;].

[5] ARIEJAN DE VROOM. How to create and apply a patch with git, 2009. URL https://www.devroom.io/2009/10/26/how-to-create-and-apply-a-patch-with-git/. [Online;].

[6] Ubuntu Wiki. Buildyourownkernel, 2019. URL https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel. [Online;].