University of Karlsruhe
System Architecture Group
Gerd Liefländer

Teaching Assistant:
Philipp Kupferschmied

System Architecture 2008/09
Programming Assignment - Practical Part 0 Update 2

Submission Deadline: 14.11.2008, 12:00h

This assignment is intended to guide you through the basic steps for setting up your development environment, and to become familiar with the tools you will use. Before you start with the steps described below, you should have found a partner with whom you work.

In order to fulfill the assignments, you will have to set up a Mercurial repository. Mercurial is a distributed source management tool, where each developer can work on his/her own, local repository. However, as you will also need to merge the work of each group member, we will set up a "store" to which both group members have read and write access. In the following, we will give a detailed explanation how to set up a directory in one user's `$HOME` folder that is accessible by both team members in a secure way. All Mercurial repositories (one for each programming assignment) are going to be created within this shared directory.

We will explain how to set up your development environment when each of you will work only with his/her ATIS account (no matter if directly in the ATIS pool or remotely via SSH). We will also give some additional hints in case you want to work locally on your own computer.

In either case, we assume `bash` as your shell (that's the default shell on the ATIS machines). To set an environment variable, you need to use `export VAR=''value''`. If you need more information use the man pages (`man bash`).

In the remainder of this document, several `cd` shell commands intentionally use absolute paths rather than shorter, relative ones; some are even completely redundant. The chosen form tries to minimize potential errors resulting from executing commands in the wrong places—feel free to replace/omit them.

In several places, `$HOME` is used instead of "~"; please keep this notation, as `$HOME` "always" expands to an absolute path required e.g. by configure tools, whereas ~ does not.

1 Setting up your Development Environment

We start with setting up a shared directory for the Mercurial repositories, to which each of you will have read and write access. This directory will be hosted on the ATIS account of one of you, to whom we will refer as *s_hghost* [1] in the following. To the other team member, we will refer to as *s_member*. **Whenever *s_hghost* or *s_member* appear in text that must be entered in the instructions below, you have to substitute it with the correct ATIS account name of the respective user.** For this initial setup process, both of you should meet in the ATIS.

1.1 Settting up the Mercurial Configuration

Mercurial can be configured with different options specified in the file `~/.hgrc`. We will set only two: the user name that will appear in the commit history, and the editor that shall be used for entering commit messages. To do so, **both** team members have to enter the following commands:

```
$ cd ~
$ cat >> .hgrc

[ui]
```

[1] *Hg* is the chemical symbol of mercury. All Mercurial commands start with *hg*.

```
username = Your name
editor = Your favorite editor
```

Press Ctrl + d

As username, you should use your real name, perhaps together with your email address. It can for example look like this: “Philipp Kupferschmied ⟨pk@ibds.uka.de⟩” As an editor, you can specify whichsoever you like (as long as it is installed on the machine). If you are unsure which one to choose and not familiar with editors like *vi*, *vim*, or *emacs*, we suggest to use *nano*.

1.2 Setting up the “Shared Directory”

To gain access to the shared directory in *s_hghost*’s account, *s_member* needs to create an SSH key. Remote access via that key will be restricted so that *s_member* can only execute Mercurial commands within the shared directory.
This description relies on the OpenSSH client (installed in the ATIS pool, see `ssh -V`); other SSH implementation may require different paths (e.g. `~/.ssh2/`) or different arguments.

1.2.1 Preparing *s_member*’s Account

s_member has to create a key on his/her account, associate it with the `sysarch_hg` hostalias, and copy the public part into *s_hghost*’s account via:

```
$ mkdir -p ~/.ssh
$ cd ~/.ssh
$ ssh-keygen -t dsa -P "" -f sysarch_s_member
$ cat >> config

Host sysarch_hg
Hostname i08fs1.ira.uka.de
IdentityFile ~/.ssh/sysarch_s_member
PubkeyAuthentication yes
User s_hghost
```

Press Ctrl + d

```
$ scp sysarch_s_member.pub s_hghost@i08fs1.ira.uka.de:.ssh/
```

Make sure that you did not mistype the `PubkeyAuthentication` command, it is really *Authentication*, **not** *Authentification*. The last command will require *s_hghost* to enter his/her account password.

