Cross-platform development workspace:

bootldr:	target bootloader (s)
build-tools:	toolchain build packages and sources
debug:	debugging tools
doc:	project documentation
images:	binary images ready to be used on target
kernel:	sources and build directories for target kernels
project:	your own custom code for the target
rootfs:	root filesystem as seen on the target
sysapps:	sources for target's system applications
tmp:	temporary data and experiments
tools:	toolchain and all other tools required to build software for the target.

Workspace script:

```
export PROJECT=example-sys
export PRJROOT=/home/karim/${PROJECT}
export TARGET=powerpc-linux
export PREFIX=${PRJROOT}/tools
export TARGET_PREFIX=${PREFIX}/${TARGET}
export PATH=${PREFIX}/bin:${PATH}
cd $PRJROOT
```

To use this script:

```
$ . devex
```

GNU toolchain version combinations:

http://www.embeddedtux.org/gnu-tools.html

Creating GNU toolchain components directory:

```
$ cd ${PRJROOT}/build-tools
$ mkdir build-binutils build-boot-gcc \
> build-glibc build-gcc
```

Installing kernel headers:

```
$ cd ${PRJROOT}/kernel
$ tar xvjf linux-2.4.20.tar.bz2
$ cd linux-2.4.20
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- \
> menuconfig
$ mkdir -p ${TARGET_PREFIX}/include
$ cp -r include/linux/ \
> ${TARGET_PREFIX}/include
$ cp -r include/asm-ppc/ \
> ${TARGET_PREFIX}/include/asm
$ cp -r include/asm-generic/ \
> ${TARGET_PREFIX}/include
```

Building and installing binutils:

```
$ cd ${PRJROOT}/build-tools
$ tar xvzf binutils-2.11.2.tar.gz
$ cd build-binutils
$ ../binutils-2.11.2/configure \
> --target=$TARGET --prefix=${PREFIX}
$ make
$ make install
```

Bootstrap compiler setup:

```
$ cd ${PRJROOT}/build-tools
$ tar xvzf gcc-2.95.3.tar.gz
$ cd build-boot-gcc
$ ../gcc-2.95.3/configure --target=$TARGET \
> --prefix=${PREFIX} --without-headers \
> --with-newlib --enable-languages=c
$ make all-gcc
$ make install-gcc
```

C library setup:

```
$ cd ${PRJROOT}/build-tools
$ tar xvzf glibc-2.2.1.tar.gz
$ tar -xvzf glibc-linuxthreads-2.2.1.tar.gz \
> --directory=glibc-2.2.1
$ cd build-glibc
$ CC=powerpc-linux-gcc \
> ../glibc-2.2.1/configure \
> --host=$TARGET --prefix="/usr" \
> --enable-add-ons \
> --with-headers=${TARGET_PREFIX}/include
$ make
$ make install_root=${TARGET_PREFIX} \
> prefix=”” install
```

Link script: ${TARGET_PREFIX}/lib/libc.so:

```
/* GNU ld script
   Use the shared library, but some functions are only in
   the static library, so try that secondarily.  */
GROUP ( /lib/libc.so.6 /lib/libc_nonshared.a )
```

Remove the “/lib/”

Full compiler setup:

```
$ cd ${PRJROOT}/build-tools/build-gcc
$ ../gcc-2.95.3/configure --target=$TARGET \
> --prefix=${PREFIX} --enable-languages=c,c++
$ make all
$ make install
```

uClibc configure, build, and install:

```
$ cd ${PRJROOT}/build-tools/uClibc-0.9.19
$ make CROSS=powerpc-linux- menuconfig
$ make CROSS=powerpc-linux-
$ make CROSS=powerpc-linux- PREFIX=”” install
```

uClibc configuration:

Linux kernel header location

```
                ${PRJROOT}/kernel/linux-2.4.20
```

Shared library loader path

```
                /lib
```

uClibc development environment directory

```
                ${PRJROOT}/tools/uclibc
```

uClibc development environment system directory

```
                $(DEVEL_PREFIX)
```

uClibc development environment tool directory

```
                $(DEVEL_PREFIX)
```

Kernel configure:

```
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- menuconfig
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- oldconfig
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- xconfig
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- config
```

Kernel build and install:

```
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- clean dep
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- zImage
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- modules
$ cp arch/ppc/boot/images/vmlinux.gz \
> ${PRJROOT}/images/vmlinux-2.4.20.gz
$ cp System.map ${PRJROOT}/images/System.map-2.4.20
$ cp vmlinux ${PRJROOT}/images/vmlinux-2.4.20
$ cp .config ${PRJROOT}/images/2.4.20.config
$ make ARCH=ppc CROSS_COMPILE=powerpc-linux- \
> INSTALL_MOD_PATH=${PRJROOT}/images/modules-2.4.20 \
> modules_install
$ depmod.pl -k ./vmlinux -F ./System.map \
> -b ${PRJROOT}/images/modules-2.4.20/lib/modules > \
> ${PRJROOT}/images/modules-2.4.20/lib/modules/modules.dep
```

Root Filesystem Hierarchy:

