1. **STUDY OF INSTRUCTIONAL OPERATING SYSTEMS**

**Date:**

**Question:** Study the Manual, Understand the functionalities and answer the following:

Prerequisite for Installation of OS161 - Instructional Operating System

**Introduction**

OS/161 is a teaching operating system is a simplified system used for teaching undergraduate operating systems classes. OS/161 is indeed a simplified operating system which includes a standalone kernel and a simple userland, all written in C. It runs on a machine simulator, System/161.

***OS/161*** is the operating system, the ***sys161*** is the simulator.

**Installing OS/161 on Your Own Machine**

**(This will be Experiment No 3 – so not to dig much about it now)**

Setting up OS/161 on your own machine is more involved, because both the sys/161 simulator (on which OS/161 runs) and the associated toolchain (compiler, debugger, binary utilities) are NOT set up by default. To run OS/161 on your home machine, you will need to download and build your own copies of the simulator and the toolchain before you can work with OS/161.

<https://www.student.cs.uwaterloo.ca/~cs350/common/Install161NonCS.html>

If things go wrong please have a look at the FAQ for answers and solutions.

<https://www.student.cs.uwaterloo.ca/~cs350/common/os161-faq.html>

**what sudo apt-get update does?**  
  
In a nutshell, apt-get update doesn't actually install new versions of software.  
apt-get update downloads the package lists from the repositories and "updates" them to get information on the newest versions of packages and their dependencies.

Repositories are where the packages or softwares are stored.How Linux works is by adding these repositories, and indexing all the packages in those.

So when we type sudo apt-get update, Operating system will update the list of packages so that we can install them by specifying their name.

**Generally you would get Linux software in the tarball format (.tgz) This file has to be uncompressed into any directory using tar command. In case you download a new tarball by the name game.tgz, then you would have to type the following command**

x - Extract.

v – Verbosely show the .tar file progress.

f – File name type of the archive file.

**$ tar xfvz helloall.tgz**

<http://www.tecmint.com/18-tar-command-examples-in-linux/>

This would create a directory within the current directory and unzip all the files within that new directory. Once this is complete the installation instructions ask you to execute the 3 (now famous) commands : **configure, make & make install**. Most of the users do this and successfully install their softwares. But most of the newbies have no idea what this really does. The rest of the article shall explain the meaning of these 3 commands

Each software comes with a few files which are solely for the purpose of installation sake. One of them is the configure script. The user has to run the following command at the prompt

**$ ./configure**

The above command makes the shell run the script named ' configure ' which exists in the current directory. The configure script basically consists of many lines which are used to **check some details about the machine on which the software is going to be installed**. This script **checks for lots of dependencies on your system**.

For the particular software to work properly, it may be requiring a lot of things to be existing on your machine already. When you run the configure script you would see a lot of output on the screen , each being some sort of question and a respective yes/no as the reply. If any of the major requirements are missing on your system, the configure script would exit and you cannot proceed with the installation, until you get those required things.

**THE MAIN JOB OF THE CONFIGURE SCRIPT IS TO CREATE A ' MAKEFILE '**

This is a very important file for the installation process. Depending on the results of the tests (checks) that the configure script performed **it would write down the various steps that need to be taken (while compiling the software) in the file named Makefile**.

If you get no errors and the configure script runs successfully (if there is any error the last few lines of the output would glaringly be stating the error) then you can proceed with the next command which is $make

**$ make**

The ' make ' is actually a utility which exists on almost all Unix systems.

For make utility to work it requires a file named Makefile in the same directory in which you run make. As we have seen the configure script's main job was to create a file named Makefile to be used with make utility. (Sometimes the Makefile is named as makefile also)

make would use the directions present in the Makefile and proceed with the installation. The Makefile indicates the sequence, that Linux must follow to build various components / sub-programs of your software. The sequence depends on the way the software is designed as well as many other factors.

**The Makefile actually has a lot of labels** (sort of names for different sections). Hence depending on what needs to be done the control would be passed to the different sections within the Makefile Or it is possible that at the end of one of the section there is a command to go to some next section.

