Chapter 1

Comments on User Mode Linux and Testing Perl Script

The general setup is fairly straight forward. The UML web-page hosted by sourceforge has vast amounts information about setting up UML[**?**]. The trick is setting up the network section with debugging. The documentation for setting up the network section is not detailed.

1.1 Installing the UML utilities

The first thing is to install the UML tools. The UML tools allow the virtual networks to talk to each other and provides additional support tools.

To install the UML utilities, download the uml_utilities_<build number>.tar.bz2 from the user mode linux download page (http://user-mode-linux.sourceforge.net/dl-sf.html). With root privilege, uncompress the tar.bz2. This will uncompress the source into a directory "tools". Change the directory into tools and execute a “make all”.

```
tar xvjf uml_utilities<build number>.tar.bz2
cd tools; make all
make install
```

Fig. 1.1: Unpackaging UML tools

This will install the UML tools on the host machine. Again these tools are needed for the UML sessions to communication with each other.

There are five main tools. It is not necessary to call them up or use them by command line prompt. The UML session needs the tools installed and will use them in the background. The five main tools are:

- uml_moo - merge COW(Copy On Write) file with its backing file
- uml_mconsole - UML management console
- uml_switch - switch daemon
- uml_net - setuid helper for network setup
- tunctl - create and control TUN/TAP interfaces

1.2 Compile the kernel the UML kernel

The Linux sourced code is available at the kernel.org repository[**?**]. At the time of the experimenting with UML the newest kernel was 2.4.18. The Linux kernel is intimidating . Unpackaged the code is about 300 MegaBits without any builds.

he program "patch" updates the source code. For any Perl fans, this is the same patch invented by Larry Wall. It is used for maintaining source code and for distributions of versions. Patch takes a diff file (differences of the original file and newer file) and makes the original file a newer version. Take the UML patch file and apply it to the root level of the UML source code. The UML patch is available from the UML sourceforge web-page (the file name looks similar to: uml-patch-2.4.19-51.bz2). The bz2 extension represents bzip2 format (a compression format similar to zip files. Fig. 1.2 shows how to uncompress the file type and apply the uncompressed patch file.

```
bunzip2 uml-patch-2.4.19-51.bz2
patch -p1 < uml-patch-2.4.19-51 # at the root level of the source code
```

Fig. 1.2: Installing UML patch file

Notice the file name does not have the .bz2 extension. The patch command can actually work from a couple of different levels. The -p1 means at the root level of the source code. -p0 means one directory above (../ directory) and the updates are redirected into the directory.

Once the patch is applied, the next step is to compile the kernel code. This is going to involve two commands. These commands are shown in Fig. 1.3 and need to be executed at the top level of the patch Linux source tree.

```
make xconfig ARCH=um
make linux ARCH=um
```

Fig. 1.3: UML Make command

The GUI menu screen in Fig. 1.4 should pop up.

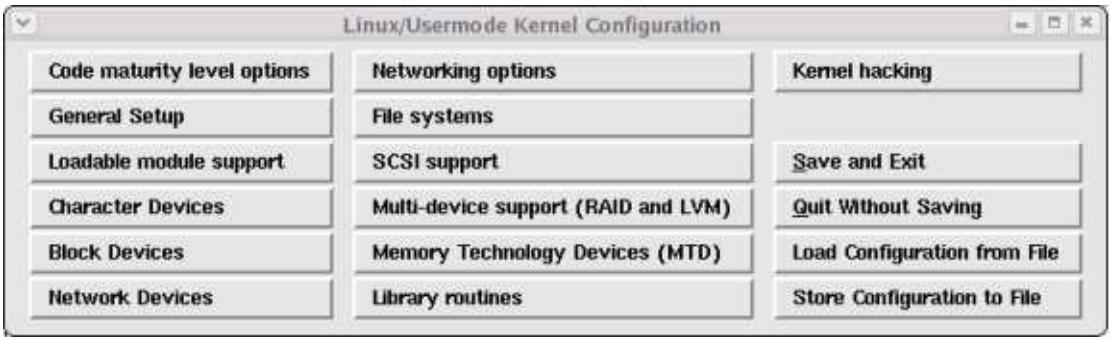

Fig. 1.4: Uml make config screen

Just the default is good. To tweak the kernel a little more, click the "kernel hacking" button and turn on all four options. This adds some extra information in the back-traces and also ensures the debugging component of the kernel is working properly. In the newer kernel (2.4.18+), especially with the skas mode patch into the host kernel setting these debug options breaks the compile. Turning on all the options for the "kernel hacking" in kernel version before 2.4.17 gave additional debug information to be used while debugging the kernel. This thesis' experiments are based on the 2.4.26 kernel which works with the skas host patch.

After setting the desired options, hit the "save and exit" button and exit the GUI. This will save the configuration into a file called, ".config." ".config" is a hidden file.

There were two commands mentioned (patch -p1 < uml-patch-2.4.19-51; make xconfig). If the next command does not execute properly, try “make dep ARCH=um.” There have been instances where the UML kernel would not compile completely. "make dep ARCH=um" fixes the problem.

“make linux ARCH=um” creates the Linux executable at the top level of the Linux kernel source tree. The compile should take almost 10 minutes the first time the UML Linux kernel is compiled. Once the kernel is compiled; the second compile only builds modified objects. So, future recompilation do not take as long. Once the Linux executable is create the just copy the "linux" file to the same directory as the root_fs.

1.3 Creating the root_fs

If an experimenter is bold enough to create their own root_fs, there are tools to assist in this venture. For those who prefer the simple, a root_fs’ can be downloaded from the UML’s sourceforge website[**?** **?**].

Slackware is probably the purest Linux distribution, meaning the programs and libraries are not "enhanced" for a specific Linux kernel and architecture. A root file systems build with RedHat are the worst violators of changing or enhancing source code to perform better. UML is an architecture built on virtual software devices, not optimized for RedHat’s enhancements.

One way to make a custom root_fs is to create an empty root system file (maybe 400-600 megs) with the "dd" command (dd if=/dev/zero of=new_filesystem seek=100 count=1 bs=1M). Then mount the empty root_fs using "mount <name of empty file> <mount directory> -o loop" and follow the instructions from the document "linux from scratch"[**?**]. This will create a personalized Linux distribution. Creating a root_fs from a distribution might be better.

There are four tools are available to create root_fs[1] [**?**]; the two most common are uml builder[**?**] and mkrootfs[**?**]. Both tools use RPM distributions (i.e., Redhat, Suse, and Mandrade). Mkrootfs is a very simple tool for advanced users and is command line. Mkroofs is mainly used for creating multiple root_fs from a script. Uml Builder is a simple GUI interface for single root_fs creation.