/bin	=>	Essential user binaries
/boot	=>	Bootloader and kernel images
/dev	=>	Device files
/etc	=>	System configuration
/home	=>	User home directories
/lib	=>	Essential shared libs + kernel modules
/mnt	=>	Temporary mount point
/opt	=>	Add-on software packages
/sbin	=>	Essential system binaries
/tmp	=>	Temporary files
/usr	=>	Secondary hierarchy (mostly user apps)
/var	=>	Variable data generated by daemons

Creating root directories:

```
$ cd ${PRJROOT}/rootfs
$ mkdir bin dev etc lib proc sbin tmp usr var
$ chmod 1777 tmp
$ mkdir usr/bin usr/lib usr/sbin
$ mkdir var/lib var/lock var/log var/run \
> var/tmp
$ chmod 1777 var/tmp
```

Copying glibc libraries to target rootfs:

```
$ cd ${TARGET_PREFIX}/lib
$ cp *-*.so ${PRJROOT}/rootfs/lib
$ cp -d *.so.[*0-9] ${PRJROOT}/rootfs/lib
$ cp libSegFault.so libmemusage.so \
> libpcprocfile.so ${PRJROOT}/rootfs/lib
$ powerpc-linux-strip \
> ${PRJROOT}/rootfs/lib/*.so
```

Copying uClibc libraries to target rootfs:

```
$ cd ${PREFIX}/uclibc/lib
$ cp *-*.so ${PRJROOT}/rootfs/lib
$ cp -d *.so.[*0-9] ${PRJROOT}/rootfs/lib
```

Copying kernel modules:

```
$ cp -a ${PRJROOT}/images/modules-2.4.20/* \
> ${PRJROOT}/rootfs
```

Creating device files:

```
$ cd ${PRJROOT}/rootfs/dev
$ su -m
Password:
# mknod -m 600 mem c 1 1
# mknod -m 666 null c 1 3
# mknod -m 666 zero c 1 5
# mknod -m 644 random c 1 8
# mknod -m 600 tty0 c 4 0
# mknod -m 600 tty1 c 4 1
# mknod -m 600 ttyS0 c 4 64
# mknod -m 666 tty c 5 0
# mknod -m 600 console c 5 1
# mknod -m 600 mtd0 c 90 0
# mknod -m 600 mtd1 c 90 2
# mknod -m 600 mtdblock0 b 31 0
# mknod -m 600 mtdblock1 b 31 1
# exit
```

Installing BusyBox:

```
$ cd ${PRJROOT}/sysapps/busybox-0.60.5
$ make TARGET_ARCH=ppc CROSS=powerpc-linux- \
> PREFIX=${PRJROOT}/rootfs all install
```

Sample /etc/inittab:

```
::sysinit:/etc/init.d/rcS
::respawn:/sbin/sh
::respawn:/bin/custom-app
::restart:/sbin/init
::shutdown:/bin/mount -a -r
```

Sample /etc/init.d/rcS:

```
#!/bin/sh
mount -n -o remount,rw /
mount /proc
/sbin/ifconfig eth0 192.168.172.205
```

Sample /etc/fstab:

```
/dev/nfs        /               nfs             defaults
none            /proc           proc            defaults
```

Creating filesystem images:

```
$ cd ${PRJROOT}
$ mkcramfs rootfs/ images/cramfs.img
$ genromfs -d rootfs/ -f images/romfs.img
$ mkfs.jffs2 -r rootfs/ -o images/jffs2.img
```

Creating a filesystem image for use as a RAM disk:

```
$ cd ${PRJROOT}
$ mkdir tmp/initrd
$ dd if=/dev/zero of=images/initrd.img bs=1k count=8192
$ su -m
# /sbin/mke2fs -F -v -m0 images/initrd.img
# mount -o loop images/initrd.img tmp/initrd
# cp -av rootfs/* tmp/initrd
# umount tmp/initrd
# exit
$ gzip -9 < images/initrd.img > images/initrd.bin
```

Writing filesystem image to flash (on target):

```
# eraseall /dev/mtd2
# cat /tmp/initrd.bin > /dev/mtd2
```

Kernel boot parameters:

Root device	`root=ROOT_DEV`
Seconds before reboot on panic	`panic=NB_SECONDS`
init program	`init=INIT_PATH`
Default console	`console=CONSOLE_DEV`
Location of NFS server	`nfsroot=IP:DIR`
Mount rootfs as read-only	`ro`
Mount rootfs as read-write	`rw`
Change default RAM disk size	`ramdisk_size=SIZE`

Sample /etc/dhcpd.conf:

```
ddns-update-style ad-hoc;
subnet 192.168.172.0 netmask 255.255.255.0 {
option routers 192.168.172.222;
option subnet-mask 255.255.255.0;
host example-sys {
hardware ethernet 00:CF:78:44:AB:9E;
fixed-address 192.168.172.205;
option host-name “example-sys”;
next-server 192.168.172.222;
filename “/home/karim/example-sys/images/vmlinux-2.4.20.img”;
option root-path “/home/karim/example-sys/rootfs”;
}}
```

