Foundation of PIOS

Yu Zhang (Clara)
clarazhang@gmail.com

June 6, 2011

QEMU

1 Building PIOS (Determinator)

Note that PIOS (the instructional kernel used in CS422) and Determinator (the research kernel) currently share the same git source repository but exist in separate, parallel branches of that repository. The master (default) branch is the instructional PIOS, whereas the dcm branch holds the Determinator research kernel.

To get the Determinator source code, make sure your SSH public key has been installed in our gitosis repository as described in SSH keys, then type:

```
$ git clone git@dedis.cs.yale.edu:pios
$ git checkout -b dcm origin/dcm
```

Now look over the README file in the root directory of the source tree you git, except be aware that this README was written for PIOS (the instructional kernel) rather than Determinator. In particular, to build the full Determinator system correctly, you will need to configure it for LAB=9 SOL=9 (instead of the suggested lab 4 or 5 as described in the README for PIOS), as follows:

`make LAB=9 SOL=9 labsetup`

You may not yet be able to build Determinator successfully, however. Building Determinator requires using a pios-gcc toolchain that was created with modifications specifically for PIOS/Determinator, as described below.

1.1 Determinator build tools: prerequisite packages

Before building binutils and gcc, make sure you have the following prerequisite packages installed:

- gmp 4.1+
- mpfr 2.3.2+
- unifdef: `sudo apt-get install unifdef`

- makeinfo: `sudo apt-get install texinfo`

If building on 64-bit Linux, make sure you have the 32-bit Linux headers installed. On Debian-based distributions, install libc6-dev-i386.

1.2 Building binutils and gcc for Determinator

The PIOS specific binutils and gcc may be obtained and built as follows.

1. Check out copies of the pios branch of the local gcc and binutils repositories:

   ```
   $ git clone -b pios git@korz.cs.yale.edu:binutils
   $ git clone -b pios git@korz.cs.yale.edu:gcc
   ```

2. Configure the binutils and gcc source trees for target pios and install them in the directory .../pios/xc where .../pios/ is the root of your PIOS/Determinator source tree. xc means cross-compiler, and the Determinator makefiles expect the cross compiler to install some includes in xc/pios/include and some libraries in xc/pios/lib.

   ```
   $ mkdir binutils/obj
   $ cd binutils/obj
   $ ../configure --target=pios --prefix=.../pios/xc --disable-werror
   $ make && make install
   $ mkdir ../../gcc/obj
   $ cd ../../gcc/obj
   $ ../configure --target=pios --prefix=.../pios/xc --disable-werror
    --enable-languages=c
   $ make && make install
   ```

   If you run into errors while compiling gcc about not being able to find stddef.h or some other standard header file, it may be because you have not built PIOS yet. The gcc compilation process looks in your PIOS source tree for those header files. Go into your PIOS/Determinator source tree and make:

   ```
   $ make LAB=9 SOL=9 labsetup && make
   ```

   Then, `make clean && make && make install` again in your gcc source tree.

3. Either put the .../pios/xc/bin/ directory in your PATH or add symlinks from /usr/local/bin to all the binaries in that directory.

4. Try to `make clean; make install` again in the PIOS/Determinator directory.

1.3 Scripts in PIOS

1.3.1 Special dev files

- `/dev/zero`: a special file in Unix-like operating systems that provides as many null characters (ASCII NUL, 0x00) as are read from it. One of the typical uses is to provide a character stream for initializing data storage.
- `/dev/null`: a special file that discards all data written to it (but reports that the write operation succeeded) and provides no data to any process that reads from it (yielding EOF immediately).

1.3.2 Tips of Makefile

Commands If commands in Makefile are led by `@`, they will be executed without printing verbose commands.

Functions Some functions used in PIOS Makefiles are introduced below.

- `$(wildcard pattern...)`: Wildcard expansion happens automatically in rules. But wildcard expansion does not normally take place when a variable is set, or inside the arguments of a function. If you want to do wildcard expansion in such places, you need to use the *wildcard* function.
- `$(subst from, to, text)`: Performs a textual replacement on the *text* text: each occurrence of *from* is replaced by *to*.
- `$(patsubst pattern, replacement,text)`: Finds whitespace-separated words in *text* that match *pattern* and replaces them with *replacement*. '%' in *pattern* or *replacement* acts as a wildcard.
  e.g. `$(patsubst %.c,%.o,$(wildcard *.c))`
  It first get a list of all the C source files in a directory, and then change the list into a list of object files.
- `$(addprefix prefix, names...)`: The value of *prefix* is prepended to the front of each individual name of *names* separated by whitespace and the resulting larger names are concatenated with single spaces between them.