Uml Builder has a step by step interface allowing for networking and xwindow setup. At the end of the setup, the GUI creates the script file for turning on the xwindows. As a side note, the script

[1] http://user-mode-linux.sourceforge.net/fs_making.html has more information about these tools

file for turning on xwindows is not an easy script file. Unlike turning on networking and debugging, the script for setting up xwindows is extremely complicated, is sparely documented on the web, and is pages long. If a script for xwindows is needed, consider the twenty minute configuration time with uml builder oppose to hacking the instructions.

1.4 Resizing the root_fs

There are times when resizing the file system is necessary because the default size is not enough. To increase the file size just do these three command in the frame below. The file size in this example is increased to 640 MB, but the 640 MB can be changed to a more desirable size. resize2fs is part of the e2fsprogs utility suite[**?**] for ext2 file format.

```
dd if=/dev/zero of=root_fs bs=1 count=0 seek=640MB
e2fsck -f root_fs
resize2fs -p root_fs
```

Fig. 1.5: Resizing a uml filesystem

1.5 Parameters for running UML

If the root_fs is in the same directory where the Linux executable is launched, the user can merely type "./linux" and UML should start. Parameters are not really needed. The parameters are used for networking options, redirecting the location of the root_fs, and setting the memory. The main advantage for this appendix is the debugger setup. When debugging UML, have the Linux executable and the source at the top level of the source tree. If the UML exectuable is located somewhere else, the debugger (gdb) cannot find the source code. Fig. is an example of a script used to run UML.

```
./linux umid=windom mem=128M debug=go ubd0=root_fs udb2=swap
```

Fig. 1.6: Running uml

This example is without network connection (networking will talked about in the next section).

The ./linux, represents in this current directory. There are RPM binary versions of UML (which do not allow for debugging and other good stuff). The RPM binary version of UML installs the Linux executable in the /usr/bin. Now while this is nice and good, it interferes with the running of the Linux executable. The directory paths are confused and UML no longer had debugging capability. This is what happens when there are two versions of the same file both in the path. This is not an uncommon problem, so make a note. Run the Linux executable with the current directory symbol in the front. It is an extra "./".

The umid=windom is a label given to the Linux kernel. The UML utilities references the UML session by the label passed in with umid=<id>. The UML utilities controls the networking, monitors UML sessions, and provides a jail program to catch hackers.

The mem=128M is the memory UML will be reserving. This is defaulted to 128 megabits, but can be as low as 16 M (16 megabit) without xwindows. Most of the machines used for the experiments only had 512 megabits and with three machines at 128, it left the host operating system with 128. Not good.

The debug=go is an important switch. This switch does not work if the skas is patched into the host machine's kernel.

The udb0=root_fs is the name of the root_fs. If this option is left blank, the Linux kernel executable will default to root_fs.

The last option, udb2=swap is the swap space for the UML. The "swap" is assigned to udb2 and is not a file but use the host kernel's swap.

1.6 Network Setup

There are two more optional parameters which were not mentioned. These options are for the networking. While UML has five different network drivers it can utilize, the two main virtual network drivers are Tuntap and Ethertap.

A UML session requires two IP addresses per device driver[**?**]. The first IP address needs to be in the command line prompt, and the second IP address is used by the "ifconfig" utility within the UML session. The Ethertap and Tuntap virtual network drivers need two IP addresses to interface with the IP Tables and the virtual machines. In Fig. 1.7 is a copy of what the routing tables look likes

after the UML interface drivers (tuntap drivers) are interfaced with the host machine's networking components.

```
[root@walden uml5]# route -n
IP routing table Destination Gateway Genmask Flags Metric Ref Use Iface
128.198.60.172    0.0.0.0           255.255.255.255   UH 0 0 0 tap1
128.198.60.173    0.0.0.0           255.255.255.255   UH 0 0 0 tap0
128.198.60.128    0.0.0.0           255.255.255.128   U  0 0 0 eth0
127.0.0.0         0.0.0.0           255.0.0.0         U  0 0 0 lo
0.0.0.0           128.198.60.129    0.0.0.0           UG 0 0 0 eth0
```

Fig. 1.7: Routing table with tuntap entries

On the far right, there are two interfaces labeled tap0 and tap1. These interfaces are used by the UML session stating 128.198.60.172 and 128.198.60.173 are to be routed to the virtual machines using the tuntap drivers.

The next section will talk about the first of the two drivers Ethertap more in depth. While it is possible to have two UML sessions using either Tuntap or Ethertap, Ethertap is the older of the two and is no longer recommended in UML development. It still deserves some mention.

1.6.1 Ethertap

Fig. 1.8 is a command line using Ethertap.

```
./linux umid=windom eth0=ethertap,,,128.198.60.133 mem=60M udb2=swap
```

Fig. 1.8: Uml run with ethertap

eth0=ethertap,,,128.198.60.133 is the command line parameter which interfaces with the IP routing table entry for Ethertap. Ethertap is another device driver similar to the Ethernet driver and provides packet reception and transmission for the application level. Ethertap does not allow for multiple network interfaces in a UML session and is not as secure as Tuntap.

1.6.2 Tuntap

Tuntap like Ethertap is a virtual network device driver and interfaces with the host machine's IP routing table[**?**]. Tuntap does allow for a UML session to have multiple interfaces to the host machine's routing table. For a while, the use of Joseph Mac's Linux Virtual Machine (LVS) was pondered. LVS requires two network interfaces to do a NAT (Network Address Translation). The first network interface was for the back-end servers and the other for general Internet access. Having one UML session with two network interfaces allows for the creation of many virtual Local Area Networks (LAN's) on one machine.

Fig. 1.9 is a command line using tuntap using a single interface:

```
./linux umid=windom eth0=tuntap,,,128.198.60.172 mem=128M //
debug=go ubd0=root_fs udb2=swap
```

Fig. 1.9: Uml run with tuntap

The umid=windom is used for assigning an ID to the UML session. As a side note, the ID to the UML session is used by the UML utilities tools to monitor and give external commands to the UML session from the host machine. Fig. 1.10 is the command line parameters for two network interfaces.

```
./linux umid=windomCM eth0=tuntap,,,128.198.60.162 eth1=tuntap,,,128.198.60.163 //
mem=128M debug=go ubd0=root_fs udb2=swap
```

Fig. 1.10: Uml run with two tuntap interfaces

Add an additional "eth(n)" driver (where n is the number of the device) gives the UML session another network device. If there is another Tuntap driver, an additional IP address needs to be assigned through the command line.

