Instructions For Task B1: System Calls

1 Introduction

In this task, you will implement an operating system call into a small kernel running on a AMD64 compliant processor. The system call is supposed to return the version of the kernel.

The AMD64 instruction set is an 64-bit extension to the IA32 instruction set. If you have a PC or a recent Mac then you most probably have a processor that is compliant with AMD64. The AMD64 instruction set was originally designed by AMD but was quickly adopted by Intel. Intel call their variant Intel 64. There are some small but notable differences between AMD64 and Intel 64. However, it is relatively straight-forward to design software and operating systems that runs on both variants. Modern operating systems typically runs in 64-bit mode today, although the paradigm shift is underway.

2 Learning goals

During this task you will be working towards the following learning goals:

- You can explain in your own words the differences between a monolithic operating system, a micro-kernel operating system, and an exo-kernel based operating system.
- Given a schematic overview of the software stack of a real time system, you can label, in the diagram, key operating system parts. You can also explain in your own words and in text the purpose of those parts.
- Given reference literature, you can write down in text and in your own words definitions or explanations of the following concepts: System call; Kernel mode; User mode.
- Given a skeleton uniprocessor operating system, you can implement operating system calls.

While working on the task you will also study a small operating system kernel suitable for a small embedded system. You will also familiarize yourself with and use tools, an emulator and a hypervisor to compile and run the kernel.

3 Report and rules

You will have to hand in written reports during the course. The reports should describe all the tasks you have completed. Begin writing on the reports as early as possible. You should also maintain a progress diary!

DTU has a zero tolerance policy on cheating and plagiarism. This also extends to the report and indeed all your work. To copy text passages or source code from someone else without clearly and properly citing your source is considered plagiarism. See the study hand book for further details.

4 Don't Panic!

Operating system development is difficult in that you work directly with the computer hardware which can be very unforgiving. You will also work in a very constrained programming environment. You cannot for instance use any of the standard C library calls as there is no C library. Take your time and be careful when working with the tasks. Make use of the discussion facilities on campusnet and the course staff.

Read the instructions. Reread them.

It is imperative that you read and understand all the code. While it is not required for you to understand assembly at a detailed level or to be able to write assembly code, you need to have a clear understanding of what is going on even in the assembly language portions of the kernel.

5 Reference material

During the work on the task you will use several tools. You might find the following manuals helpful:

- Bochs http://bochs.sourceforge.net/doc/docbook/

  Bochs is a program that simulates a PC. We will use Bochs for running the operating systems we are working with. Bochs has a built in assembly language debugger which is very useful when tracking down difficult bugs.

- GDB
  GDB is a source level debugger which you can use for debugging purposes. Usage of GDB will be demonstated in the databar.

- GNU Binutils http://www.gnu.org/software/binutils/

  This is a large collection of tools for manipulating different executable binary files. We are primarily using as and ld:

  as: http://sourceware.org/binutils/docs/as/index.html

  ld: http://sourceware.org/binutils/docs/ld/index.html

  Please note that Binutils refers to the AMD64 architecture as x86_64.

- GCC http://gcc.gnu.org/

  GCC is a collection of compilers. In the course, you use version 4.7.1 of the C compiler for the AMD64 architecture. GCC, like Binutils refers to the AMD64 architecture as x86_64.

- Gnu make http://www.gnu.org/software/make/manual/

  make is a tool that can help you coordinate compilations of larger programs.

- GRUB http://www.gnu.org/software/grub/

  GRUB is a boot loader. A boot loader is a program that loads an operating system kernel. You will use GRUB to load the kernels you write.

- Eclipse http://eclipse.org

  Eclipse is a IDE (Integrated Development Environment) that you most likely have seen or used before. It is primarily used to develop Java applications, but can also via the CDT plugin be used to develop C – as in this course.

Finally, you can find the AMD64 reference books on http://www.amd.com/. Look for "AMD64 Architecture Tech Docs".