1.2.2 Preparing *s_hghost*’s Account

Afterwards, *s_hghost* must allow *s_member* to access the shared directory on his/her behalf, but must also ensure that *s_member* is not allowed to do anything else but running Mercurial commands inside this directory. To this end, **only** *s_hghost* executes the following commands:

```
atis$ mkdir -p ~/.ssh
atis$ cd ~/.ssh
atis$ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/hg-ssh
atis$ chmod +x hg-ssh
atis$ ( echo -n 'command="cd /home/stud/s_hghost/sysarch/sharedrepos &&
/home/stud/s_hghost/.ssh/hg-ssh *",no-port-forwarding,no-X11-forwarding,
no-agent-forwarding,no-pty '; cat sysarch_s_member.pub; ) >> authorized_keys
```

Note 1: There must be no whitespace in the `echo` command except for those in the string between the double quotes and the one after `no-pty` (which are mandatory)!

Note 2: The `echo` command should be entered as a single line, it is broken into three here due to space limitations only.
Note 3: Repeat the `echo` command for each SSH key which shall have access to the shared directory. For example, if you want to work locally on your computer at home, create a new SSH key on that machine, copy the public part of the key to `~/.ssh` on *s_hghost*'s account, and repeat the `echo` command, where you replace `sysarch_s_member.pub` with the filename of the newly copied public key.

This setup restricts remote access to the directory `~/sysarch/sharedrepos` (which will be created later on) on *s_hghost*'s account. *s_member* is only allowed to execute the `hg-ssh` script inside this directory. This script is a wrapper around the actual Mercurial functions and performs some additional checks.

1.3 Installing the Cross-Compiler Toolchain

In the programming assignments, you will develop an operating system for the MIPS architecture. As most of you will not develop on a MIPS machine, you need to install cross-compiler tools that can generate code for MIPS.

1.3.1 ATIS account

On ATIS pool computers, these tools are already installed. To save space, do not install them again... You only need to add the tools to your PATH via:

```
atis$ cat >> ~/.bashrc

export PATH=/opt/cs161/bin:$PATH
```

Press Ctrl + d

If you do not work in the ATIS pool, but log in remotely via SSH, you have to modify the file */.bash_login* (if it exists) in the same way.

1.3.2 Home Accounts

On each development machine you will need around 500 MB of free disk space in order to build and install the cross-compiler toolchain. First, you will need to fetch the source archives. In order to build the toolchain, you need to download, build, and install binutils, gcc and gdb—in that order. All these tools build at least on x86 under Linux, *BSD, and possibly MacOS X.
Note: You must not have "." in your `$PATH` prior to "/bin"; otherwise the build will fail with obscure errors. For security reasons, adding "." to `$PATH` is a bad idea anyway.
The following steps need to be taken for every development machine—i.e. every computer you would like to work on—**except ATIS machines**, as the tools are already installed there (see `/opt/cs161/bin`).

1. The easy way

   For your convenience, there is a script available from our web sites which downloads, builds, and installs the whole toolchain automatically. Simply type:

   ```
   $ mkdir -p ~/sysarch
   $ cd ~/sysarch
   $ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/cs161-install.sh
   $ bash cs161-install.sh
   ```

   The script will also update your `~/.bashrc`, adding `$HOME/sysarch/bin` to your PATH for easy use of the tools.

   To be able to access the remote Mercurial repository, update your `~/.ssh/config` file as described above for the ATIS accounts. You either have to copy the key you created at the ATIS to your local machine, or you can create a new one.

2. The harder way

   If something goes wrong with the script or if you want to keep full control over the installation process, proceed along the following steps:

   (a) Download the sources:

   ```
   $ mkdir -p ~/sysarch
   $ cd ~/sysarch
   $ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/cs161-binutils-1.4.tbz2
   $ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/cs161-gcc-1.4.tbz2
   $ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/cs161-gdb-1.4.tbz2
   ```

   (b) Unpack the archives:

   ```
   $ tar -xjf cs161-binutils-1.4.tbz2
   $ tar -xjf cs161-gcc-1.4.tbz2
   $ tar -xjf cs161-gdb-1.4.tbz2
   ```

   (c) Build and install binutils, gcc and gdb:

   ```
   $ cd cs161-binutils-1.4
   $ ./toolbuild.sh --cs161dir="$HOME/sysarch"
   $ cd ..
   $ cd cs161-gcc-1.4
   $ ./toolbuild.sh --cs161dir="$HOME/sysarch"
   $ cd ..
   $ cd cs161-gdb-1.4
   $ ./toolbuild.sh --cs161dir="$HOME/sysarch"
   $ cd ..
   ```

   (d) Set up environment variables:

   ```
   $ cat >> ~/.bashrc

   export PATH="$HOME/sysarch/bin":$PATH
   ```

   Press Ctrl + d
   You will also have to adapt your `~/.ssh/config`, as described above.