1.7 Once inside the UML session

Run the commands in Fig. 1.11 at the UML prompt to gain network connectivity.

```
ifconfig eth0 172.31.0.169
route del -net 172.31.0.0 dev eth0 netmask 255.255.0.0
route add -host 172.31.0.101 dev eth0
route add default gw 172.31.0.101
```

Fig. 1.11: Ifconfig commands for setting up the network device

Where 172.31.0.169 is the IP address of the UML session (not the same IP address as the one passed in). 172.31.0.0 representing the subnet. 172.31.0.101 represents the host machine's IP address. This IP address will be used as a gateway for the UML session. A script file can set all the network options once UML session starts. To adding a second network interface, add the command line parameters in Fig. 1.12 setting the eth1 interface.

```
ifconfig eth1 192.68.0.100
route del -net 192.68.0.0 dev eth1 netmask 255.255.0.0
route add -host 192.68.0.100 dev eth1
```

Fig. 1.12: Ifconfig commands for setting up a second network device

1.7.1 Inside of Slackware – configuring the Slackware root_fs

The Slackware root_fs can be used just "as is" without any modifications. There are many tweaks that can be done to the Slackware root_fs. These tweaks include: setting the number of xterm windows spawning off, setting up the hostname, configuring dns, adding additional programs (like ssh), and how to set the routing table to automatically start up without a script or typing in the routing commands.

1.7.2 spawning off xtermals (setting the number of xterminals)

Slackware is a pure version of Linux. The maintainers of Slackware do not make excessive modifications to Linux kernel, libraries, or programs to run well together. Not adding additional modifications makes Slackware work well with the non-conforming UML kernels. The only problem with having a "pure version", is not having the configuration utilities found in the RedHat distributions.

Without these UI configuration utilities modifications are done by editing files.

Spawning off xterminals are found in the /etc/inittab. Open up the /etc/inittab file with an editor. Scrolling down the /etc/inittab, do a search on "spawn" and grab the section listed below.

Like most scripting languages or Linux config files the "#" means comment. In Fig. 1.13, the c0: is the terminal the UML session is called on. The remaining c1-c6 are spawned xtermals. When c1-c6 are uncommented, the UML session will bring up additional xtermals connected into the running UML sessions.

```
# These are the standard console login getties in multiuser mode:

c0:1235:respawn:/sbin/agetty 38400 tty0 linux
#c1:1235:respawn:/sbin/agetty 38400 tty1 linux
#c2:1235:respawn:/sbin/agetty 38400 tty2 linux
#c3:1235:respawn:/sbin/agetty 38400 tty3 linux
#c4:1235:respawn:/sbin/agetty 38400 tty4 linux
#c5:1235:respawn:/sbin/agetty 38400 tty5 linux
#c6:12345:respawn:/sbin/agetty 38400 tty6 linux
```

Fig. 1.13: Spawning xterminals

1.7.3 Setting up DNS

Setting up the DNS consists of three parts: correctly configuring the host name, setting up the host file, and assigning the DNS servers. The first involving modifying a file called HOSTNAME (all caps). The HOSTNAME file is in the /etc directory. The file only consists of the full DNS name of the UML session. For example, an UML session with the name walden.uccs.edu would only have walden.uccs.edu inside the HOSTNAME file.

The host file (/etc/hosts) is the first place the DNS server checks for a listing (by default). Unlike the HOSTNAME file this is not unique to Slackware, generally all Linux distributions have this file. Open and add all the DNS entries for your private network. The format is space delimited: <IP address> <Full DNS name> <nickname>

The last file is the /etc/resolv.conf. The resolv.conf lists all the available DNS servers. The format is also space delimited: nameserver <IP address of the DNS server>. Word of caution the DNS resolver is sensitive the to the host machine's firewall. If the firewall is turned on, the answer

from the DNS server will not be able to make it through the firewall. The firewall is not set up to handle TAP's IP address and rejects the packet designated for TAP driver.

1.7.4 Installing and uninstalling programs

To install a Slackware program, either mount the root_fs or turn on the host machines ftp server to get the package onto the root_fs. Fig. 1.14 shows the command to mount the root_fs.

```
mount root_fs /mnt/root_fs -o loop
```

Fig. 1.14: Mounting a root file system

/mnt/root_fs is directory where the file system should be mounted. Pending on the version of Slackware, a mirror site should have the packages needed. The Slackware directory structure is awkward at first. The âdirectory listingâ file should mention where a package is kept. For example the ssh package would be under the "n" directory (for network). The package should end in .tgz extension. Fig. 1.15 shows the commands to install and uninstall slackware packages.

```
installpkg xf_bin.tgz # installs a package
removepkg xf_bin.tgz # uninstalls a package
# xf_bin is the package name
```

Fig. 1.15: Installing and uninstalling slackware packages

1.7.5 Configuring the network on startup

Under Slackware root_fs, the network configuration file is located in /etc/rc.d. Before 9.1, fill in the IP address (IPADDR field), netmask(NETMASK FIELD), and gateway (GATEWAY field) in the rc.inet1. If there are questions, there are BOLD comments instructing what fill in. For Slackware 9.1, fill in the network parameters in the /etc/rc.d/rc.inet1.config. Filling in the config file will save time by avoiding the manually network setup (using ifconfig).

1.8 Debugging with UML

Starting about UML kernel patch 2.4.19, Jeff Dikes (the maintainer of UML) came up with a the Skas (Separate Kernel Address Space) patch which causes the UML session to run in an entirely different host space from its processes[**?**]. It also allows for a debugger to attach to the main process and step through the Linux source code. While debugging was in the previous tt mode (Tracing Thread), a separate debug window using GDB would come up with the correct thread attached. The newer skas mode allows for outside debuggers to attach to the kernel.

1.8.1 Installing the skas patch on the host machine

When recompiling a kernel, the default settings are missing device drivers which most distributions include as modules. Three things are needed to install the skas patch. First, get a Redhat kernel source tree as close to the skas patch version as possible. linux-2.4.22-1.2115.nptl (from Fedora core 2) works with host-skas3.patch[**?**]. Second, download a Linux kernel from the Linux Kernel Repository[**?**] compatible with the skas patches listed at the UML website. For this example, kernel 2.4.24 works. Once the kernel is downloaded , and unpackaged (for kernel-2.4.24.tar.bz2 type: tar xvjf kernel-2.4.24.tar.bz2; for kernel-2.4.24.tar.gz: tar xvzf kernel-2.4.24.tar.gz). Change into the top directory of the kernel code (cd linux-2.4.24) and type "make mrproper". This is a clean, just like "make clean" but stronger. Apply the skas patch, "patch -p1 < host-skas3.patch". Copy from the Redhat source kernel a config file (kernel-2.4.22-i686.config) located under linux-2.4.22-1.2115.nptl/configs. Copy this config file to the top of the 2.4.24 kernel root tree and rename it to ".config" (cp ../linux-2.4.22-1.2115.nptl/configs/kernel-2.4.22-i686.config .config)[**?**]. Then type "make old config". A bunch of options should come up. Set all of the them to the default (just hit return) except the âproc_mmâ, set it to true. The âproc_mmâ is the skas patch. Once this is done type "make dep; make bzImage; make modules; make modules_install". This will build the kernel and install the modules. Next you need to install the kernel (cp arch/i368/boot/bzImage /boot/vmlinuz-2.4.24), set the initrd (mkinitrd /boot/initrd-2.4.24.img 2.4.24), and add a grub entry (just make a copy of a previous entry and change the initrd and vmlinuz files to match the other one). Then reboot. For additional help doing this, see the howto kernel document[**?**].