6 Getting started

The following assumes that you have already installed VirtualBox and the development image supplied by the course staff. For further details see the document " VirtualBox_Install_Guide.pdf" available on CampusNet.

1. Download the file "source_code_skeleton_archive_file_for_task_B1.tar.gz" from CampusNet within the virtual machine. This archive file contains all source code files for the kernel you will study and change when working with this task.
2. On the desktop of the virtual machine you should see a folder named "Home" - open it
3. Right-click and create a folder named 02333
4. Open the "Download" folder
5. Double-click the file you just downloaded
6. Press Ctrl+e and point the dialog box to the 02333 folder you created
7. Open Eclipse and create a new project of the type "Makefile project with existing sources"
8. Point it to the 02333 folder, and find the folder with the src folder *in it* – not the src folder itself. It should be directly under the task1 folder
9. Finish the guide with the defaults

As an alternative to the above steps, you can do the same work directly from the command line if you are familiar with Unix command line tools and shells.

Congratulations - you now have a build environment! Now look through the sources located inside the src folder. You need to understand most of of them in detail!

Note: you need some additional work to get Eclipse to build and run the sources, so for the time being, we will be using the Linux terminal to build and run.

7 Source code tree

But first, take some time to navigate the source tree and sift through the source code. Don't be alarmed, operating system code can be hard to understand. It is not expected of you to understand the assembly code files, boot32.s, boot64.s, enter.s and relocate.s at a detailed level.

However, they are documented so that you can follow their overall structure. You need to undertand what is going on in them.

In the tarball you will find several folders and files. Here is a very brief introduction:

There should be two folders, one named bochs, and one named databar_assignments

The bochs folder holds four files:

- bochsrc and bochsrc.gdb which are configuration files for bochs
- grub.cfg which is a configuration file for the GRUB boot loader.
- Makefile.mk which is a makefile that contains rules for building the boot image

The databar_assignments folder contains a folder named task_B1 which holds the skeleton code archive.

- The doc folder holds Doxygen generated documentation. Have a look at it!
- After the first build a folder called objects will be created. It holds compiled code.
- The src folder holds all source code files:
  - boot32.s which holds assembly code that puts the CPU into 64-bit mode. This file is not easy to understand. Use the comments.
  - boot64.s which initializes some important CPU registers, initializes and starts the operating system. Like boot32.s, this file is not easy to understand.
  - enter.s which holds assembly code that initialize the kernel. It also contains the low-level assembly code that handles system calls. Like boot32.s and boot64.s, this file is not easy to understand.
  - kernel.c which is the main kernel source code and holds high-level initialization and system call code.
  - syscall.c which holds the implementations of the system calls.
  - kernel.h which holds various C and C pre-processor definitions and declarations.
  - link32.ld which is a script used by the ld link editor to generate the 32bit portion of the kernel.
  - link64.ld which is a script used by the ld link editor to generate the 64bit portion of the kernel.
  - relocate.s contains assembler code for calling the entry code of the 64bit portion of the kernel.
- The Makefile file is a file that the make tool use to orchestrate the compilation of the kernel. The make tool will also, if the compilation is successful, invoke the Bochs emulator.
- Doxyfile is configuration file for a automated documentation system called “Doxygen”

You should now draw a sketch of the organization of the system. Relate the organization of the system to the different types of operation systems discussed in the text book by Tanenbaum. What kind of operating system is it?

Which portion execute in kernel mode? Which portions in user mode? What is the difference between user and kernel mode?

The kernel supports a single program running in user space. How is the kernel invoked from the user-level program? How is control passed back to the user-level program?

All of these questions should be answered in the midterm report. The course staff will not provide answers.

8 Testing the environment

To test the environment, start a terminal. It is located in the “start” menu. Go to the 02333 folder with a variant of the following command:

```
cd 02333/databar_assignments/task_B1
```

Now you can invoke make with the following command:

```
make
```

The make tool will now, if your environment is working, compile the kernel and start bochs.

Bochs will now open a new window. This window is used by Bochs to show the simulated PC's screen.

