KUDOS

is the

Koebenhavns Universitet Discrete Operating System

The user-guide of the KUDOS system
Version 1.0
May 27, 2014

Juha Aatrokoski, Timo Lilja, Leena Salmela,
Teemu J. Takanen, Aleksi Virtanen and Philip Meulengracht

Contents

Chapter 1

Introduction

KUDOS is a derivative project from BUENOS which aside from supporting the old BUENOS system, now also supports the Intel x86-64 Architecture. The new operating system, KUDOS, is meant also meant as a exercise base for the operating system course, but with a more common platform in mind. Unlike its old project BUENOS, KUDOS will run on more recent real-world hardware[1] without any special architecture, but rather on your own laptop at home.

The system is kept structurally intact to BUENOS, however all platform-specific code has been split up into sub-directories, and is controlled with the makefile. KUDOS systems are ready for multi-core, however support for starting other application processors has not been implemented, while the BUENOS part has the multi-core support. Both BUENOS and KUDOS also provides skeleton code for threading, wide variety of synchronization primitives, userland support and proccesses.A simple custom filesystem and code for networking is also provided (However no network-card drivers are provided).

To keep both 32 bit code (the mips project) and 64 bit code (the x86-64 project) under the same roof, many modifications had to be made to the old BUENOS project, and thus the structure of the project has changed partially. It has also changed the procedure for starting up the new operating system, KUDOS.

Just like BUENOS, the main idea of the system is to give you a real, working multiprocessor operating system kernel which is as small and simple as possible. To boot KUDOS on a real computer all you would need is simply GRUB2 as a bootloader, and then make sure to add KUDOS as a boot entry into GRUB2. KUDOS now supports your everyday intel 64 bit architecture and thus can run on your everyday computer. No code modifcations would be neccessary.

1.1 Tools Needed

If you are a student participating on an operating systems project course, the course staff has probably already set up a development environment for you. If they have not, look below to get the appropriate tools needed.

1.1.1 Tools needed for the MIPS Kernel

To run it on a virtual machine:

1. A virtual machine (VirtualBox, Bochs or QEmu) with linux

2. The KUDOS source code

[1] In theory.

3. A Mips-32 cross compiler
4. YAMS (see below for details)

1.1.2 Tools needed for the x86-64 Kernel

To run the x86-64 kernel you have two options, you can either run it on a virtual machine (you should get a premade distrubuted disk image which contains both a kudos boot and a linux boot from your teacher), or you can run it directly on your own computer

To run it on a virtual machine:

1. A virtual machine (VirtualBox, Bochs or QEmu)
2. A linux distribution with GRUB2 on the virtual machine
3. The KUDOS source code
4. A x86-64 cross compiler

To run it on your own computer:

1. The KUDOS source code
2. A x86-64 cross compiler
3. A computer with GRUB2
4. A small raw partition that can be used for tfs filesystem.
5. You need knowledge of how to add new boot-entries into GRUB, or learn so from google.

1.2 Expected Background Knowledge

Since the KUDOS system is written using the C programming language, you should be able to program in C. For an introduction on C programming, see the classical reference [K&R]. You also need to know quite much about programming in general, particularly about procedural programming.

We also expect that you have taken a lecture course on operating systems or otherwise know the basics about operating systems. You can still find OS textbooks very handy when doing the exercises. We recommend that you acquire a book by Stallings [Stallings] or Tanenbaum [Tanenbaum].

Since you are going to interact directly with the hardware quite a lot, you should know something about hardware. A good introduction on this can be found in the book [Patterson].

Since kernel programming generally involves a lot of synchronization issues a course on concurrent programming is recommended. One good book on this field is the book by Andrews [Andrews]. These issues are also handled in the operating systems books by Stallings and Tanenbaum, but the approach is different.

1.3 KUDOS for teachers

As stated above, the KUDOS system is meant as an assignment backbone for operating systems project courses. This document, and the roadmap, while primarily acting as reference guide to the system, is also designed to support project courses. The document is ordered so that various kernel programming issues are introduced in sensible order and exercises (see also section 1.4) are provided for each subject area.

While the system as such can be used as a base for a large variety of assignments, this document works best if assignments are divided into five different parts as follows:

1. **Synchronization and Multiprogramming.** Various multiprogramming issues relevant on both multiprocessor and uniprocessor machines are covered in the threading chatper and the synchroniation chapter.
2. **Userland**. Userland processes, interactions between kernel and userland as well as system calls are covered in the userland chapter.
3. **Virtual Memory**. The current virtual memory support mechanisms in KUDOS are explained in the virtual memory chapter, which also gives exercises on the subject area.
4. **Filesystem**. Filesystem issues are covered in the filesystems chapter.
5. **Networking**. Networking in KUDOS is explained in the networking chapter, but note that the base system doesn't include a driver for the network interface. Thus it is recommended to provide one as a binary module for students or use the device interface as a part of this round and let students implement one.