If you work as `root`—which is a **really** bad idea!—you might need to type `source ~/.bashrc` every time you open a shell.

1.4 Installing System/161

The MIPS simulator, System/161, is required to execute your operating system kernel. It must be installed on all machines that you want to use for development—including your ATIS accounts.

```
$ mkdir -p ~/sysarch
$ cd ~/sysarch
$ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/sys161-1.13.tbz2
$ tar -xjf sys161-1.13.tbz2
$ cd sys161-1.13
$ ./configure --installdir="$HOME/sysarch/root" mipseb
$ make && make install
$ cd ~/sysarch/root
$ cp sys161.conf.sample sys161.conf
```

2 First Steps With OS/161

You will be using Mercurial to manage the OS/161 sources of your group. Mercurial documentation can be found at `http://www.selenic.com/mercurial`. If you have just set up your environment, log out and back in to activate the changes in your `~/.bashrc`.

2.1 Import the Supplied Sources

s_hghost now has to download the sources of assignment 0 and create a new mercurial repostory inside the shared directory `~/sysarch/sharedrepos`. The following steps **must not** be performed more than once per assignment and group!
First download and unpack the OS/161 sources via:

```
$ mkdir -p ~/sysarch/sharedrepos/asst0-src
$ cd ~/sysarch/sharedrepos/asst0-src
$ hg init
$ cd ~/sysarch/
$ hg clone sharedrepos/asst0-src
$ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/asst0-src.tbz2
$ tar -xjf asst0-src.tbz2
$ rm asst0-src.tbz2
$ cd ~/sysarch/asst0-src
$ hg add
$ hg commit
$ hg tag asst-base
$ hg push
```

These commands create a new, **empty** repository in `~/sysarch/sharedrepos/asst0-src`. *s_hghost* then locally clones the repository in `~/sysarch/asst0-src`, extracts the downloaded tarball into this cloned (and still empty) repository, adds the files to the repository, commits these changes, and gives this initial revision a tag. Then, he/she pushes the newly created changeset back to `~/sysarch/sharedrepos/asst0-src`. In this way, the shared repository does not contain a working directory but only the store (or, in other words, no files are checked out). This helps to ensure that *s_hghost* does not accidentally work on the shared repository, but on his/her local clone in `~/sysarch/asst0-src` instead.
`hg commit` will open the editor you specified in your `~./hgrc` so that you can enter a commit message. A commit message should contain a summary of what you changed since the last commit. Since this is the first commit on this repository, enter something like “Initial commit”, save the file, and close the editor. `hg tag` gives the inital revision a name.

2.2 Obtain a Working Copy

As said before, Mercurial is a distributed source management software. Thus, each of you will clone the entire repository and subsequently work on this local clone. Committed changes will only affect the local copy. In order to sync between clones, Mercurial provides the commands `hg push` and `hg pull`. Details on how to use these commands will follow below. *s_member* has to clone the repository with the following commands

```
$ mkdir -p ~/sysarch
$ cd ~/sysarch
$ hg clone ssh://sysarch_hg/asst0-src
$ cd asst0-src
```

In contrast, *s_hghost* has already cloned the repository during the setup process.
After you have completed these steps, both *s_hghost* and *s_member* should have a directory `~/sysarch/asst0-src` to work on.

2.3 Build an OS/161 Kernel

To the business end of this assignment: You will now build and install your first OS/161 kernel.

1. First you need to configure your source tree:

   ```
   $ cd ~/sysarch/asst0-src
   $ ./configure --ostree="$HOME/sysarch/root"
   ```

2. Then you can build and install the user-land part of OS/161:

   ```
   $ cd ~/sysarch/asst0-src
   $ make
   ```

   As you will not modify the user-land tools, these two steps are required only once per assignment (though you may perform them as often as you like).

3. The kernel itself must be configured separately:

   ```
   $ cd ~/sysarch/asst0-src/kern/conf
   $ ./config ASST0
   ```

4. The last step is to build and install the kernel:

   ```
   $ cd ~/sysarch/asst0-src/kern/compile/ASST0
   $ make
   ```

2.4 Execute Your Kernel

If you have made it this far, you have built and installed the entire OS! Now, let's see how to run it...

```
$ cd ~/sysarch/root
$ ./sys161 kernel
```

This executes the MIPS simulator System/161 and lets it execute your OS kernel. To power off the simulated machine, type `q` at the menu prompt.

