Getting Started with the *x*-kernel

Larry L. Peterson

June 1997

This note describes the procedures that should be followed to install, configure, build, and run the *x*-kernel as a user-level Unix process. It purposely focuses on a vanilla system. To understand all the options available to you, consult the *x*-kernel Programmer's Manual [1]. We assume the following throughout the rest of this note.

- You will be installing the *x*-kernel in directory /usr/xkernel, and this directory is reachable (via NSF, for example) from all machines on which you want to run the *x*-kernel. You may choose some root directory other than /usr/xkernel. If you do select another root directory, change the steps given below accordingly.
- You will be running the *x*-kernel on a pair of Solaris (Sparc) workstations connected by an Ethernet. We will refer to these two machines as machine A and machine B in the following discussion. If you are building the *x*-kernel on SunOS (on Sparc), IRIX (on SGI), or OSF/1 (on Alpha)—then substitute sunos, irix, or osf1 for solaris, respectively, in the steps that follow. If you are running the *x*-kernel on Linux (on Alpha or x86), then substitute linux-alpha or linux-x86, respectively, for solaris.[1]

Ten Step Process

1. Create directory /usr/xkernel and copy the compressed tar file from ftp://ftp.cs.arizona.edu/xkernel/xkernel.tar.Z into this directory. Then type:

   ```
   cd /usr/xkernel
   uncompress xkernel.tar.Z
   tar xf xkernel.tar
   ```

   This will create several subdirectories in /usr/xkernel. A description of the *x*-kernel directory hierarchy can be found in /usr/xkernel/README. Note that you can skip this step if the *x*-kernelhas already been installed at your site.

2. Create a directory in which you will build an instance of the *x*-kernel. This is called your build directory, and in general, each different configuration of the *x*-kernel has its own build directory.

   ```
   cd /usr/xkernel/user_level/build
   mkdir solaris
   ```

   If directory /usr/xkernel/user_level/build/solaris already exists—because someone else has already followed this procedure—then you have two choices.

   (a) Select another name for your build directory (e.g., mydir) and create a directory with this name in directory /usr/xkernel/user_level/build. The catch to this option is that you have to either have write permission in directory /usr/xkernel/user_level/build, or you have to have someone that has write permission create your build directory for you.

[1] The SunOS and Irix versions of the *x*-kernel are included in the distribution, but they are no longer supported.

(b) Create a build directory in your own home directory; e.g., /usr/llp/xk_build/solaris. This is the preferred approach if a lot of different people are going to be building the *x*-kernel.

Substitute the name of your build directory, as appropriate, in the rest of the steps.

3. Copy configuration files into your build directory.

```
cd /usr/xkernel/user_level/build/solaris
cp ../Template/Makefile.solaris Makefile
cp ../Template/graph.comp .
mkdir client server
cp ../Template/rom.client client/rom
cp ../Template/rom.server server/rom
chmod 664 Makefile graph.comp client/rom server/rom
```

Note that if your build directory is not in /usr/xkernel/user_level/build, then you will have to use the full path names for these various files (e.g., /usr/xkernel/user_level/build/Template/graph.comp) rather than the relative path names (e.g., ../Template/graph.comp) given above.

4. Edit the "XRT =" line of the Makefile you just copied into your build directory to reflect the root of your *x*-kernel source tree. Assuming your tree is at /usr/xkernel, this means changing

```
XRT = ../../..
```

to

```
XRT = /usr/xkernel
```

Note that this step is necessary only if your build directory is underneath your home directory. It is not necessary if your build directory is in /usr/xkernel/user_level/build since from there, ../../.. already points to the root of the *x*-kernel source tree.

