EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

2390 EMAC, Way
Carbondale, IL 62902
Phone: (618) 529-4525
Fax: (618) 457-0110
http://www.emacinc.com

Remote Debugging using EMAC's SIB Linux

EMAC's PC-SBC product line consists of hardware equivalent to that of an x86 PC. You can develop programs on a Linux-based x86 desktop computer with no modification or need for cross-compiling.

Statically Linking Against Libraries

You need to be aware that your application may eventually be linked against libraries that may not be included with the SBC. In this case, you can a) add the necessary files into the `/usr/lib` (preferred) or the `/lib` directories and run the `ldconfig` command (not included on the SiB), or b) link your application with static versions of the required libraries. Dynamic libraries typically end with a `.so` extension, and static libraries end with `.a`. EMAC Linux includes the following libraries as part of the standard build, in the `/lib` directory:

```
libc6-2.2.3.so
ld-2.2.4.so
libcrypt-2.2.4.so
libdb1-2.2.4.so
libdl-2.2.4.so
libncurses.so.4.2
libncurses.so.5.0
libnsl-2.2.4.so
libnss_compat-2.2.4.so
libnss_db-2.2.so
libnss_dns-2.2.4.so
libnss_files-2.2.4.so
libreadline.so.4.2
libresolv-2.2.4.so
libutil-2.2.4.so
```

The following libraries are included in the "extra C++ libraries" module:

```
libg++-2-libc6.1-1.8.2.so
libm-2.2.4.so
libpthread-0.9.so
libstdc++-3-libc6.2-2-2.10.0.so
```

Building and Debugging

Debugging on the SBC is accomplished using the `gdb` program running on a Linux-based desktop PC. Its matching component on the SIB is a binary named `gdbserver`. `gdbserver` allows one to develop code on a host machine, and still monitor the program running in its native environment on the SIB. You can use `gdbserver` over a ethernet link, or a serial link using a null modem cable. Whichever link you use, you will need a separate connection to the SIB in order to issue commands-one for your gdb session, the other to issue other commands to the SIB. The rest of this document assumes that you have established an ethernet connection to the SIB.

Selecting a standard location for your binaries

When you are ready to test your newly created binary, use the FTP mechanism to copy the binary over to the SIB. Use the `chmod` command to change the ownership so that the binary is executable, and move the binary to the desired location. The standard locations for binaries on a UNIX system are as follows:

/sbin	System binaries, available as soon as the system starts, before other filesystems are mounted. Most of the time, this directory is not in a normal user's path.
/bin	Binaries for root and non-root users, available as part of the base system. Again, this directory should be available before any separate filesystems are mounted.
/usr/sbin	Noncritical system binaries. This may be mounted on a separate partition, so do not put binaries here which are critical to checking filesystems, etc.
/usr/bin.	Noncritical binaries available for root and non-root users
/usr/local/sbin	Binaries for root user, added in addition to any vendor-supplied ones.
/usr/local/bin	Binaries for non-root users, added in addition to any vendor-supplied ones.

System V Init Scripts

EMAC Linux uses System V init scripts. This is a directory structure that allows daemons and programs to start when the SIB is started. A directory called `/etc/init.d` holds scripts that facilitate starting and stopping programs. These scripts, at minimum, take the commandline arguments `start` and `stop`. They may additionally take other arguments such as `reload` and `restart`, depending upon the program's nature. The directories `/etc/rc0.d` through `/etc/rc6.d` contain symbolic links to the scripts in `/etc/init.d` and allow the SIB's startup scripts to start and stop the programs, depending upon needs. The directories `/etc/rc0.d` through `/etc/rc6.d` correspond to different runlevels, which can be utilized to offer different SIB functionality depending upon which runlevel setting is made the default in `/etc/inittab`. Runlevels 0 and 6 are run on system shutdown and reboot, respectively. As shipped from EMAC, the default runlevel is 2, which corresponds to the symbolic links in directory `/etc/rc2.d`. When a symbolic link is made, it begins with the letter `S` and a number, typically in the range `00` to `99`. The links are executed numerically; when any two scripts share a number, they are executed alpabetically. An example is the symbolic link to start the `syslog` and `klog` daemons (perform an `ls -l` command to see the symbolic link's target):

```
/etc/rc2.d/S10sysklogd -> /etc/init.d/sysklogd
```

This instructs the startup scripts to start the `sysklogd` script located in the `/etc/init.d` directory with the `start` parameter as the first commandline argument. In certain directories, such as the `/etc/rc6.d` and `/etc/rc0.d` directories, you may notice some symbolic links begin with a K. This instructs the startup scripts to shut down certain programs upon initialization of a runlevel. This actually causes the startup scripts to run the script with `stop` as the first and only commandline parameter. If a script accepts other parameters such as `restart` or `reload`, these must be executed by hand at a command prompt.

Debugging your code

The binary object code can be stripped to remove debugging symbols, before moving it to the SIB. `gdbserver` does not need them. In order to invoke `gdbserver` on an ethernet connection, issue the following (preferrably as the root user) on the target system:

```
target> gdbserver host:2345 <prog_name> <args>
```

this will start a process listening on TCP port 2345. This is a safe setting; some network configurations may need to change this value. If you're using a serial connection, the command would be (ttyS1 is COM2):

```
target> gdbserver /dev/ttyS1 <prog_name> <args>
```

if your binary is already running, you can attach to it (PID is the process ID of your binary, as reported by the `ps` command):

```
target> gdbserver /dev/ttyS1 --attach <PID>
```

Of course, `prog_name` and `args` are the name of the executable and any arguments to it. If you are using a serial debugging connection, be aware that you cannot share a COM port between the SIB's serial console and a debugging console.

On the development platform, you will need an unstripped binary compiled with the proper debugging symbols. For the `gcc` compiler, you need to specify the `-g` option on the commandline during the build process. Start `gdb` and load the binary:

```
host> gdb <prog_name>
```

next, you need to connect to the target machine:

```
gdb> target remote /dev/ttyS1
```

for a serial connection, or

```
gdb> target remote target_host_name:2345
```

Again, this connects on TCP port 2345; You may need to change this value if there is an application running on the SIB which already uses TCP port 2345.

Using GDB

Refer to the attached GDB manpage for further instructions. You should now have the ability to debug your application as it runs remotely.

References:
GDB Manpage
GDBserver Manpage