1.3.1 Preparing for KUDOS Course

To implement an operating systems project course with KUDOS, the course administrators need to decide whether the mips or the x86-64 architecture will be used for testing, or both architectures will be tested. As a minimum the below items should be provided:

- Provide students with a development environment with precompiled YAMS, MIPS32 ELF cross compiler and x86-64 cross compiler and GRUB2 as bootloader. See YAMS usage guide for instructions on setup of YAMS and the cross compiler environment.
- Decide which exercises are used on the course, how many points they are worth and what are the deadlines.
- Decide any other practical issues (are design reviews compulsory for students, how many students there are per group, etc.)
- Familiarize the staff with KUDOS, GRUB2 and YAMS.
- Introduce KUDOS to the students.

1.4 Exercises

Each chapter in the roadmap contains a set of exercises. Some of these are meant as simple thought challenges and some as much more demanding and larger programming exercises.

The thought exercises are meant for self study and they can be used to check that the contents of the chapter were understood. The programming exercises are meant to be possible assignments on operating system project courses.

The exercises look like this:

1.1. This is a self study exercise.

⌨ **1.2.** This is a programming assignment. They are indicated with a bold exercise number and a keyboard symbol.

1.5 Contact Information

Latest versions of `KUDOS` can be found at the project homepage:

http://www.ku.dk/projects/kudos/

Latest versions of `YAMS` for the mips-kernel can be downloaded from the project home-page at:

http://www.niksula.hut.fi/u/buenos/

Authors from `KUDOS` can be contacted on mail lpz849@alumni.ku.dk. Currently there is no publicly available mailing list to subscribe, but one may be created if needed.

Authors from the old project (`BUENOS`) can be contacted through the mailing list `buenos@cs.hut.fi`, if you have questions about the old project.

Chapter 2

Using KUdos

2.1 Installation and Requirements

The KUDOS-mips system requires the following software to run:

- YAMS machine simulator, version 1.3.0 or above[1]
- GNU Binutils for mips-elf target
- GNU GCC cross-compiler for mips-elf target
- GNU Make

First you have to set up the YAMS machine simulator. From YAMS documentation you can find instructions on how to set up Binutils and GCC cross-compiler.

After the required software is installed installing KUDOS is straightforward: you simply extract the KUDOS distribution tar-file to some directory.

The KUDOS-x86-64 system requires the following software to run:

- GNU Binutils for x86_64-elf target
- GNU GCC cross-compiler for x86_64-elf target
- GNU Make
- GNU GRUB or GRUB2

First you have to set up a virtual machine with a linux distribution. Then you should acquire the source code of binutils and gcc, and cross-compile it for x86_64 as a target. For extra help it is possible to consult this webpage to help with the build-process of GCC

http://wiki.osdev.org/GCC_Cross-Compiler

After the required software is installed installing KUDOS is straightforward: you simply extract the KUDOS distribution tar-file to some directory and build it.

[1] A previous version of YAMS can also be used if the output format is set to "binary" in the linker script ld.script

2.2 Compilation

To compile KUDOS for MIPS, you want to invoke the command `make mips` in the main directory of KUDOS

To compile KUDOS for x86-64, you want to invoke the command `make x64` in the main directory of KUDOS

You can compile both MIPS and x86-64 systems by invoking `make` in the main directory of the BUENOS. After compiling the system, you should have a binary named `kudos_64` or `kudos_mips` in the main directory.

2.3 Booting the System

2.3.1 Booting the mips system

After the system has been properly built, you can start YAMS with KUDOS binary by invoking

```
yams kudos_mips
```

at the command prompt. If you want to give boot arguments to the system, see the section about boot arguments in the roadmap.

If you are using the default YAMS configuration that is shipped with KUDOS, you have to start the `yamst` terminal tool before invoking `yams`. The terminal tool provides the other end-point of the `yams` terminal simulation. To start `yamst`:

```
yamst -lu tty0.socket
```

in another terminal (e.g. in another XTerm window).

2.3.2 Booting the x86-64 system

After the system has been properly built, you need to update the onboard installation of KUDOS, which is simply a fat32 partition on the virtual machine that contains the KUDOS kernel in the path /boot/kUdOS. A make-command has been provided to make this process easier, you can update the KUDOS installation by simply invoking `make install` in the root directory of KUDOS after a successful build.

The command simply mounts the fat32 partition, and copies over the local image file (kudos_64 kernel image) to the fat32 partition and overwrites the old image.