1.8.2 Crashing gracefully

This section contains three tips which can help immensely when crashing the User Mode Linux session. These tips are: set a breakpoint in panic.c:panic function, verify the User Mode Linux processes are killed, and check the file system using fsck.ext2.

User Mode Linux's kernel modifications has the scheduler call the panic.c:panic function to prompt the user when a crash occurs. This call to panic.c is good for the kernel developer in many ways: setting a breakpoint within the panic.c:panic function will create a back trace letting the developer know what caused the kernel crash and gives what values were on the stack when the crash occurred.

When a crash occurs, the User Mode Linux's processes might still be alive and attached to the root_fs. It is always a good idea to do a process status (ps command) and pipe it with grep, looking for âlinuxâ or the UML session's ID name. Fig. 1.16 shows an example.

```
[frank@walden bin]$ ps aux | grep linux
frank 3332 pts/2 S /bin/sh /home/frank/linuxRs2/runRS2
frank 3333 pts/2 S /home/frank/linuxRs2/linuxRS2 (feline) [/sbin/rmmod]
frank 3335 pts/2 T [linuxRS2]
frank 3340 pts/2 S /home/frank/linuxRs2/linuxRS2 (feline) [/sbin/rmmod]
frank 3341 pts/2 S /home/frank/linuxRs2/linuxRS2 (feline) [/sbin/rmmod]
frank 3592 pts/4 S /bin/sh /home/frank/linuxRs/runRS
frank 3593 pts/4 S /home/frank/linuxRs/linuxRS (b2b) [/sbin/rmmod]
frank 3595 pts/4 T [linuxRS]
frank 3600 pts/4 S /home/frank/linuxRs/linuxRS (b2b) [/sbin/rmmod]
frank 3601 pts/4 S /home/frank/linuxRs/linuxRS (b2b) [/sbin/rmmod]
frank 3675 pts/5 S /bin/sh /home/frank/linuxRs3/runRS3
frank 3676 pts/5 S /home/frank/linuxRs3/linuxRS3 (feline) [/sbin/rmmod]
frank 3678 pts/5 T [linuxRS3]
frank 3683 pts/5 S /home/frank/linuxRs3/linuxRS3 (feline) [/sbin/rmmod]
frank 3684 pts/5 S /home/frank/linuxRs3/linuxRS3 (feline) [/sbin/rmmod]
frank 12852 pts/8 S grep linux
```

Fig. 1.16: Checking for a uml process

Now there are three different UML sessions running: feline's, b2b's, and walrus. To kill the process, grab the process numbers which is the number after the owner of the process (in the example above the number after "frank"). Type "kill -9 <process numbers>". All the process numbers

for all the selected UML session needs to be terminated. Using the example above, the command to kill the feline UML session would be: "kill -9 3333 3340 3341 3676 3683 3684". If all the UML process are not terminated after a UML kernel crash and the UML is restarted, a message like the one in Fig. 1.17 appears.

```
VFS: Cannot open root device "ubd0" or 62:00
Please append a correct "root=" boot option
Kernel panic: VFS: Unable to mount root fs on 62:00
```

Fig. 1.17: Locked file system panic message

In conclusion, the most common error causing a UML session to not start (when everything else is working) is another process still attached to the root_fs. Using a "kill -9" on the UML process fixes this problem.

The final tip is to use "/sbin/fsck.ext2 -p <root_fs name>" to check your root_fs systems. It is about five times faster than letting the UML session do the disk check. So when the kernel crashes and all the processes are terminated, type in "/sbin/fsck.ext2 -p <root_fs name>", and it will check the file system a lot faster.

1.8.3 Using a .gdbinit

Having a file called “.gdbinit” with the commands to by pass all the beginning breaks, expedites the debugger’s startup.

Before opening the gdb program, put the file â.gdbinitâ at the top of the level of the UML source. Inside the “.gdbinit” type the two lines in Fig. .

```
handle SIGSEGV pass nostop noprint
handle SIGUSR1 pass nostop noprint
```

Fig. 1.18: Stopping gdb from breaking at signals

The .gdbinit is the default script gdb looks for when started. This is helpful because without these two commands, gdb or any other debugger would stop every second or two. The “.gdbinit”

file can be put in the home directory or use gdb -x <name of the file> to have gdb read the initial script.

1.8.4 Debugging in gdb

Once gdb starts, it is really hard to break it. It is recommended to start gdb without the Linux executable file appended in the command line. Otherwise, the UML session might have to be rebooted (hitting control-c a couple of times might work).

Start gdb. Once gdb is loaded, set it least one breakpoint. Then type "file," then the path to the Linux executable. For example: "file /home/frank/linuxSrc/linux". Gdb will automatically load the Linux executable with the correct parameters. When the program breaks, print the different variables or back-trace to better understand the kernel. For emacs user, gdb also interfaces with emacs[**?**].

1.8.5 Gdb debugging inside Eclipse

Install Eclipse. First download eclipse from one of their mirror sites[2].

Unzip the eclipse package using the unzip command "unzip eclipse-platform-3.0-linux-gtk.zip" also download and install the CDT plug-in. A recommended site is: ftp://eclipse.mirrors.tds.net/pub/eclipse.org/tools/ 2.0-linux.gtk.x86.zip. The plug CDT in should be unzipped at the same directory level as the eclipse zip file (was unzipped).

Creating a project. File -> New->Project; In the new project window as shown in Fig. 1.19.

[2] An example eclipse file would be at ftp://eclipse.mirrors.tds.net/pub/eclipse.org/eclipse/downloads/drops/R-3.0-200406251208/eclipse-platform-3.0-linux-gtk.zip

Fig. 1.19: New eclipse project

Select C->Standard Make C Project. Then hit "next" as shown in Fig. 1.20.

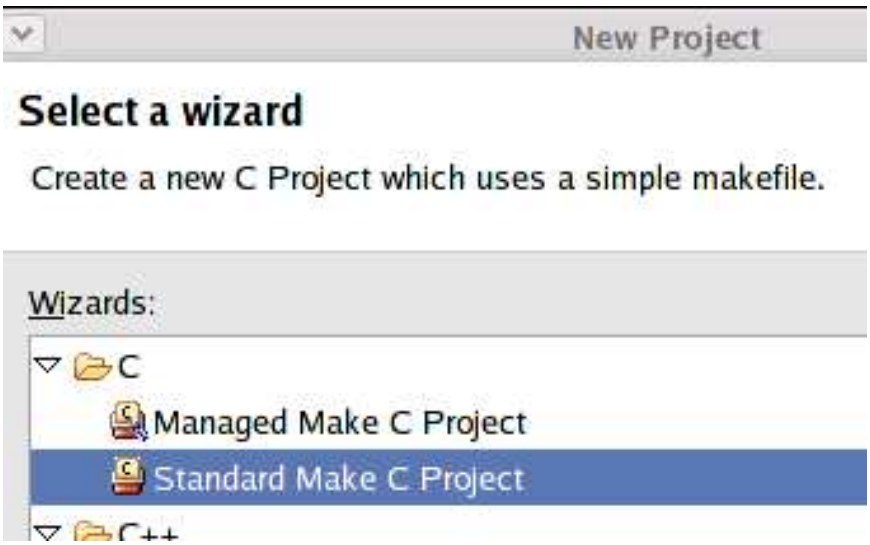

Fig. 1.20: Starting a standard make c project

Project name is the directory where your Linux source code lives. For example, “/home/frank/” is the workspace directory. The top level of my Linux source tree is /home/frank/linuxSrcE. The project is linuxSrcE (just linuxSrcE – no additional directory names) as shown in Fig. 1.21. Leave "Use default" clicked on, then hit "finish".

