Tips and Tricks for Embedded Linux

"error in loading shared libraries: undefined symbol: __libc_start_main"

Problem: You have build an application binary which is linked against shared libraries, and v try to run it on the target system you get the error message "undefined symbol: __libc_start_m

Explanation: Most probably the versions of the shared libraries you are using in your target e are not compatible with the versions in your build (cross development) environment. This pro typical when you are using libc-2.1.x libraries when building the binaries, but your target envi still using libc-1.xx libraries.

Solution: You must make sure to run your binaries in an environment which contains only th that were used when linking the binaries. There are two easy ways to to that:

- Use a ramdisk image: if you are running from a ramdisk image (initrd), the make sure y only the correct versions of the shared libraries and the dynamlic loder (/lib/ld-...) in yo image.
- When running for instance with the root filesystem mounted over NFS, you can use "ch up your own private environment. Here is a mini-HOWTO:
  1. Make sure you are running with a **statically** linked shell, for instance "/bin/sash". a dynamically linked shell like "/bin/bash".
  2. Create a new directory where you can build your own test environment.
  3. Put your binaries and all needed libraries (including the dynamic loader) in a dire hierarchie below your newly created test directory.
  4. Put a statically linked shell in your new directory hierarchie.
  5. Use "chroot *test_directory static_shell*" to change to your test enviornment. Cha root directory. Run your tests.

  Example:

```
...
Stand-alone shell (version 2.1)
> mkdir /tmp/testroot
> mkdir /tmp/testroot/lib /tmp/testroot/bin
> cp demo /tmp/testroot/bin/
> cp /bin/sash /tmp/testroot/bin/sh
> cp /CDK/lib/libc-2.1.2.so /tmp/testroot/lib/
> cp /CDK/lib/ld-2.1.2.so /tmp/testroot/lib/
> cd /tmp/testroot/lib/
> ln -s libc-2.1.2.so libc.so.6
> ln -s ld-2.1.2.so ld.so.1
> ls -lR /tmp/testroot
total 2
drwxr-xr-x   2 root     root         1024 Feb 15  2000 bin
drwxr-xr-x   2 root     root         1024 Feb 15  2000 lib
/tmp/testroot/bin:
total 431
-rwxr-xr-x   1 root     root         7044 Feb 15  2000 demo
-rwxr-xr-x   1 root     root       430508 Feb 15  2000 sh
/tmp/testroot/lib:
total 1799
-rwxr-xr-x   1 root     root       181105 Feb 15  2000 ld-2.1.2.so
lrwxrwxrwx   1 root     root           11 Feb 15  2000 ld.so.1 -> ld-2.1
-rwxr-xr-x   1 root     root      1650689 Feb 15  2000 libc-2.1.2.so
lrwxrwxrwx   1 root     root           13 Feb 15  2000 libc.so.6 -> libc
> chroot /tmp/testroot /bin/sh
Stand-alone shell (version 2.1)
> cd /
> -ls -l . *
.:
drwxr-xr-x  4 0        0            1024  Feb 15 2000  .
drwxr-xr-x  4 0        0            1024  Feb 15 2000  ..
drwxr-xr-x  2 0        0            1024  Feb 15 2000  bin
drwxr-xr-x  2 0        0            1024  Feb 15 2000  lib
bin:
drwxr-xr-x  2 0        0            1024  Feb 15 2000  .
drwxr-xr-x  4 0        0            1024  Feb 15 2000  ..
-rwxr-xr-x  1 0        0            7044  Feb 15 2000  demo
-rwxr-xr-x  1 0        0          430508  Feb 15 2000  sh
lib:
drwxr-xr-x  2 0        0            1024  Feb 15 2000  .
drwxr-xr-x  4 0        0            1024  Feb 15 2000  ..
-rwxr-xr-x  1 0        0          181105  Feb 15 2000  ld-2.1.2.so
lrwxrwxrwx  1 0        0              11  Feb 15 2000  ld.so.1
-rwxr-xr-x  1 0        0         1650689  Feb 15 2000  libc-2.1.2.so
lrwxrwxrwx  1 0        0              13  Feb 15 2000  libc.so.6
> demo
```

"error in loading shared libraries: libc.so.6: cannot open sh object file: No such file or directory"

Problem: You have build an application binary which is linked against shared libraries, and w try to run it on the target system you get the error message "cannot open shared object file: No or directory"

Explanation: By default, the linker tries to be helpful and saves the path of the libraries it use linking in the binary. This is quite useful under normal conditions, but it does not work when working in a cross development environment which is different from the target environment. HardHat Linux by MontaVista has this problem: in the cross environment the libraries are fou

```
/opt/hardhat/devkit/ppc/8xx/powerpc-hardhat-linux/lib/
```

If you put your libraries on the target in the "/lib" directory, they will not be found because the

linker will be searching for "/opt/hardhat/devkit/ppc/8xx/powerpc-hardhat-linux/lib/".

Solution:

1. MontaVista uses a workaround: they put a symbolic link in their target which makes it the cross envrionment. On your target, create the following directories:

   ```
   /opt
   /opt/hardhat
   /opt/hardhat/devkit
   /opt/hardhat/devkit/ppc
   /opt/hardhat/devkit/ppc/8xx
   ```

   In the "/opt/hardhat/devkit/ppc/8xx" directory, create a symbolic link with that name "powerpc-hardhat-linux" which points to the root directory:

   ```
   /opt/hardhat/devkit/ppc/8xx/powerpc-hardhat-linux -> /
   ```

   This can be eaily done using the folowing commands:

   ```
   mkdir -p /opt/hardhat/devkit/ppc/8xx
   ln -s / /opt/hardhat/devkit/ppc/8xx/powerpc-hardhat-linux
   ```

   Note: Older versions of HardHat Linux used the name "powerpc-linux" for the link.
2. We think it is better to avoid the problem in the beginning instead of fixing it later: whe your binaries, tell te linker where it will find the shared libraries on the target. If your sl libraries are (as usual) in the "/lib" directory, add the following option to the GCC com when linking your binary:

   ```
   -Wl,-R/lib
   ```

   In our own CDK, the linker is configured to use this option by default.

Home j
 Last update: 2000