Then to run the new kernel build, you must restart your virtual machine and choose "kUdOS Live Build" in the GRUB2 boot menu. If you wish to pass boot arguments to KUDOS, you may want to only use the arrow buttons to select KUDOS, and then press 'e'.

2.4 Compiling Userland Programs

2.4.1 Compiling

Userland programs are compiled using the same cross-compiler that is used for compiling KUDOS. To compile userland binaries go to the userland directory `tests/` and invoke `make`. Every userland-program is compiled into two different targets, one for mips (ex. halt), and one for x86-64 (ex. halt64). All userland programs for the x86-64 architecture gets the extension '64'.

2.4.2 Installing - mips

To run compiled programs they need to be copied to a YAMS disk, where KUDOS can find them. TFS-filesystem (see filesystem chapter) is implemented and a tool (see the TFS-tool chapter) is provided to copy binaries from host filesystem to KUDOS filesystem.

2.4.3 Installing - x86-64

To run compiled programs they need to be copied to a image created by the tfs-tool in the `utils/` directory. Afterwards, this image now needs to be written to disk, so KUDOS can find the find this particular tfs image.

A makefile command has been provided to simplify this procedure, you can copy the image file created by `tfstool` in the `utils/` directory to the KUDOS main directory, and then invoke the command `make copy IMG=<image>` This will install the image file you provide onto the pre-defined empty partition on disk using the dd linux command (disk write/read utility). It will override an old installation of the same image, so remember that if you had previously copied a file, it wont exist after you have written a new image using the `make copy <image>`

2.5 Using the Makefiles

KUDOS has two makefiles that are used to build the binaries needed by KUDOS. The system makefile builds the KUDOS binaries and the submission archive needed to submit the exercises for reviewing. This makefile is in the KUDOS main directory and is called `Makefile`. The other makefile is the makefile responsible for building the userland binaries. This makefile is in the `tests/` directory.

2.5.1 System Makefile

KUDOS uses somewhat unorthodox monolithic makefile. The system is based on Peter Miller's paper [Miller]. KUDOS is divided to modules that correspond to the directory structure of the source code tree (see kernel overview chapter).

The files in the module directories are built to KUDOS binaries. These module directories have a file called *module.mk* that contains the name of the module and list of the files included from this module. So, for example, the `module.mk` in the lib directory:

```
# Makefile for the lib module

# Set the module name
MODULE := lib

FILES := libc.c xprintf.c rand.S bitmap.c debug.c

SRC += $(patsubst %, $(MODULE)/%, $(FILES))
```

If you add files to your system, you have to modify only the FILES variable. There should be no need to change anything else.

The KUDOS makefile system also contains some platform-specific modules now aswell, which means if you wanted to extend one of the platform-specific builds, we can take a look at a platform specific module. For example, the `module.mk` in the init directory:

```
# Makefile for the init module

# Set the module name
MODULE := init/x86\_64

FILES := _boot.S main.c

X64SRC += $(patsubst %, $(MODULE)/%, $(FILES))
```

Here it is easy to see, that instead of adding files to the ”SRC” variable, we add the specifically to the ”X64SRC”, which means they are only compiled for the x86-64 target.

The main makefile is in the main directory and named `Makefile`. There are few features in the Makefile that you have to be aware of. In the unlikely event that you wish to add a new module (directory) to the system, you have to modify the `MODULES` variable by extending it with the module name. Remember that this name must be same as the directory where the module is. When you do your exercises, you have to wrap them with CHANGED_*n* C-Preprocessor variables. You can define these variables by modifying the `CHANGEDFLAGS` variable. The variable `IGNOREDREGEX` is used when you build your submission archive on returning your assignment. The variable contains a regular expression pattern with which the matching files are filtered out from the actual submission archive.

The following targets can be built by the system makefile:

`all`

The default, builds both the `KUDOS`-mips binary, the `KUDOS`-x86_64 binary and the `tfstool`.

`mips`

Builds both the `KUDOS`-mips binary and the `tfstool`.

`x64`

Builds both the `KUDOS`-x86_64 binary and the `tfstool`.

`install`

Updates the onboard installation of the `KUDOS`-x86_64 binary. The actual steps performed is: mount fat32 partition (pre-setup) copy local kudos64-binary to fat32 partition, overwrite old umount fat32 partition

`copy IMG=<image>`

Writes the given image file to the onboard raw partition (also pre-setup), so that `KUDOS` can locate this partition and use it as filesystem. The actual steps performed is: dd if=IMG of=RAWPARTITION It just writes the image-file to the start of the raw partition using DD.

`util/tfstool`

Build the tfstool utility.

`clean`

Clean the compilation files.

`real-clean`

Clean also the depedency files.

`submit-archive PHASE=`*n*

Builds `submit-`*n*`.tar.gz` in the parent directory of the main buenos directory. The variable n indicates the submission round number (default is 1).

