Building User Space

Zilogic Systems

1. Commands and Utilities

- Shell commands and utilities are provided by BusyBox.

1.1. Obtaining BusyBox

- Download BusyBox from http://busybox.net/downloads/
- Extract the tar ball using

```
tar -jxf busybox-1.13.1.tar.bz2
```

1.2. Configuring BusyBox

- BusyBox is highly configurable, the commands to build and features in each command can be fine tuned during the configuration process.
- BusyBox can be configured using

```
$ make menuconfig
```

- A menu interface is provided through which various configuration options can be modified.

1.2.1. Specifying the Cross Compiler

- The following option has to be set to the compiler prefix. The compiler prefix is `arm-none-linux-gnueabi-`.
- The trailing hyphen in the prefix is required.

```
Busybox Settings > Build Options > Cross Compiler prefix
```

1.2.2. Static Busybox

- Sometimes it is useful to compile busybox as a static binary, without run-time dependency on another library.
- To compile statically turn on the following option.

```
Busybox Settings > Build Options > Build BusyBox as a static binary
```

1.2.3. Specifying Commands

- BusyBox comes with the default configuration, which includes most of the commands available.
- Unselect commands that are not required.

2. Building Busybox

- BusyBox can be built using

```
$ make
```

- Build can be accelerated by specifying the `-j` option. Usually `make` builds all the files sequentially. But with `-j` option, a parallel build can be performed. The argument to `-j` specifies the no. of parallel compilations to be performed.

```
$ make -j 8
```

- After BusyBox is built, the required links can be created under `_install/` using

```
$ make install
```

- These files can be copied to the target filesystem using

```
$ ROOTFS=/path/to/rootfs
$ cd _install
$ cp -a * $ROOTFS/
```

3. Device Files

- Certain device files are required during system boot up.
- They can be created by passing `-D <filename>` option to `genext2fs`.
- The filename passed to the `-D` option contains a device table, which specifies the device nodes to be created.
- A sample device table is given below.

```
# name       type mode uid gid major minor start inc count

/dev/null     c    666  0    0    1    3    -    -    -
/dev/console  c    660  0    0    5    1    -    -    -
/dev/ttyS0    c    660  0    0    4    64   -    -    -
```

4. Configuration Files

- Minimum required configuration `/etc/inittab` and `/etc/init.d/rcS`.

/etc/inittab.

```
::sysinit:/etc/init.d/rcS
::respawn:/bin/sh
```

/etc/init.d/rcS.

```
mount -t proc none /proc
```

- Make sure `rcS` is executable.

```
chmod +x ./etc/init.d/rcS
```

5. Libraries

- No libraries have been copied since, busybox is statically compiled. Programs with dependencies on shared object files will not execute.
- This can be verified by compiling a hello world program, and executing on the target.
- The following libraries are required.

```
libc.so.6
ld-linux.so.3
```

- These libraries are present in the toolchain directory. The toolchain is located in this path.

```
/usr/share/gcc-arm-linux/
```

- Within the toolchain directory the directory of the libraries is

```
arm-none-linux-gnueabi/libc/lib/
```

- Copy the libraries on to the root filesystem.

```
$ cd $ROOTFS
$ mkdir lib
$ cd lib
$ cp /usr/share/gcc-arm-linux/arm-none-linux-gnueabi/libc/lib/libc.so.6 .
$ cp /usr/share/gcc-arm-linux/arm-none-linux-gnueabi/libc/lib/ld-linux.so.3 .
```

6. Building Applications

- Overtime Unix-like systems have diverged in minor ways and to write a program that can be compiled on all Unix systems has become tedious.
- For example, Unix systems differ in the header file names. Example: The header file for string functions is `string.h` in certain systems and `strings.h` in certain other systems.
- Another example is presence of certain system calls. Some Unix systems provide additional system calls that can provide better performance, compared to an existing system call. Example: `inotify()` system call to notify change in files and directories, is specific to Linux.
- GNU Autotools solves the problem of portability of applications.
- Applications need to be configured before they are built.
- Most free software use the GNU Autotools for configuration and build.
- Configuration is primarily used for
  1. Automatic Feature Selection
  2. Manual Feature Selection

6.1. Automatic Feature Selection

- The GNU Autotools provides a configuration facility through which the features present in a system can be discovered and the results of the discovery is made available to the application through a header file `config.h`.
- The application uses the definitions in the header file to provide alternatives appropriately.

6.2. Manual Feature Selection

- Just as busybox allows configuration of features through menu interface, other programs also require some form of manual feature configuration.
- The configuration facility provided by autotools can also be used to specify what features in the software required and what are not.

6.3. Building Programs

- Programs that use Autotools can be built using the following sequence of commands.

```
$ ./configure
$ make
$ make install
```

- The configure script does automatic feature selection.
- Manual feature selection can be done by passing options to the configure script.
- To cross compile programs that use the Autotools infrastructure, the configure script has to be invoked with options to specify the target system architecture, called `host` system in Autotools parlance.
- To specify the `host` system, the prefix of the cross compiler is to be specified. In this case without the trailing hyphen.
- It is also recommended the `build` system also be specified during cross compilation. For PCs, the build system is usually represented as

```
i686-pc-linux-gnu
```

- So the command sequence turns into

```
$ ./configure --host=arm-none-linux-gnueabi --build=i686-pc-linux-gnu
$ make
$ make install
```

- Most programs when built in this manner, assume that they will be installed in `/usr/local`. For example, when the program wants to open it's configuration file, it does it as

```
    fd = open("/usr/local/etc/myconfig");
```

- This is what is usually required when installing software on a desktop system. The package manager does not interfere with files present in `/usr/local/`.
- But to build package so that they use the `/usr` or `/` directory, the `prefix` option can be passed to the configure script. To place the programs in `/usr` instead of `/usr/local` the `prefix` option can be passed as shown below.

```
$ ./configure --prefix=/usr
```

- To place the program under `/`, the prefix option can be passed as shown below.

```
$ ./configure --prefix=
```

- When `make install` is performed we would like the programs to be copied to the target root filesystem instead of the build system. This can be done by specifying the `DESTDIR` variable during `make install`.

```
$ make install DESTDIR=$ROOTFS
```

- Combining all the above, we get

```
$ ./configure --host=arm-none-linux-gnueabi --build=i686-pc-linux-gnu --prefix=/usr
$ make
$ make install DESTDIR=$ROOTFS
```

References

- IBM DeveloperWorks, Linux initial RAM disk (initrd) overview. URL: http://www.ibm.com/developerworks/linux/library/l-initrd.html

- The GNU Autoobook Autoconf, Automake, and Libtool. URL: http://sources.redhat.com/autobook/autobook/autobook_13.html