3 Using GDB

We cannot stress strongly enough to you the need to learn to use GDB. You can find short tutorial on using GDB with OS/161 at `http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/gdbtut.html`. Please work through this tutorial to become accustomed to using GDB together with System/161. Finish the GDB tutorial before proceeding. At least, create or update your `~/sysarch/root/.gdbinit` with the following contents:

```
target remote unix:.sockets/gdb
dir ../asst0-src/kern/compile/ASST0
set print array on
set print pretty on
b panic
```

4 Modifying Your Kernel

We will now go through the steps required to modify and rebuild your kernel. We will add a new file to the sources. The file contains a function we will call from existing code. We need to add the file to the kernel configuration, re-config the kernel, and then rebuild it.

1. Download `hello.c` and store it in `~/sysarch/asst0-src/kern/main/` via

   ```
   $ cd ~/sysarch/asst0-src/kern/main
   $ wget http://i30www.ira.uka.de/~pkupfer/edu/2008ws/sysarch/hello.c
   ```

2. Find an appropriate place the in the kernel code and add a call to `complex_hello()` (defined in `hello.c`) to print out a greeting. The message should appear immediately before the prompt.

   (Hint: One of the files in `kern/main/` is *very* appropriate.)

3. As we added a new file to the kernel code, we need to add it to the kernel configuration in order to build it. Edit `kern/conf/conf.kern` appropriately to include `hello.c`.

4. Whenever you change the kernel config, you need to re-configure the kernel:

   ```
   $ cd ~/sysarch/asst0-src/kern/conf
   $ ./config ASST0
   ```

5. Now rebuild the kernel.

   ```
   $ cd ~/sysarch/asst0-src/kern/compile/ASST0
   $ make
   ```

6. Run your kernel as before—note that the kernel will panic with an error message.

7. Use GDB to locate the error. Probably a combination of `display`, `break`, and `step` commands will come in handy...

   Use `./sys161 -w kernel` to start System/161, so that it immediately breaks and waits for a debugger to connect.

   Now open a new terminal, change to the directory `~/sysarch/root` and start `cs161-gdb kernel`. Then use GDB's `continue` command (or `c`) to release your kernel.

8. Fix the bug in an editor of your choice, recompile, and run the kernel again to see the welcome message.

   You need not reconfigure the kernel tree, as you did not add or remove files from the configuration; a simple `make` in `kern/compile/ASST0` should do the trick.

5 Practicing Mercurial

Mercurial can be used in differeny ways, none of which being the only "right" way. For the programming assignments, we suggest using it in a CVS-like manner, as we think this eases development and avoids conflicts. Of course, you can use Mercurial in every way you want, the Mercurial website documents various other working practices.
Mercurial organizes projects in repositories. A repository consists of a working directory and a store. The store contains the complete history of a project, the working directory contains all files of the project at a specific point in time. There are basically two ways to get a local Mercurial repository: The first is to create a new one with `hg init`, the second is to clone an already existing repository via `hg clone`. After having created/obtained a local repository, you can start editing files in its working

directory. Editing files does not affect the store unless you commit your changes, using `hg commit`. Commiting records the state of the working directory as a new revision. Note that commiting only affects your local repository, not the repository you might have cloned it from. To synchronize between different repositories, Mercurial offers `hg pull` and `hg push`. As the names suggest, `hg pull` pulls changesets from another repository to the local repository, whereas `hg push` pushes local changesets to another repository. The Mercurial website offers more detailed documentation about the usage of Mercurial. A good starting point is http://www.selenic.com/mercurial/wiki/index.cgi/UnderstandingMercurial Now we will perform some operations with Mercurial.

1. Let's check what changes you have made to your sources so far:

   ```
   $ cd ~/sysarch/asst0-src
   $ hg diff | less
   ```

   You should see a 'diff' of all the changes you made to your source tree.

2. Note that the new file `hello.c` you added was not included in the diff output. This is because you have not told Mercurial to include it in the repository. Do this now by executing the following commands:

   ```
   $ cd ~/sysarch/asst0-src/kern/main
   $ hg add hello.c
   ```

   Note that `hg add` only marks the file for being added with the next commit, rather then adding it immediately.