2.5.2 Userland Makefile

To build userland binaries go to the `tests/` subdirectory and invoke `make`. There are no special targets and the makefile is organised so that every binary is built. If you wish to add your own binaries to the makefile, add your source files to the `SOURCES` variable at the beginning of the makefile.

2.6 Using Trivial Filesystem

For easy testing of `KUDOS`, some method is needed to transfer files to the filesystem in `KUDOS`. The Unix based utility program, `tfstool`, which is shipped with `KUDOS`, achieves this goal. `tfstool` can be used to create a Trivial Filesystem (TFS, see **??** for more information about TFS) to a given file, to examine the contents of a file containing TFS and to transfer files to the TFS. `KUDOS` implementation of TFS does not include a way to initialize the filesystem so using `tfstool` is the only way to create a new TFS. `tfstool` is also used to move userland binaries to TFS. When you write your own filesystem to `KUDOS`, you might find it helpful to leave TFS intact. This way you can still use `tfstool` to transfer files to the `KUDOS` system without writing another utility for your own filesystem.

The implementation of the `tfstool` is provided in the `util/` directory. The `KUDOS` makefile can be used to compile it. Note that `tfstool` is compiled with the native compiler, not the cross-compiler used to compile `KUDOS`. The implementation takes care of byte-order conversion if needed.

To get a summary of the arguments that `tfstool` accepts you may run it without arguments. The accepted commands are also explained below:

`create` ***filename size volume-name***

Create a new TFS to file *filename*. The total size of the file system will be *size* 512-byte blocks. Note that the three first blocks are needed for the TFS header, the master directory and the block allocation table and therefore the minimum size for the disk is 3. The created volume will have the name *volume-name*.

Note that the number of blocks must be the same as the setting in `yams.conf`

`list` ***filename***

List the files found in the TFS residing in *filename*.

`write` ***filename local-filename*** `[`***TFS-filename***`]`

Write a file from the local system (*local-filename*) to the TFS residing in the file *filename*. The optional fourth argument specifies the filename in TFS. If not given, *local-filename* will be used.

Note that you probably want to give a TFS filename, since otherwise you end up with a TFS volume with files named like `tests/foobar`, which can cause confusion since TFS does not support directories.

`read` ***filename TFS-filename*** `[`***local-filename***`]`

Read a file (*TFS-filename*) from TFS residing in the file *filename* to the local system. The optional fourth argument specifies the filename in the local system. If not given, the *TFS-filename* will be used.

`delete` ***filename TFS-filename***

Delete the file with name *TFS-filename* from the TFS residing in the file *filename*.

2.7 Starting Processes

To start a userland process in `KUDOS` you have to

1. have kudos kernel binary (compile if it doesn't already exist).
2. have the userland binary (compile if it doesn't exist).
3. have a filesystem disk image (use `tfstool` to create this).
4. copy the userland binary with `tfstool` to the file system image.
5. write the file system image using `make copy IMG=image` if needed
6. boot the system with proper boot parameters (see boot arguments section).

`KUDOS` is shipped with simple userland binary `halt` which invokes the only already implemented system call `halt`. Here is an example of how to compile `KUDOS`-mips, install the userland binary and boot the system.

```
cd kudos
make mips
make -C tests/
util/tfstool create store.file 2048 disk1
util/tfstool write store.file tests/halt halt
yamst -lu tty0.socket # (in another window, socket is in the main dir)
yams kudos-mips 'initprog=[disk1]halt'
```

After running the above commands the `KUDOS` output should go to the window where you started `yamst`. The `halt` program merely shutdowns the system, thus `YAMS` should exit with the message `"YAMS running...Shutting down by software request"`.

Bibliography

[Andrews] Andrews, G. R., *Foundations of multithreaded, parallel and distributed programming*, ISBN 0-201-35752-6, Addison-Wesley Longman, 2000

[Patterson] Patterson, D. A., *Computer organization and design: the hardware/software interface*, ISBN 1-55860-491-X, Morgan Kaufmann Publishers, 1998

[Stallings] Stallings, W., *Operating Systems: Internals and Design Principles*, 4th edition, ISBN 0-13-032986-X, Prentice-Hall, 2001

[K&R] Kernighan B. W., Ritchie D. M., *The C Programming Language*, 2nd Edition, ISBN 0-13-110362-8, Prentice-Hall, 1988

[Tanenbaum] Tanenbaum, A. S., *Modern Operating Systems*, 2nd edition, ISBN 0-13-031358-0, Prentice-Hall, 2001

[Miller] Miller, Peter, *Recursive Make Considered Harmful*, `http://www.tip.net.au/~Emillerp/rmch/recu-make-cons-harm.html`

Index