In your original terminal window you will soon see something similar to the following text:

```
00000000000i[XGUI ] [x] Mouse off
00000000000i[     ] set SIGINT handler to
bx_debug_ctrlc_handler
Next at t=0
(0) [0xfffffff0] f000:fff0 (unk. ctxt): jmp far
f000:e05b         ; ea5be000f0
<bochs:1>
```

This means that Bochs has finished initializing the simulated PC and is waiting for you.

Type c and return to start the execution of the kernel.

Eventually, you will see something like the following message:

```


The kernel has booted!

```

```
Hello World!
fffffffffffffffe
request return to dbg prompt received, 0x8AE0 command
(iodebug)
Next at t=28093971
(0) [0x00100560] 0018:0000000000100560 (unk. ctxt): mov
qword ptr ds:[rip+0x0000000000019cb5],
0x0000000000000000 ; 48c705b59c010000000000
<bochs:2>
```

This means that your environment is working.

Type q, return and then return again to exit the emulator.

Have a look at the Doxygen documentation. You can find an index file under the doc folder in the html folder. When opened in a browser, you will be provided with a navigatable collection of HTML pages representing your code and corresponding comments. You can recreate the HTML files at any point. This is sometimes useful when editing source code.

The pre-supplied configuration makes it very easy. Just run the command:

```
doxygen
```

from the task_B1 folder.

9 Implement a system call

To complete the task, you will have to implement a system call that returns the kernel version number to the user-level program.

Arguments and data is passed between the program and the kernel via registers. Each system call has a uniquely identifying number. This number is a positive integer. The program selects which system call to invoke by setting the rax register to the integer corresponding to the system call. All system calls may return a result value in the rax register.

Start with studying the syscall.c file. This file has three system call already implemented. Study how the system calls are implemented and how data is passed to and from the kernel. Hint: Each system call is implemented as a case statement. The SYSCALL_ARGUMENTS macro expands to a structure of type struct context. This type is defined in kernel.h.

You are going to implement the version system call. This system call has the number 0. There is a macro, SYSCALL_VERSION, in kernel.h that you can use instead of hard coding the number.

Add a new case statement in syscall.c and add the implementation of the version system call. Hint: The version is to be returned in the rax register. The kernel version is defined in the KERNEL_VERSION macro in kernel.h.

Hint: use make and follow the instructions in section 8 to test your implementation.

Hint: the command:

```
make clean
```

Will remove any files created when compiling the kernel.

10 Test cases

The skeleton code has a built-in test case. The program running on top of the kernel use the system call that you are to implement and then prints the value on the Bochs console. When the system call is not implemented the output is as follows:

```
Hello World!
fffffffffffffffe
```

When you have successfully implemented the system call, the second line will show the kernel version number in hexadecimal form.

11 Reflection

Before continuing with another task, take time and make sure you have answers to the following questions:

1. Draw a figure of the organization of the system. Relate the organization of the system to the different types of operation systems discussed in the text book by Tanenbaum. What kind of operating system is it? Why?
2. Which portion execute in kernel mode? Which portions in user mode? What is the difference between user and kernel mode?
3. How is the kernel invoked from the user-level program? Explain and elaborate!
4. How is control passed back to the user-level program? Explain and elaborate!

12 Feedback meeting

When you have completed the task and you are able to answer all questions in section 11, you can ask for a feedback meeting. To do that you need to prepare a mini-poster of at most one A3 page, i.e., two A4 pages, with a high level description of your design and implementation and answers to all the questions in section 11. You can then ask for a meeting during a databar session. All group members have to be present at the meeting. Your work will be reviewed for correctness and you will be asked for a brief oral presentation of your solution as well as possibly complementary questions. The purpose of the activity is for you to gain confidence as to what grade you can get on the reports.

13 Report

In the midterm report, document and describe your implementation and your experiences. You must also address the questions in section 11.

Please note that in the report, the contributions of each member of the group must be clear.

You write a single report which handed in at the end of the course. In this report, you document all the tasks you complete during the course.

Good Luck!