3. As your kernel has now reached a small milestone, it makes sense to commit your changes to the repository to publish and preserve this state.

   ```
   $ cd ~/sysarch/asst0-src
   $ hg commit
   ```

   Again, you will have to enter a (meaningful) commit message.

   If you do not like being asked to type a log entry into an editor each time, you can simply append `-m ''<log message>''` after the commit argument to the Mercurial command: `hg commit -m ''log entry message''`.

   Note that this commit only affects your local clone of the repository. If the other group member shall see your changes, too, you have to push them to the "main" repository, hosted on *s_hghost*'s account. To do this, type the following:

   ```
   $ hg push
   ```

   In case someone else already pushed something to the shared repository (and thus created one or more revisions), Mercurial will not perform the push operation, unless you explicitly force it to do so. However, this is not a good idea. Instead, you should first pull the remote changesets to your repository, locally merge them with your local changes, and finally push back the changes. Execute the following commands in order to do this:

   ```
   $ hg pull
   $ hg merge
   $ hg push
   ```

   Note that you might have to resolve conflicts manually when you run `hg merge`. See the slides of the introduction lecture for details.

4. Now edit `kern/main/main.c`: Remove a large part of the file. Try to rebuild your kernel, which should fail.

   ```
   $ cd ~/sysarch/asst0-src/kern/compile/ASST0
   $ make
   ```

5. Restore the previous version of `main.c`:

   ```
   $ cd ~/sysarch/asst0-src/kern/main
   $ rm -f main.c
   $ hg update
   ```

   Check to see if your kernel builds again.

6 Submitting the Assignment

Once you have added `hello.c` to your kernel, fixed the bugs, and run the kernel to see the welcome message, you have completed the first assignment. When running your kernel using `sys161 kernel ''q''`, you should see output similar to

```
sys161: System/161 release 1.13, compiled Oct 12 2007 16:36:52
OS/161 base system version 1.10
Copyright (c) 2000, 2001, 2002, 2003
  President and Fellows of Harvard College. All rights reserved.
Put-your-group-name-here's system version 0 (ASST0 #3)
Cpu is MIPS r2000/r3000
344k physical memory available
Device probe...
lamebus0 (system main bus)
.
.
.
con0 at lser0
pseudorand0 (virtual)

Hello World!!!
OS/161 kernel: q
Shutting down.
The system is halted.
```

This is also generally the way we test your submission.
Now it is time to submit your solution. Submission requires some coordination with you partner to ensure that both of you have completed their work before it is submitted (this is especially true for the later assignments, which allow for more concurrent work). Before submitting, both members should locally commit their latest changes, and push those changes to the shared repository using `hg push`. Merge conflicts locally if necessary (using `hg pull` and `hg merge`, as described above). Decide on one of you to submit your solution (there is no need to send you tutor the same solution twice). Let us assume you chose *s_member*. *s_member* first ensures that his/her local repository is up-to-date by running

```
$ cd ~/sysarch/asst0-src
$ hg pull
$ hg update
```

Now *s_member* can tag the current revision to mark it as the final version

```
$ cd ~/sysarch/asst0-src
$ hg tag asst-final
$ hg push
```

Running `hg push` after having tagged the current revision ensures that the tag becomes visible in the shared repository, too. Finally, *s_member* must have a look at the changes you made to the source originally provided:

```
$ cd ~/sysarch/asst0-src
$ hg diff -r asst-base -r asst-final > ~/asst0.diff
```

Send `~/asst0.diff` to `os161@ira.uka.de` to complete your submission. The topic of your mail must look like "sapwX, asst 0", where *X* has to be replaced with your group number. Do not forget the due date: Late submissions must be rejected. Before sending in the diff please register with your tutor, by sending a mail to `os161@ira.uka.de` which contains your group id, the name of both team members, and the name of the desired tutor.
You are now done.
Even though the generated diff output should represent all the changes you have made to the supplied code, occasionally students do something "ingenious" and generate non-representative diff output.
Always keep your Mercurial repository so that we may recover your assignment should something go wrong.
Note: If for some reason you need to change and re-submit your assignment after you have tagged it "asst-final", you will need to either delete the tag, commit the changes, re-tag, and re-diff your assignment, or choose a different final tag name, commit the changes, tag with the new tag, and re-diff using the new tag. To delete a tag, use `hg tag --remove tagname`.