Exopc Getting Started Guide

Parallel and Distributed Operating Systems Group

MIT Laboratory for Computer Science
545 Technology Square
Cambridge MA 02174

exopc-request@amsterdam.lcs.mit.edu
http://www.pdos.lcs.mit.edu

June 4, 2017

1 Requirements

Exopc is not yet self-hosting. This means that it must be compiled under a different operating system than itself. Currently, exopc must be built under OpenBSD or Linux, though FreeBSD or NetBSD probably work too. We use either OpenBSD 2.2 with gcc 2.7.2 or linux 2.0.x with libc6 and gcc 2.7.2. Exopc uses the same binary format as OpenBSD. This means that if you're compiling under linux you have to use a cross compiler that generates OpenBSD binaries. We have included binaries for gcc 2.7.2 setup to cross compile in this distribution. To use this cross-compiler, untar tools/linux-cross/cross-tools-libc6.tgz under /usr/local/openbsd-cross.

Booting is poorly supported right now. Either you have to boot with OpenBSD's boot blocks (not supplied) or with the supplied DOS boot-loader. OpenBSD's boot blocks expect the kernel to be in an OpenBSD filesystem. Other boot-loaders should work. It would be nice to find one that didn't require us to have an OpenBSD filesystem and which could load both linux and OpenBSD kernels.[1]

Exopc currently only supports the following hardware: NCR 810/815/875 SCSI controllers and SMC Elite16, EtherEZ[2], EtherPower 10/100 ethernet cards (not made anymore) and dlink-500TX's (or other tulip based cards). You have to have an ethernet card, but a local disk is optional. At least a Pentium processor is required. Any color display adapter should work.

2 Building

Building the system is pretty straight-forward. Exopc uses gmake and you should be able to go into a directory and type "gmake" either by itself or with the targets "install" or "clean". "clean" does the normal thing while "install" copies files from the build area to the root filesystem of the exopc machine.

[1] A boot-loader called "grub" may be just this, but we haven't tried it out yet

[2] You must configure this card using the card's ezsetup.exe program to set it as follows: I/O base=280, IRQ=2, RAM Addr=d0000, wait states=no, net connection=1, rom disabled=yes

Remember that exopc mounts a root filesystem via NFS when it starts up. “install” copies files into this root filesystem on the server. In order to tell the makefiles where to copy files you have to set the `EXODEST` environment variable. This points to the top of a directory-tree that will later be your exopc root directory. For example, you may have /home/pinckney/exopc as the top of you exopc source directory and point `EXODEST` to /home/pinckney/exopc-root. Then, when you run “gmake install” from /home/pinckney/exopc, /home/pinckney/exopc-root will be populated with subdirectories bin/, etc/, usr/ etc and binaries will be copied into the correct places.

In the special case that your EXODEST points to a directory on the same disk as you are building on, you can use symlinks instead of “install”. Two variables in GNUmakefile.global, `LNPROGINSTALL` and `LNLIBINSTALL`, controll whether “ln” or “install” is used. By default, “ln” is used so if you’re building and installing on different disks you need to change this.

The following lists some caveats of the current build process:

- Technically “gmake install” should build any out of date programs and then install them, but this doesn’t always seem to work. To be on the safe-side, do a “gmake; gmake install”.
- Until you understand all the cross dependencies between different parts of the system, do full builds of the entire source tree.
- Most binaries are linked against the shared library libexos.so. Thus, if you change any of the libraries, rebuild the shared library in bin/shlib and then rebuild *all* of the binaries.
- The first program is special-cased and resides in ebin/. The section on booting contains more information about selecting which of these programs to use as the first process. In any event, the environment variable INITPROG can be set to choose among rconsoled, kbdinit, and bootp. If this program changes, you need to rebuild the kernel in sys/ to relink the first program into the kernel.

3 Booting exopc

Exopc currently boots much like a diskless workstation. Because we do not have crash recovery for local storage, we must ”start from scratch” each time the system starts. This means that there is no local root partition. Once the kernel loads, an NFS directory is mounted as the root of the exopc system. Later, a local filesystem may be created and mounted.

The kernel is typically loaded using OpenBSD’s boot blocks which means the kernel itself has to be stored in an OpenBSD filesystem. Further, since the kernel itself is fairly minimal, it has no concept of filesystems. Thus the kernel cannot load the first program. Instead, the first program is encoded into the kernel. On booting, the kernel loads this first program from the kernel’s data segment into a new process and starts the process running. This first process is then responsible for mounting the root partition via NFS and loading any subsequent programs.

This first process needs to know certain information before it may mount the root such as its ip address, the NFS root path and server address, the netmask, which ethernet interface to use, and possibly a gateway address. This information can be provided in three different ways:

- **keyboard:** you type in the relevant information each time you boot the system. This is typically only used while you are getting the system setup.
- **hardcoded:** you can hardcode the information into the first process.

- **BOOTP:** you can setup the first process to use BOOTP to get all the relevant information.

When building the kernel you must select which of these three options you wish to use. See the section below on building for more information.

Once the first program has been started and local configuration information gathered, /etc/rc.local is run. This is typically a shell script that starts various system daemons as well as typically building and mounting a local filesystem. The default rc.local should not need to be changed.