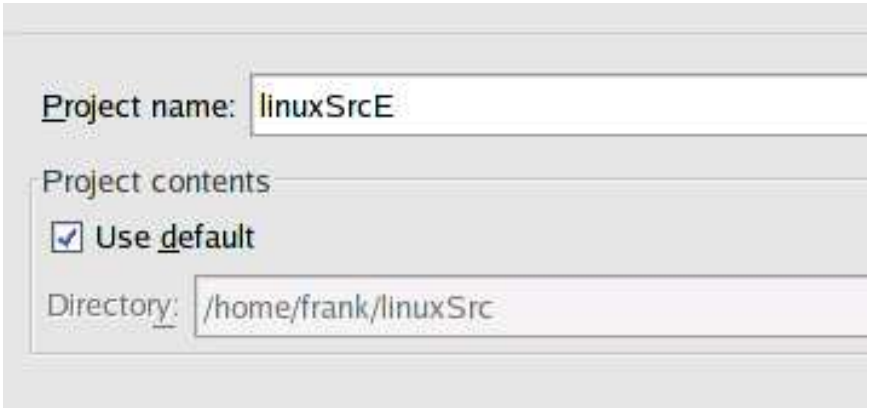

Fig. 1.21: Setting up a project

The progress window C/C++ index should be indexing the Linux tree as shown in Fig. 1.22. The process should take about fifteen minutes to half an hour. The indexer will find all your c, h, and make files files and place them in the project.

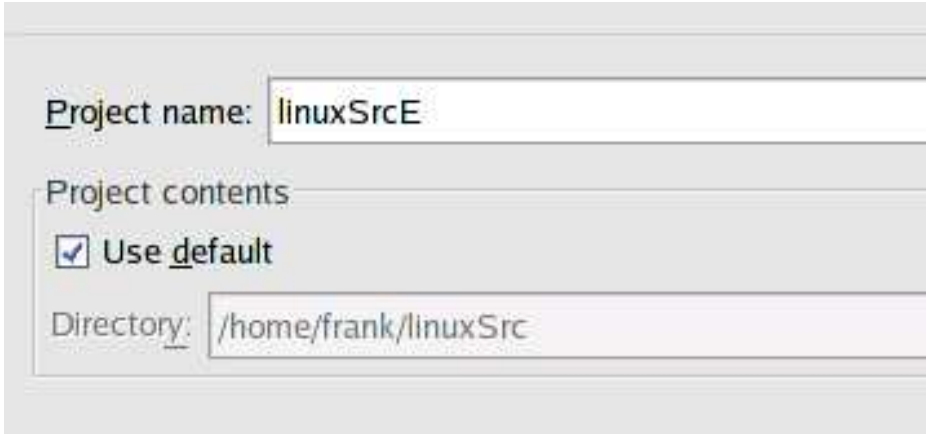

Fig. 1.22: Code indexer

De-select project->build automatically. Otherwise, the framework will try and rebuild the source code everything the code is saved.

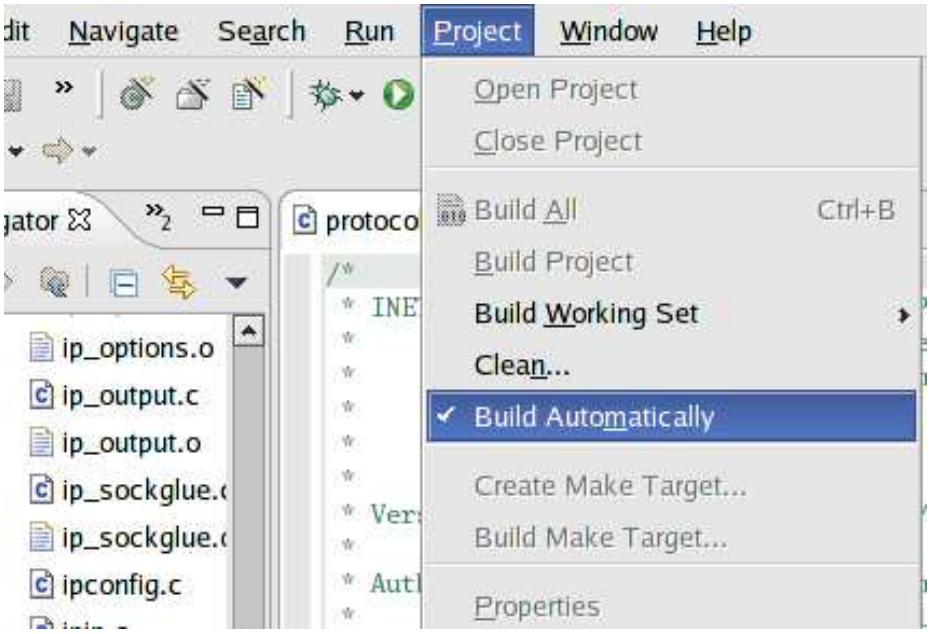

Fig. 1.23: Deselect build automatically

Adding a run / debug. Click the run icon ->run shown in Fig. 1.24. Another screen should come up.

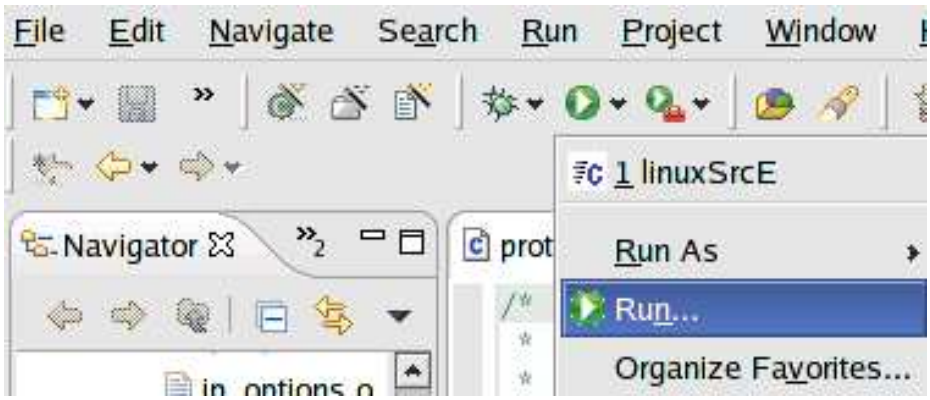

Fig. 1.24: Run icon

Click the "new" button at the bottom (make sure "C/C++ local" is highlighted otherwise the new button will be disabled), as shown in Fig. 1.25.