Automatic Variables See GNU.

- `$@` The file name of the target of the rule.
- `$(@D)` The directory part of the file name of the target, with the trailing slash removed.
- `$<` The name of the first prerequisite.
- `$^` The names of all the prerequisites, with spaces between them.

1.3.3 GNUmakefile

- Environment configuration: see `conf/env.mk, conf/lab.mk`
- Cross-compiler toolchain: infer the prefix of cross-compiler toolchain (GCCPREFIX) by running `misc/gccprefix.sh`. The cross-compiler toolchain is installed as 'pios-*' or 'i386-elf-*'. If the host tools (gcc, objdump, and so on) compile for a 32-bit x86 ELF target, that will be detected as well.
- QEMU: infer the correct QEMU by running `misc/which-qemu.sh` if QEMU is not defined.
- GDB and network port numbers: generated by performing
  `` expr 'id -u` % 5000 + n ``,
  where n is 25000 and 30000 respectively.
- QEMUPORT: an option to enable the GDB stub and specify its port number to qemu.

  `-s -p $(GDBPORT)` for qemu versions <= 0.10, and
  `-gdb tcp::$(GDBPORT)` for later qemu versions.
- Compilation Options (see gcc, ld):
  `CFLAGS += -nostdinc -m32` ...
  `LDFLAGS += -nostdlib -m elf_i386` ...
  Compiler flags that differ for kernel versus user-level code.
  ```
  KERN_CFLAGS += $(CFLAGS) -DPIOS_KERNEL  ...
  KERN_LDFLAGS += $(LDFLAGS) -nostdlib -Ttext=0x00100000 -L$(GCCDIR)
  KERN_LDLIBS += $(LDLIBS) -lgcc

  USER_CFLAGS += $(CFLAGS) -DPIOS_USER
  USER_LDFLAGS += $(LDFLAGS)
  USER_LDINIT += $(OBJDIR)/lib/crt0.o
  USER_LDDEPS += $(USER_LDINIT) $(OBJDIR)/lib/libc.a
  USER_LDLIBS += $(LDLIBS) -lc -lgcc
  ```
- Makefrags for subdirectories, including boot, kern, lib, user.
- Set NCPUS, number of CPUs.
- Set QEMUOPTS `-smp $(NCPUS) -hda $(OBJDIR)/kern/kernel.img -serial mon:stdio -k en-us -m 1100M`
- Targets in the Makefile
  - `labsetup` set values of macros LAB and SOL, and write them to conf/lab.mk.

  - `export-lab%`, `export-sol%` or `export-prep%`(where % is a wildcard character, e.g. 1 5,9): export the student lab handout and solution trees.

    `misc/mklab.pl` is used to construct various trees.
  - `build-lab%`, `build-sol%` or `build-prep%` build the corresponding student lab handout or solution tree.
  - `qemu`, `qemu-nox`, `qemu-gdb`, `qemu-gdb-nox`: the first two launch QEMU and run PIOS with/without a virtual VGA display, the last two launch QEMU for debugging with/without a virtual VGA display.

1.3.4 boot/Makefrag

- Compile `boot.S, main.c` into `boot.o, main.o`
- Link a number of object files into an elf file using (ld):
  ```
  ld -nostdlib -m elf_i386 -L$(OBJDIR)/lib -L$(GCCDIR) -N
  -e start -Ttext 0x7C00 -o $(OBJDIR)/bootblock.elf
   boot.o main.o
  ```
  `-nostdlib`: Only search library directories explicitly specified on the command line.
  `-m elf_i386`: Emulate the emulation linker. You can list the available emulations with the –verbose or -V options.
  `-e start`: Use `start` as the explicit symbol for beginning execution of your program, rather than the default entry point.
  `-Ttext 0x7C00`: Locate `.text` section in the output file at the absolute address 0x7C00.
- Output source code intermixed with disassembly from boot.elf to boot.asm using objdump.
  ```
  objdump -S bootblock.elf >bootblock.asm
  ```
- Copy the contents of an object file to another using objcopy.
  ```
  objcopy -S -O binary bootblock.elf bootblock
  ```
  `-S`: Do not copy relocation and symbol information from the source file.
  `-O binary`: Generate a raw binary file by using an output target of binary.
- `perl boot/sign.pl bootblock`: Pad the bootblock to the require 512 byte sector size and attaches the boot magic "signature" (0x55aa - means bootable) at the end.