Sample /etc/exports (NFS):

```
/home/karim/example-sys/rootfs 192.168.172.205(rw,no_root_squash)
```

Building and installing U-Boot:

```
$ cd ${PRJROOT}/bootldr/u-boot-0.4.0
$ make TQM860L_config
$ make CROSS_COMPILE=powerpc-linux-
$ cp System.map ${PRJROOT}/images/u-boot.System.map-0.4.0
$ cp u-boot.bin ${PRJROOT}/images/u-boot.bin-0.4.0
$ cp u-boot ${PRJROOT}/images/u-boot-0.4.0
$ cp tools/mkimage ${PREFIX}/bin
```

Creating U-Boot images:

```
$ cd ${PRJROOT}/images
$ mkimage -n '2.4.20 example-sys' \
> -A ppc -O linux -T kernel -C gzip -a 00000000 \
> -e 00000000 -d vmlinux-2.4.20.gz vmlinux-2.4.20.img
Image Name:   2.4.20 example-sys
Created:      Mon Jun 30 14:17:05 2003
Image Type:   PowerPC Linux Kernel Image (gzip compressed)
Data Size:    566934 Bytes = 553.65 kB = 0.54 MB
Load Address: 0x00000000
Entry Point:  0x00000000
$ mkimage -n 'RAM disk' \
> -A ppc -O linux -T ramdisk -C gzip \
> -d initrd.bin initrd.boot
Image Name:   RAM disk
Created:      Mon Jun 30 14:25:36 2003
Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
Data Size:    2585662 Bytes = 2525.06 kB = 2.47 MB
Load Address: 0x00000000
Entry Point:  0x00000000
```

U-Boot commands:

```
=> help
=> help cp
=> printenv
=> setenv kernel_addr 00100000
=> saveenv
=> bootp
=> iminfo 00100000
=> bootm 00100000
=> tftpboot 00100000 /home/karim/examples-sys/images/vmlin...
=> erase 40100000 401FFFFF
=> cp.b 00100000 40100000 $(filesize)
=> bootm 00100000 00200000
```

Updating U-Boot:

```
=> tftp 00100000 /home/karim/example-sys/images/u-boot.bin...
=> protect off 40000000 4003FFFF
=> erase 40000000 4003FFFF
=> cp.b 00100000 40000000 $(filesize)
=> setenv filesize
=> saveenv
=> reset
```

Sample /etc/lilo.conf.target:

```
boot = /dev/hdb
disk = /dev/hdb
  bios = 0x80
image = /boot/bzImage-2.4.20
  root = /dev/hda1
  label = Linux
  read-only
```

gdb setup:

```
$ cd ${PRJROOT}/debug
$ tar xvzf gdb-5.2.1.tar.gz
$ mkdir build-gdb
$ cd build-gdb
$ ../gdb-5.2.1/configure --target=$TARGET --prefix=${PREFIX}
$ make
$ make install
```

Building and installing gdbserver:

```
$ mkdir ${PRJROOT}/debug/build-gdbserver
$ cd ${PRJROOT}/debug/build-gdbserver
$ chmod +x ../gdb-5.2.1/gdb/gdbserver/configure
$ CC=powerpc-linux-gcc ../gdb-5.2.1/gdb/gdbserver/configure \
> --host=$TARGET --prefix=${TARGET_PREFIX}
$ make
$ make install
$ cp ${TARGET_PREFIX}/bin/gdbserver ${PRJROOT}/rootfs/usr/bin
```

Starting gdbserver on the target:

```
# gdbserver 192.168.172.222:2345 example-app
# gdbserver /dev/ttyS0 example-app
```

Connecting to gdbserver from the host:

```
$ powerpc-linux-gdb example-app
(gdb) target remote 192.168.172.205:2345
Remote debugging using 192.168.172.205:2345
0x10000074 in _start ()
$ powerpc-linux-gdb example-app
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x10000074 in _start ()
(gdb) set solib-absolute-prefix ../../tools/powerpc-linux/
```

Building and installing strace:

```
$ cd ${PRJROOT}/debug/strace-4.4
$ CC=powerpc-linux-gcc ./configure --host=$TARGET
$ make
$ cp strace ${PRJROOT}/rootfs/usr/sbin
```

Need Help?

We are the pros in building embedded Linux systems straight from source:
Opersys inc.
www.opersys.com
1.866.677.4546
info@opersys.com

Books

Building Embedded Linux Systems,
Karim Yaghmour, O'Reilly 2003
Understanding the Linux Kernel 2nd ed.,
Bovet and Cesati, O'Reilly 2002
Linux Device Drivers 2nd ed.,
Rubini and Corbet, O'Reilly 2001

Online

http://www.embeddedtux.org/
http://www.tldp.org/
http://lwn.net/
http://www.kerneltraffic.org/
http://www.ucdot.org/
http://www.uclinux.org/
http://www.linuxdevices.com/

OPERSYS INC.

Essential Expertise For Building Embedded Linux Systems Straight From Source

Building Embedded Linux Systems Straight From Source

Quick Reference Guide
v 1.0.5b

www.opersys.com
1.866.677.4546