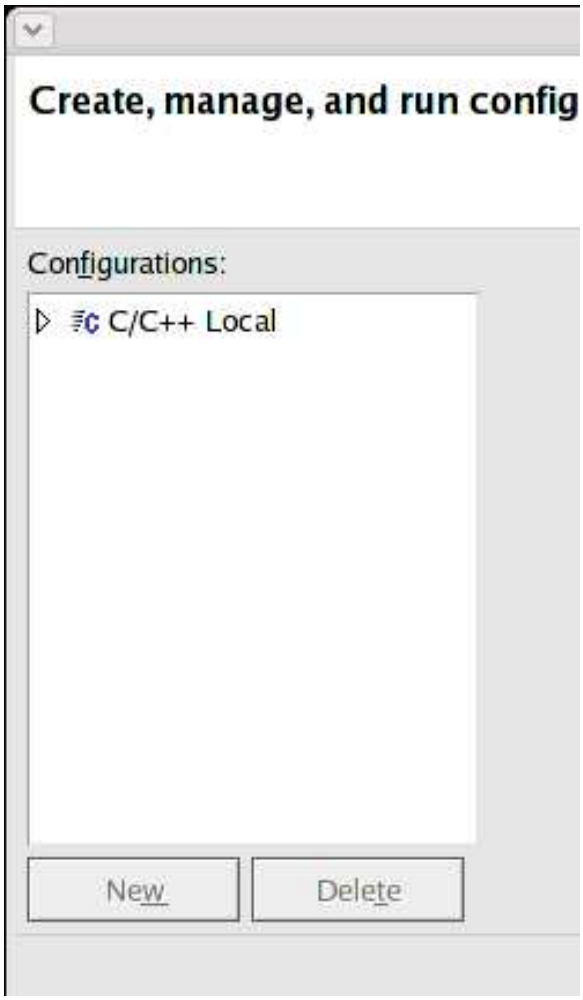

Fig. 1.25: Creating a new run

Under the "main" tab as shown in Fig. 1.26. In the name field, call the project anything. The example uses "linuxSrcE," same as the project name.

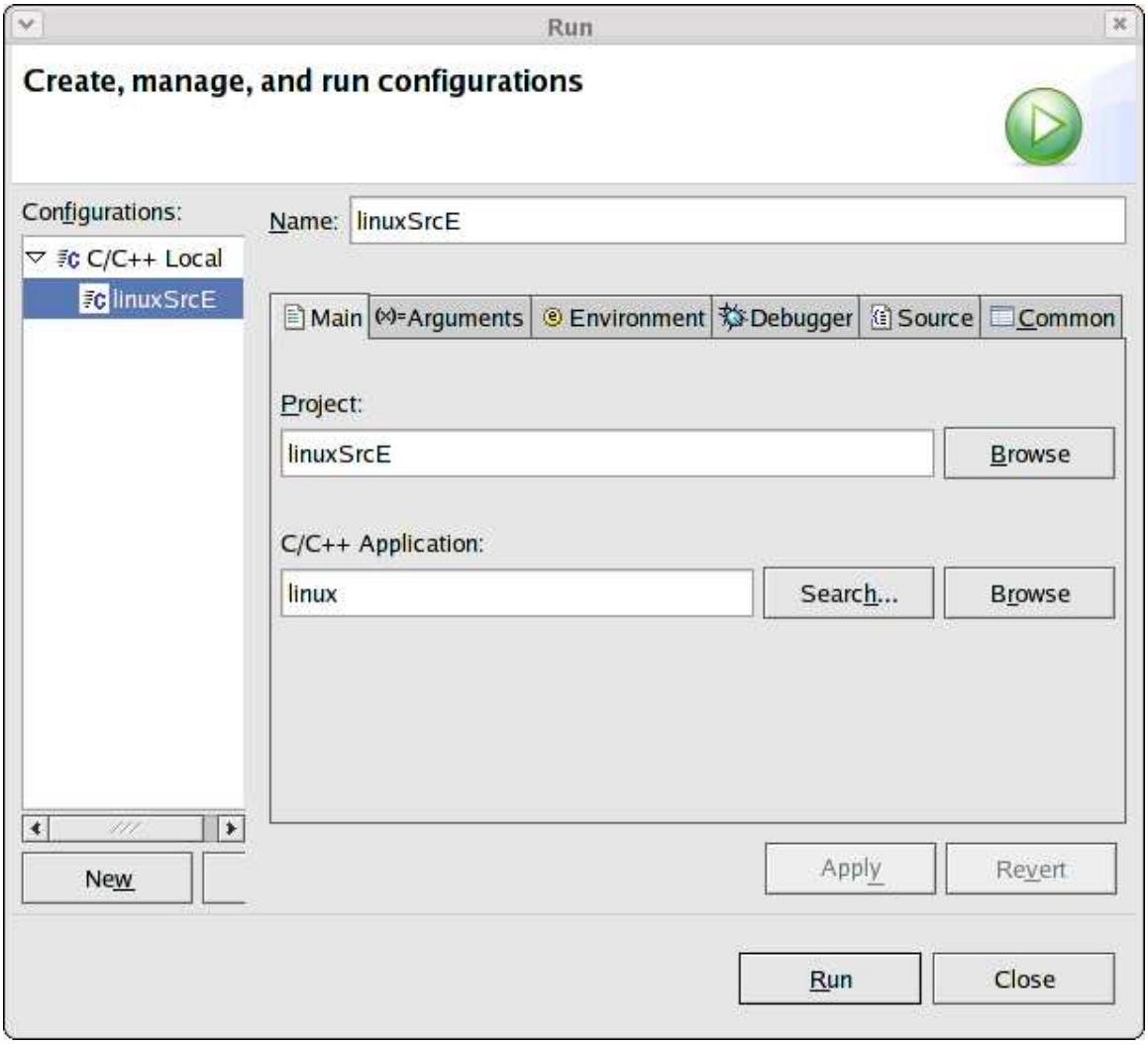

Fig. 1.26: Run main window

Under the "argument" tab (Fig. 1.27) put the arguments used to run uml (i.e. linux <argument>). The arguments the example UML session uses are: "udb0=/home/frank/root_fs umid=lamb eth0=tuntap,,,172.31.0.130 eth1=tuntap,,,172.31.0.131 mem=32M udb2=swap".