1.3.5 kern/Makefrag

- Related macros:
  - `KERN_SRCFILES`: the list of kernel source files
  - `KERN_INITFILES`: files to comprise the kernel's initial file system
  - `KERN_FSFILES`: files found in the 'fs' subdirectory

  - `KERN_BINFILES`: binary program images to embed within the kernel (for each file *f* in `KERN_INITFILES`, its binary program image is `user/`*f*)
  - `KERN_OBJFILES`: kernel object files generated from C (.c) and assembly (.S) source files

- Create `$(OBJDIR)/kern/initfiles.h` from `KERN_INITFILES` while `LAB>=4`, in which the `'/'`,`'.'`, `'-'` are replaced by `'_'`. This target is the pretask of `$(OBJDIR)/kern/file.o`.
- Compile `kern/*.c` to `$(OBJDIR)/kern/*.o`
- Compile `dev/*.c` to `$(OBJDIR)/dev/*.o`
- Compile `lib/*.c` to `$(OBJDIR)/kern/*.o`, mainly on standard I/O.
- Link the kernel itself from its object and binary files.

```
ld -o $@ $(KERN_LDFLAGS) $(KERN_OBJFILES) $(KERN_LDLIBS)
   verb| -b binary $(KERN_BINFILES)
objdump -S $@ > $@.asm
nm -n $@ > $@.sym
```

  where `$@` is `$(OBJDIR)/kern/kernel`, and its source code intermixed with disassembly and symbols are generated by objdump and nm, respectively.`-n` in `nm` represents sorting symbols numerically by their addresses, rather than alphabetically by their names.

- Build the kernel disk image using dd (convert and copy a file)

```
dd if=/dev/zero of=$(OBJDIR)/kern/kernel.img~ count=10000\
 2 >/dev/null
dd if=$(OBJDIR)/boot/bootblock of=$(OBJDIR)/kern/kernel.img~\
  conv=notrunc 2>/dev/null
dd if=$(OBJDIR)/kern/kernel of=$(OBJDIR)/kern/kernel.img~\
  seek=1 conv=notrunc 2>/dev/null
mv $(OBJDIR)/kern/kernel.img~ $(OBJDIR)/kern/kernel.img
```

  - `if=FILE`: read from FILE instead of stdin
  - `of=FILE`: write to FILE instead of stdout
  - `count=BLOCKS`: copy only BLOCKS input blocks
  - `conv=CONVS`: convert the file as per the comma separated symbol list. `CONVS` may be `notrunc`(do not truncate the output file),....
  - `skip=BLOCKS`: skip BLOCKS ibs-sized blocks at start of input

- Copy `fs/%` to `$(OBJDIR)/user/%`.

1.3.6 lib/Makefrag

- Related macros:
  - `LIB_SRCFILES`: source files comprising the minimal PIOS C library
  - `LIB_OBJFILES`: object files generated from C (.c) and assembly (.S) source files
- Compile `lib/*.c` to `$(OBJDIR)/lib/*.o`
- Compile `lib/*.S` to `$(OBJDIR)/lib/*.o`
- Create archives `$(OBJDIR)/lib/libc.a` from `$(OBJDIR)/lib/*.o`

  `ar r $@ $(LIB_OBJFILES)`, where `$@` is `$(OBJDIR)/lib/libc.a`, `r` means inserting files with replacement.

`Rules building a target install tree for a GCC (cross-)compiler.`

- Related macros:
  - `XCDIR := xc/pios`
  - `XC_INCFILES`: the list of file names in `inc/*.h`
  - `XC_DEPS`: the list of file names by prefixing `$(XCDIR)/include/` to each file name in `XC_INCFILES`, `xc/pios/include/sys` and `libc.a`, `libm.a`, `libg.a`, `libpthread.a` and `crt0.o` in —`xc/pios/lib` directory.
- Make symlinks `$(XCDIR)/include/sys` to `$(XCDIR)/include/`.
- Copy `$(OBJDIR)/lib/libc.a` to `$(XCDIR)/lib/lib%.a`.
- Copy `$(OBJDIR)/lib/crt%.o` to `$(XCDIR)/lib/crt%.o`.

1.3.7 user/Makefrag

- Related macros
  - `USER_CFLAGS`: defined in GNUmakefile.
    `$(CFLAGS) -DPIOS_USER -DPIOS_SPMC`
  - `USER_LDINIT += $(OBJDIR)/lib/crt0.o`, defined in GNUmakefile.
  - `USER_LDDEPS += $(USER_LDINIT) $(OBJDIR)/lib/libc.a`, defined in GNUmakefile.
  - `USER_LDLIBS += $(LDLIBS) -lc -lgcc`, defined in GNUmakefile.
- Compile `user/*.c` to `$(OBJDIR)/user/*.o`
- Link and create executable files.