5. Edit the last line of the graph.comp file you just copied into your build directory to reflect the root of your *x*-kernel tree. Assuming your tree is at /usr/kernel, this means changing

```
prottbl=/cs/x33/etc/prottbl.std;
```

to

```
prottbl=/usr/xkernel/etc/prottbl.std;
```

6. Edit the two rom files you just copied to reflect the addresses of the two machines on which you will be running the *x*-kernel. This will involve editing the Real IP address field of the two lines that begin with arp. Specifically, edit client/rom by changing 192.12.69.186 in the first arp line to the IP address of machine A, and by changing 192.12.69.35 in the second arp line to the IP address of machine B. Edit server/rom in exactly the same way.

Note that A and B can be the same machine, in this case, the same IP address is used in the rom file for both machines.

7. Put /usr/xkernel/bin/solaris-sparc and /usr/xkernel/bin in your search path. They should appear before /bin and /usr/bin in order to pick up GNU make before the standard Unix make. If you are building the *x*-kernel on a platform other than Solaris, then substitute sunos-sparc, osf1-alpha, irix-mips, linux-alpha, or linux-x86 for solaris-sparc, as appropriate. Also, make sure the GNU C compiler (gcc) is in your search path. (At Arizona, gcc can be found in directory /usr/local/bin.)

8. Build libraries necessary for your *x*-kernel. This could take quite a while.

```
cd /usr/xkernel/user_level/build/solaris
make system
```

Note that you can skip this step if you are not the first person to build an *x*-kernel for this platform.

9. You are now ready to build the *x*-kernel.

```
cd /usr/xkernel/user_level/build/solaris
make compose
make depend
make
```

This sequence should result in the creation of the binary file xkernel in your build directory. Note that you may see what appear to be error messages about missing files (e.g., Makefile.local and DEPS/Makedep.*) when you run make compose. These warnings can be ignored.

10. You are now ready to run the *x*-kernel. This will involve being logged onto both machines A and B. This is easiest to do by using a window-based interface and opening a separate terminal (shell) window on both A and B. First, on machine A type:

```
cd /usr/xkernel/user_level/build/solaris/server
../xkernel -s
```

This starts a version of the *x*-kernel on machine A that will act as a server for this run. Second, on machine B type:

```
cd /usr/xkernel/user_level/build/solaris/client
../xkernel -c128.1.2.3
```

This starts a version of the *x*-kernel on machine B that will act as a client for this run. Note that you have to start the server before the client. When the run is over, you can kill the *x*-kernel by typing Control-C on both A and B.

The run involves the client establishing a TCP connection to the server, with a sequence of different-sized messages then sent back and forth between the client and the server. The time it takes to exchange 100 messages of each size between the client and the server will be printed out. For example, the output on the client will consist of a sequence of lines that look something like this:

```
Protocol: TCP
Time: Fri Jun  6 16:16:11 1997
Host: cicada.CS.Arizona.EDU
Participant: client
Round Trips: 100

Message Length (bytes): 1
Times (sec):
select: Interrupted system call
     0.061818
     0.060986
     0.063218
```

This means it took 0.061818 seconds (or 61ms) to send 100, 1-byte messages between the client and server machines, for an average of .61ms per round-trip.

Troubleshooting

Many of the problems encountered when running an *x*-kernel turn out to be configuration problems. The following are three of the most common symptoms, and some things you might try if these symptoms occur.

- The *x*-kernel aborts with the message

  ```
  could not open protocol table file prottbl
  ...
  ```

  Make sure you edited the last line of graph.comp correctly in step 5, and that /usr/xkernel/etc/prottbl.std exists and is readable.

- The *x*-kernel hangs (doesn't abort) and repeatedly prints out the message

  ```
  ARP: Could not get my ip address for interface eth (still trying)
  ```

  Make sure you edited the two rom files correctly in step 6, and that you started the server on machine A and the client on machine B in step 10.

- Both the client and server initialize, but then seem to hang. You will know that the server has successfully initialized when it prints the message

  ```
  tcp timing test
  I am the server
  ```

  and that the client has successfully initialized when it prints the message

  ```
  tcp timing test
  I am the client
  Sending (1) ...
  msg lengh: 1
  ```

  Make sure you edited the two rom files correctly in step 6.