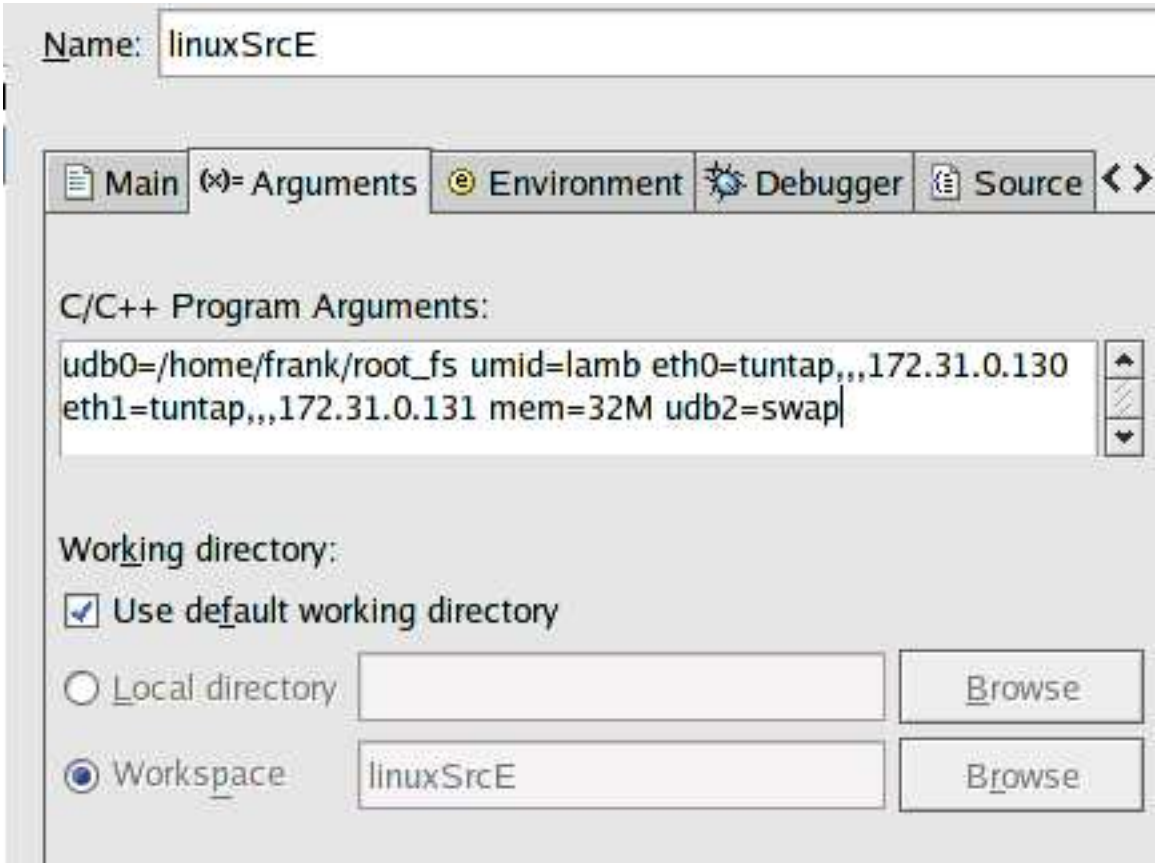

Fig. 1.27: Run argument window

Under the "debugger" tab(Fig. 1.28). Set the drop down combo box to "GDB debugger" and click the radio button "run program in debugger". De-select "Stop at main() on startup"GDB debugger. Gdb debugger should be gdb. If you have built a newer version, put in the absolute directory where the newer gdb is located. In "GDB command file", put the location of your ".gdbinit" file.

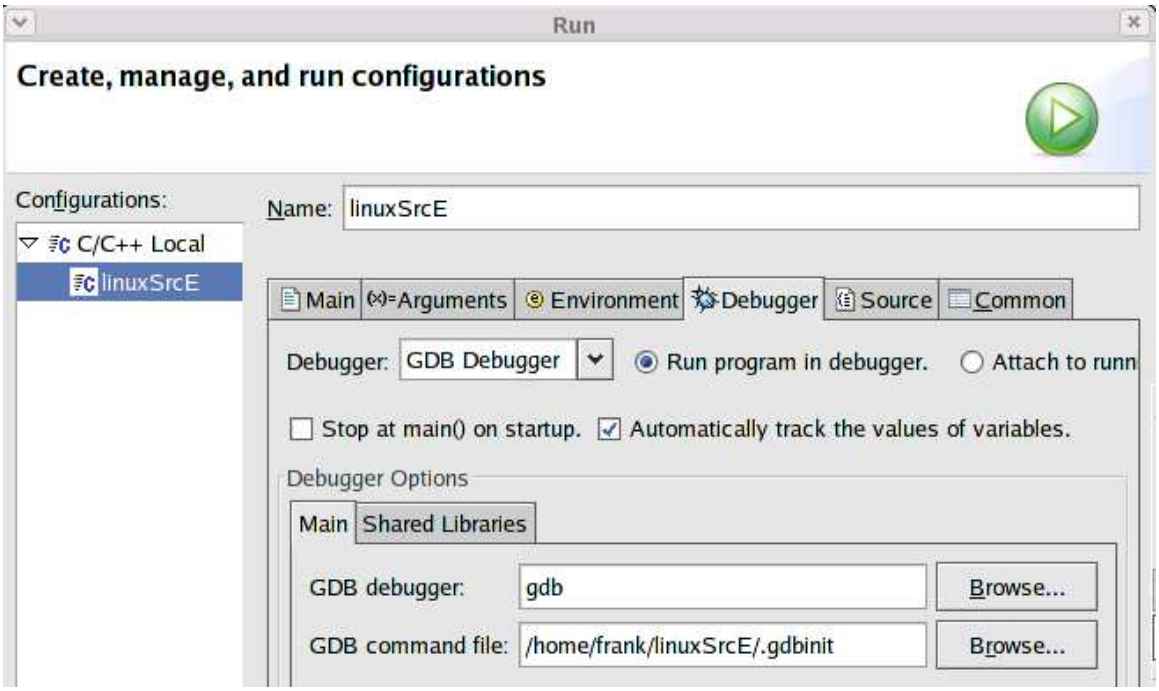

Fig. 1.28: Debug Window

Add a make file. Window->Show view->Make target. The "make target" window on the side should come up (Fig. 1.29). Right click on the root directory (in the example linuxSrcE) and click add make target.