3.1 Using the keyboard to enter boot-time information

If `kbdinit` is selected as the first process, it will prompt you with the following questions:

```
Please Enter your IP Address:
Please Enter your netmask:
Please Enter interface of this IP address [%s]:
Please Enter your Gateway [press return if you don't care]:
Please Enter IP Address of your NFS Server:
Please Enter name your NFS directory:
```

The NFS directory you mount should be the same directory that is pointed to by your EXODEST env variable.[3] The building section has more information on EXODEST.

3.2 Hardcoding boot-time information

This is the original method of configuring boot-time information and remains primarily for compatibility. For historical reasons the program that implements it is called rconsoled. All local configuration info. is stored in the file include/exos/site-conf.h. Most of the entries are self-explanatory. The one that is not is `MY_IP_MAP`. Each line of it is of the form:

```
/* 1 */ {"panam", {18,26,4,38}, {0x0,0x0,0xc0,0x81,0xe0,0xe2}, 1},
```

The first field is the machine's name. The second field is the machine's ip address. The third field is the ethernet address of the primary interface. The fourth field is the interface number. The last entry in this table should consist of all zero entries to mark the end.

You must have at least two entries in `MY_IP_MAP`: one for the exokernel machine itself and one for the root NFS server. The former is looked up by its ethernet address and the latter by its name.

3.3 Using BOOTP for boot-time information

This is the preferred way of configuring the first process. It obtains all information via the BOOTP protocol. To use it, enable the bootp daemon on some machine by uncommenting the line that starts with "bootps" in the /etc/inetd.conf file and restarting inetd by sending it SIGHUP signal (e.g., "kill -HUP").

All the information about the machine's ip address, netmask, nfs root is obtained via the BOOTP protocol. To use it add an entry to your /etc/bootptab[4]. Here's an example of what ours looks like:

[3](note that OpenBSD machines want the exact directory that you are mounting to appear on the /etc/exports file, or you can put the first directory on that partition followed by the "-alldirs" flag such that ALL directories in that partition are exported.

[4]make sure you use the ip address of your nfs server since DNS is not setup at mount time

```
avoch::ha=0000c0d0a54e:\
:sm=255.255.255.0:gw=18.26.4.1:
:ef="NFS 18.26.4.40:/disk/av0/hb/tree":\
:ip=18.26.4.33:\
:to=auto:
```

3.4 Reloading kernels over the network

Once the system is up and running, a new kernel may be overlayed on top of the existing one using the exokernel's `loadkern` program over NFS. Specifically, running `loadkern <xok image>` from the command line (on an exokernel system) will reboot the system, loading the specified kernel image. This utility removes the need to boot into OpenBSD every time you build a new kernel and want to run it. Just leave your old kernel on the OpenBSD partition, the boot blocks will load this old one, and when the system is up you can reboot it with the new kernel over NFS. (The same is true of other booting methods, like via DOS or floppy.)

The three initial programs discussed above all will check to see if the latest installed kernel is newer than the one that was booted. If it is, these initial programs will automatically reboot using the newest kernel image. This works by checking /boot/xok.osid in the root NFS partition. If the contents of this file don't match the os_id of the running kernel, then /boot/xok will be reloaded on top of the existing kernel. If the file /boot/.noboot exists, then this automatic reloading will be disabled.

Of late, we have observed iffy behavior in the auto-detection/negotiation between SMC EtherPower 10/100 cards and a SMC TigerSwitch 100 when reloading a second kernel during the boot process.

3.5 Booting from a floppy

The xok kernel can be booted from a DOS partition rather than an OpenBSD file system. However, this is not self-contained booting. You still need the rest of the xok files accessible over NFS. The benefit of this method is that you don't need an OpenBSD file system to store a copy of your xok kernel on. You can almost think of this as a network boot loader specially for xok.

You'll need a DOS floppy with nothing on it except command.com and whatever system hidden files go with it. In other words, format a floppy with DOS and run "sys a:" to transfer the minimum required DOS files for booting DOS.

Then place "boot.com" in the root directory of the floppy. This file is provided by OpenBSD for booting OpenBSD kernels from DOS. A copy has been included in the tools/floppy_boot directory of the xok distribution.

Next, once you've successfully compiled/built xok, place the kernel image (sys/obj/xok) onto the floppy as well. The common mechanism for doing this on linux or BSD systems is to use the mtools utilities (e.g., mcopy). At the moment, the kernel is small enough to fit onto the floppy. Hopefully, it won't get too big to fit on there any time soon. Actually, my latest kernel is too big, but a small DOS partition on the hard disk works fine.

Now, if you boot with that floppy you can type "boot xok" at the DOS prompt and xok should load.[5].

If you wish, you can create an autoexec.bat file on your floppy to automatically run "boot xok" every time. Simply create a file called autoexec.bat in the root of the floppy with "boot xok" as the first line.

[5]Remember that once loaded it will check in EXODEST/boot/ to see if there's a newer version of the kernel and automatically load that instead

3.6 Misc Unix environment setup

You may need to edit some of the files in tree/etc/, notably rc.local, hosts.conf and passwd. Files under tree/ are copied over to exopc root filesystem when you do a "gmake install". Because we use the OpenBSD libc our password setup follows OpenBSD's. This means you will have to add/modify accounts by editing master.passwd and then running tree/etc/rebuild_passwd. All of the passwords in that file are currently "exopc". If you want to add a new home directory edit directories/GNUmakefile to create one.

You need to change resolv.conf to point to your name server and to set the domain search order. You also need to change rc.local to tell gettime to get the time from one of your own computers rather than ours.