Other Things to Try

Now that you are able to build a vanilla system, you can start experimenting with different optional configurations. The following gives a list of things you might try. Consult the Programmer's Manual for more information on each item.

- Edit graph.comp in your build directory by adding the string

  ```
  trace=TR_MAJOR_EVENTS
  ```

  to the end of each of the following four lines

  ```
  name=eth protocols=simeth;
  name=arp protocols=eth;
  name=ip protocols=vnet;
  name=tcp protocols=ip;
  ```

  For example, the first line would become

  ```
  name=eth protocols=simeth trace=TR_MAJOR_EVENTS;
  ```

Similarly with the other lines. Once this edit is done, repeat steps 9 and 10 from above. This change will increase the amount of tracing information that is output when the *x*-kernel runs. This output will now include events from TCP, IP, ARP, and the Ethernet driver (ETH). To see an even greater level of trace detail, change TR_MAJOR_EVENTS to TR_FUNCTIONAL_TRACE in each of the above lines, and repeat steps 8 and 9 again.

- Edit the Makefile in your build directory by commenting out the line

  HOWTOCOMPILE = DEBUG

  and un-commenting the line

  #HOWTOCOMPILE = OPTIMIZE

  Then repeat steps 9 and 10 from above. (You will also need to repeat step 8 if this is the first time the *x*-kernel has been built in OPTIMIZE mode on this platform.) This will build an optimized version of the *x*-kernel that does not include the trace statements that generated much of the output you saw before. Typically, you want to run the *x*-kernel in OPTIMIZED mode when you care about the performance measurements it generates.

- Edit graph.comp to include a different set of protocols. For example, instead of configuring TCP into your kernel, you could configure in UDP. This is done by replacing the lines

  name=tcp protocols=ip;
  name=tcptest protocols=tcp;

  with the lines

  name=udp protocols=ip;
  name=udptest protocols=udp;

  Then repeat steps 9 and 10 from above. You can also play with OPTIMIZE versus DEBUG mode, as well as the trace level while in DEBUG mode.

  Note that /usr/xkernel/user_level/build/Template contains various example_* subdirectories, each of which contains configuration files for various protocol stacks you can try. For example, try running an RPC protocol stack, or the MSP or SWP protocols.

- Repeat the entire process—steps 2 through 10—for some other Unix platform, for example, Linux on x86s rather than Solaris on Sparc.

- Implement a new protocol, and then configure an instance of the *x*-kernel that tests this new protocol. The best way to get started on this process is to copy an existing protocol into your build directory, and then edit this protocol to implement the desired functionality. We recommend protocol ASP as a template for doing this because it is so simple. ASP is implemented by the three files in /usr/xkernel/protocols/asp/asp.c, /usr/xkernel/protocols/asp/asp_internal.h, and /usr/xkernel/include/prot/asp.h. (Copy all three of these files to your build directory, and change their names to reflect the name of your protocol.) You will also need a test protocol that sits on top of your new protocol; we recommend /usr/xkernel/protocols/test/asptest.c for this purpose. Finally, you will need to edit graph.comp to configure your new protocol into the *x*-kernel.

  Of course, the hard work is to write the *x*-kernel code that implements the desired functionality. You will need to study the *x*-kernel Programmer's Manual [1] and Tutorial [2] to help you with this task. It will be especially helpful to study protocol ASP, which is described in detail in the Tutorial. We recommend you start with something simple, like changing the header format.

References

[1] Network Systems Research Group, Department of Computer Science, University of Arizona. *x-kernel Programmer's Manual (Version 3.3)*, Jan. 1996.

[2] L. L. Peterson, B. S. Davie, and A. C. Bavier. *x-kernel Tutorial*. Network Systems Research Group, Department of Computer Science, University of Arizona, Jan. 1996.