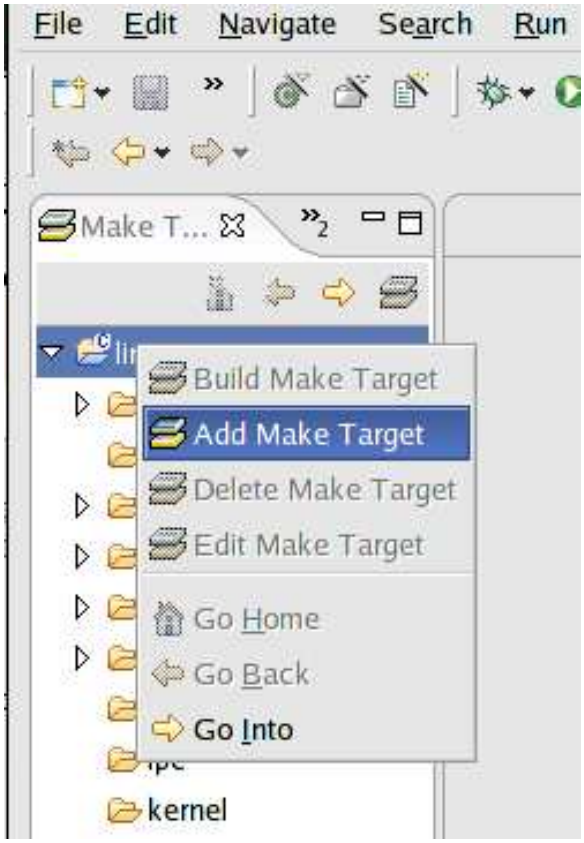

Fig. 1.29: Adding a make target

A window "Create a new make target" should come up. Fill in the target name with a label for the make command; the example uses “linux ARCH=um”. In the "make target" field fill in “linux ARCH=um”, then de-select “use default”. In "build command," type "make linux ARCH=um". Un-select "stop on first build error" and select "run all project builders." Hit the "create" button. Fig. 1.30 is a filled in window. Setup part is done.

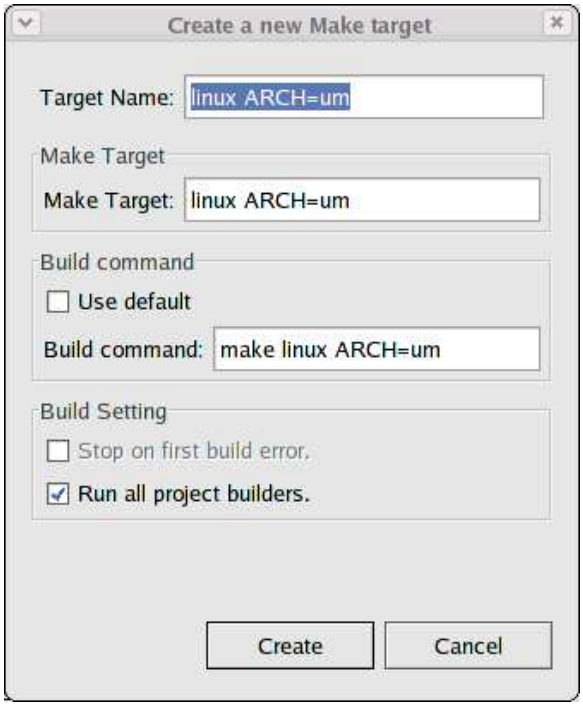

Fig. 1.30: Setting up make target

To build the source, right click the newly created make target and select "run make" as shown in Fig. 1.31.

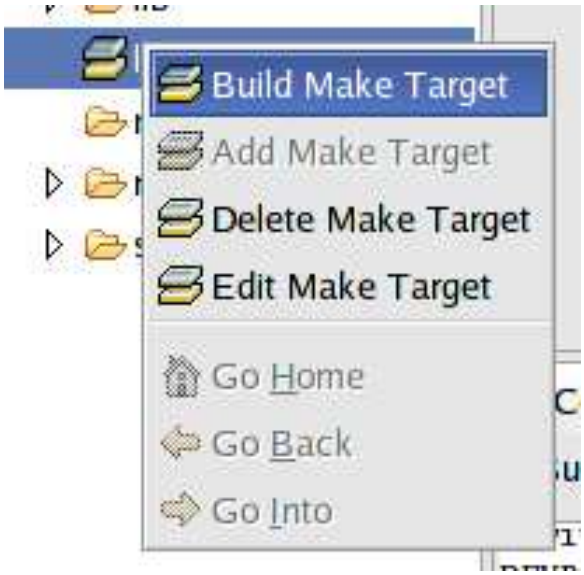

Fig. 1.31: Building make target

To run there should be a icon on the tool bar (green circle with a white play triangle) as shown

in Fig. 1.32. It should drop down and you should see the name of the run session.

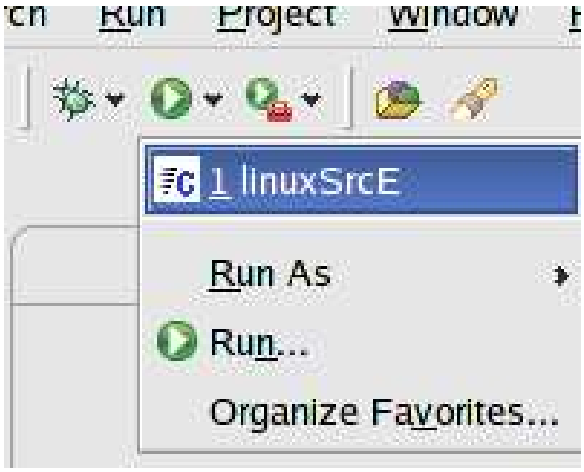

Fig. 1.32: Running the program

To debug there should be a cockroach looking icon next to the run as shown in Fig. 1.33. Click it and you should see the name of the run session. Once you start debugging, there are two windows you should be looking for the "console" and the "debug".

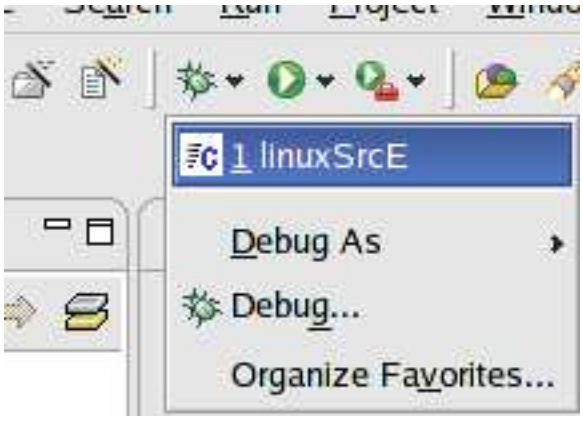

Fig. 1.33: Start the debugger

1.9 Perl script used for testing

There are two perl scripts used for testing, client.pl and server.pl (both included with the media). The client.pl reads read in an input file called input.txt and setsup the proc file system by sending commands to the system prompt. The commands setup the kernel by changing the values in the

proc file system (/proc) and also bandwidth limitation are for each of the connections.

A TCP socket connection is established with the server.pl. The server.pl is listening for the client.pl. When client.pl starts the connection with the machine running server.pl, it sends the commands read in from the input.txt. The server.pl reads only the commands needed to setup the server.

When the client.pl script waits one second and then start a web benchmark[**?**] to measure the speed of the connection. When the web benchmark is completed the results are parsed from the output and stored in a output file.