**Basically the make utility compiles all your program code and creates the executables.** For particular section of the program to complete might require some other part of the code already ready, this is what the Makefile does. It sets the sequence for the events so that your program does not complain about missing dependencies.

One of the labels present in the Makefile happens to be named ' install ' .

If make ran successfully then you are almost done with the installation. Only the last step remains which is

\*\* If you still need clarity on this – Kindly Contact Faculty In-Charge.

**$ make install**

As indicated before make uses the file named Makefile in the same directory.

When you run make without any parameters, the instruction in the Makefile begin executing from the start and as per the rules defined within the Makefile (particular sections of the code may execute after one another..thats why labels are used..to jump from one section to another).

But when you run make with install as the parameter, the make utility searches for a label named install within the Makefile, and executes only that section of the Makefile.

The install section happens to be only a part where the executables and other required files created during the last step (i.e. make) are copied into the required final directories on your machine. E.g. the executable that the user runs may be copied to the /usr/local/bin so that all users are able to run the software. Similarly all the other files are also copied to the standard directories in Linux. **Remember that when you run *make*, all the executables were created in the temporary directory where you had unzipped your original tarball. So when you run *make install,* these executables are copied to the final directories.**

Thats it !! Now the installation process must be clear to you. You surely will feel more at home when you begin your next software installation.

Questions

1. Why the size of “Executable” way bigger than the size of “Object File” – justify you answer?
2. What is the relation or difference between “ELF” and “a.out”?
3. What are the 4 Stages of Compilation?
4. What is the purpose of "-Wall" option in GCC command?
5. What are the 3 stages of happenings during Pre-Processing?
6. What happens during “sudo apt-get update” ?
7. What happens during "./configure" ?
8. What happens when we run "make" utility?

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl No:** | **Assessment Process Description** | | **Mark(s)** |
| 1 | Precise answering | | 5 |
| 2 | Explanation | | 5 |
|  | Total(10) | | 10 |
| Remarks |  | | |
| Date of Completion: | | Signature | |

**Result:**

**(ii) BUILDING OS161 KERNEL FROM SOURCE**

**Date:**

**Question:** Study the Manual, Understand the functionalities and answer the following:

**OS/161 INSTALLATION**

**Sources:**

<https://www.student.cs.uwaterloo.ca/~cs350/common/Install161NonCS.html>

(Steps : 1 to 8)

<https://www.student.cs.uwaterloo.ca/~cs350/common/Install161.html>

(Steps: 9 to 12)

**Note : Installation steps are available in the above websites**

**PREREQUISITES:**

1. Check for availability for UBUNTU 12.04
2. Enable the internet connection on system using Dashhome -> Network->Network Proxy. Under proxy setting set manual proxy.

HTTP Proxy : 172.16.0.2 Port:8080

HTTPS Proxy : 172.16.0.2 Port:8080

FTP Proxy : 172.16.0.2 Port:8080

Socks Host : 172.16.0.2 Port:8080

Click -> Apply system wide.

Use AUTHENTICATION PASSWORD to set changes.

1. Open Terminal and type the commands one by one for execution

**Sudo apt-get install update**

**sudo apt-get install gettext**

**sudo apt-get install texinfo**

**sudo apt-get install libncurses5-dev**

1. goto home folder using files built-in application

press**Ctrl+H** to show hidden files

open**.bashrc** file in **gedit**

add the following line in the last of the file and save and log out of the user to apply changes

**export PATH=$HOME/sys161/bin:$HOME/sys161/tools/bin:$PATH**

**PROCEDURE:**

**STEP 1:**

Download the required installation files as follows.

|  |  |
| --- | --- |
| **File Name** | **Link** |
| **Binutils for MIPS** | <http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-binutils.tar.gz> |
| **GCC MIPS Cross-Compiler** | <http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-gcc.tar.gz> |
| **GDB for Use with OS/161** | <http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-gdb.tar.gz> |
| **bmake for use with OS/161** | <http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-bmake.tar.gz> |
| **mk for use with OS/161** | <http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-mk.tar.gz> |
| **sys/161** | <http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/sys161.tar.gz> |
| **OS/161** | <http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161.tar.gz> |

Note :

**i.**

**ALSO use -i flag in make and make install commands if make seems to encounter errors or is incompletely executed**

**ii.**

**In step 5, ALSO use**

**make MAKEINFO=missing -i**

**make install MAKEINFO=missing -i**

**STEP 2: Build and Install the Binary Utilities (Binutils)**

Unpack the binutils archive:   
**tar -xzf os161-binutils.tar.gz**

Move into the newly-created directory:   
**cd binutils-2.17+os161-2.0.1**

Configure binutils:   
**./configure --nfp --disable-werror --target=mips-harvard-os161 --prefix=$HOME/sys161/tools**

Make binutils:   
**make**

Finally, once **make** has succeeded, install the binutils into their final location:   
**make install**

This will create the directory **$HOME/sys161/tools/** and populate it.

Note that you may need to log out and log back in again so that this PATH change will take effect. You can check the current setting of the PATH environment variable using the command   
  
**printenv PATH**

**Step 4: Install the GCC MIPS Cross-Compiler**

Unpack the gcc archive:   
**tar -xzf os161-gcc.tar.gz**

Move into the newly-created directory:   
**cd gcc-4.1.2+os161-2.0**

Configure gcc   
**./configure -nfp --disable-shared --disable-threads --disable-libmudflap --disable-libssp --target=mips-harvard-os161 --prefix=$HOME/sys161/tools**

Make it and install it:   
**make   
make install**

**Step 5: Install GDB**

Unpack the gdb archive:   
**tar -xzf os161-gdb.tar.gz**

Move into the newly-created directory:   
**cd gdb-6.6+os161-2.0**

Configure gdb   
**./configure --target=mips-harvard-os161 --prefix=$HOME/sys161/tools --disable-werror**

Make it and install it:   
**make   
make install**

**Step 6: Install bmake**

Unpack the bmake archive:   
**tar -xzf os161-bmake.tar.gz**

Move into the newly-created directory:   
**cd bmake**

Unpack mk within the bmake directory:   
**tar -xzf ../os161-mk.tar.gz**

Run the bmake bootstrap script   
**./boot-strap --prefix=$HOME/sys161/tools**

As the **boot-strap** script finishes, it should print a list of commands that you can run to install bmake under **$HOME/sys161/tools.**

The list should look something like this:

**mkdir -p /home/kmsalem/sys161/tools/bin  
cp /home/kmsalem/bmake/Linux/bmake /home/kmsalem/sys161/tools/bin/bmake-20101215  
rm -f /home/kmsalem/sys161/tools/bin/bmake  
ln -s bmake-20101215 /home/kmsalem/sys161/tools/bin/bmake  
mkdir -p /home/kmsalem/sys161/tools/share/man/cat1  
cp /home/kmsalem/bmake/bmake.cat1 /home/kmsalem/sys161/tools/share/man/cat1/bmake.1  
sh /home/kmsalem/bmake/mk/install-mk /home/kmsalem/sys161/tools/share/mk**

**Run the commands printed by** **boot-strap** **in the order in which they are listed inyour terminal screen.**

**Step 7: Set Up Links for Toolchain Binaries**

**mkdir $HOME/sys161/bin**

**cd $HOME/sys161/tools/bin**

**sh -c 'for i in mips-\*; do ln -s $HOME/sys161/tools/bin/$i $HOME/sys161/bin/cs350-`echo $i | cut -d- -f4-`; done'**

**ln -s $HOME/sys161/tools/bin/bmake $HOME/sys161/bin/bmake**

When you are finished with these steps, a listing of the directory **$HOME/sys161/bin** should look similar to this:

**bmake@ cs350-gcc@ cs350-ld@ cs350-run@**

**cs350-addr2line@ cs350-gcc-4.1.2@ cs350-nm@ cs350-size@**

**cs350-ar@ cs350-gccbug@ cs350-objcopy@ cs350-strings@**

**cs350-as@ cs350-gcov@ cs350-objdump@ cs350-strip@**

**cs350-c++filt@ cs350-gdb@ cs350-ranlib@**

**cs350-cpp@ cs350-gdbtui@ cs350-readelf@**

**Check for the availability of all the 22 files**

**Request to manually type the commands from step 8 onwards.**

**Step 8: Build and Install the sys161 Simulator**

Unpack the sys161 archive:   
**tar -xzf sys161.tar.gz**

Move into the newly-created directory:   
**cd sys161-1.99.06**

Next, configure sys161:   
**./configure --prefix=$HOME/sys161 mipseb**

Build sys161 and install it:   
**make  
make install**

Finally, set up a link to a sample sys161 configuration file  
**cd $HOME/sys161   
ln -s share/examples/sys161/sys161.conf.sample sys161.conf**

**Step 9: Install OS/161**

First, create a directory to hold the OS/161 source code, your compiled OS/161 kernels, and related test programs.   
**cd $HOME  
mkdir cs350-os161**

Next, move the OS/161 archive into your new directory and unpack it:   
**mv os161.tar.gz cs350-os161  
cd cs350-os161  
tar -xzf os161.tar.gz**This will create a directory called **os161-1.99** (under **cs350-os161**) containing the OS/161 source code. You should now be able build, install, and run an OS/161 kernel and related application and test programs by following steps.

**Step 10: Configure OS/161 and Build the OS/161 Kernel**

The next step is to configure OS/161 and compile the kernel. From the **cs350-os161** directory, do the following:   
  
**cd os161-1.99  
./configure --ostree=$HOME/cs350-os161/root --toolprefix=cs350-  
cd kern/conf  
./config *ASST0*  
cd ../compile/*ASST0*  
bmake depend  
bmake  
bmake install**

**Step 11: Build the OS/161 User-level Programs**

Next, build the OS/161 user level utilities and test programs:   
**cd $HOME/cs350-os161/os161-1.99  
bmake  
bmake install**

**Step: 12 Try Running OS/161**

You should now be able to use the SYS/161 simulator to run the OS/161 kernel that you built and installed. The SYS/161 simulator requires a configuration file in order to run. To obtain one, do this:   
  
**cd $HOME/cs350-os161/root**

**cp $HOME/sys161/sys161.conf sys161.conf**

**sys161 kernel-ASST0**

You should see some output that looks something like this:

**sys161: System/161 release 1.99.06, compiled Aug 23 2013 10:23:34**

**OS/161 base system version 1.99.05**

**Copyright (c) 2000, 2001, 2002, 2003, 2004, 2005, 2008, 2009**

**President and Fellows of Harvard College. All rights reserved.**

**Put-your-group-name-here's system version 0 (ASST0 #1)**

**316k physical memory available**

**Device probe...**

**lamebus0 (system main bus)**

**emu0 at lamebus0**

**ltrace0 at lamebus0**

**ltimer0 at lamebus0**

**beep0 at ltimer0**

**rtclock0 at ltimer0**

**lrandom0 at lamebus0**

**random0 at lrandom0**

**lhd0 at lamebus0**

**lhd1 at lamebus0**

**lser0 at lamebus0**

**con0 at lser0**

**cpu0: MIPS r3000**

**OS/161 kernel [? for menu]:**

The last line is a command prompt from the OS/161 kernel. For now, just enter the command **q** to shut down the simulation and return to your shell.

After logging out, to get back again into OS/161 just follow **STEP 12**.

**Questions:**

1. What does the command *sys161 kernel***-ASST0 do?**

2. What is the use of *make* command?

3. Syntax and purpose of *tar* command and example.

4. Difference between *emulator* and *simulator*

5. What is the purpose of *.bashrc* file?

6. What is a network*proxy*, *port*?

7. What is the purpose of *export* command? Is there any *import* command?

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl No:** | **Assessment Process Description** | | **Mark(s)** |
| 1 | Precise answering | | 5 |
| 2 | Explanation | | 5 |
|  | Total(10) | | 10 |
| Remarks |  | | |
| Date of Completion: | | Signature | |

**Result :**