Overview of Environmental Sample Processor (ESP) Support Software

5/7/22 Brent Roman brent@mbari.org

Major ESP Software Components

- U-Boot Open Source system boot loader
  - Starts GNU/Linux from power up or hardware reset
- GNU/Linux Operating System
  - Open source operating system
  - Includes complete suite of UNIX text processing utilities
- The Unix Shell and its Scripting Language
  - Starts/Stops and manages processes
  - Central to almost all Linux (and Unix) computers
- Ruby Scripting Language
  - Object oriented, general purpose programming
  - Slow, but safe and very flexible

Das U-Boot Universal boot loader

- Usually runs for only the first few seconds
  - to load and start the Linux Kernel
- May load Linux from various storage media
  - for redundancy in case some media fails
  - to allow different kernel versions to be started
- Maintains an “environment” of startup options
  - Environment is a set of *key=value* pairs
  - Stored in non-volatile memory on the ESP CPU board
- Includes a very basic scripting language
  - Only accessible from the ESP’s console serial port

Linux Kernel

- Controls access to all hardware
- Implements file storage and networking protocols
- Somewhat extensible via Device Driver modules
  - Driver modules are valid only for one particular kernel version
  - Updating the core kernel is difficult
    - may be necessary to support new types of networking or hardware
- End users should not interact directly with the kernel
- Boot loader determines kernel parameters at startup
- Outputs a log that can be useful for debugging

GNU/Linux Operating System

- Linux is everywhere
  - All Android Phones [*Linux kernel only*]
  - >95% all internet servers
- GNU/Linux is derived from Unix
  - Utilities and languages are (nearly) identical {*Despite GNU acronym*}
  - Every Apple Computer runs Unix
    - The entire ESP software suite runs on Apple Computers
- Windows is not derived from Unix
  - However, recent Windows versions include a
    - Windows Subsystem for Linux
    - Should be easy to port ESP suite to this, *someday*

GNU/Unix commands

- Evolved over the past 50 years
  - Some commands' names are obvious
    - head, tail, find, kill, ping, echo, sort, history, shutdown, alias, unalias, touch
  - But, common ones are obtuse and needlessly terse
    - wc, mkdir, rm, rmdir, man, ls, grep, gzip, gunzip
  - Google 'Linux commands'
    - If you want detailed info on specific *command*
      - try googling: man *command*
      - or type: $ *command* --help

Common Unix command syntax

- Almost all Unix commands parse like this:
  - *command* {--*option*{=*optArg*}? }* {-*o* {*optArg*}? }* {*argument* }*
- Each command runs in a ‘process’
- Examples:
  - ls #list all the files in the current directory
  - ls -l   #list files in long format
  - ls --color=never -l   #in long format without colors
  - ls -l /var/log   *#files in the /var/log directory, long form*
  - tail /var/log/messages   #output last lines of system log
  - tail -n30 /var/log/messages #last 30 lines of system log

Unix Environment Variables

- The “environment” is a list of *key=value* pairs
- Processes inherit parts of their parent’s environment
  - in addition to any explicit command arguments
  - Only those environment variables marked for “export”
- Shell built-in commands alter its environment
  - `FOO=bar` #sets variable FOO to the string “bar”
    - Note that there may be no whitespace!
  - `FOO=”Hello there!”` #whitespace must be quoted
  - `export FOO` #causes FOO to be exported into new processes
  - `unset FOO` #deletes the variable FOO
  - `env` #displays all ***exported*** environment variables
  - `echo $FOO` *#displays Hello there!*
- *No process may alter the environment of another!*

Some Environment Variables typically set by Unix shell

- `SHELL=/bin/sh` #the path to the SHELL binary
- `USER=esp` #user account name
- `PWD=/home/esp` #current directory
- `HOME=/home/esp` #user's home directory
- `TZ=US/Pacific` #override default time zone
- `TERM=xterm` #terminal type
- `PATH=/bin:/usr/bin` #binary search path
  - Path above causes shell to search first in /bin, failing that, it tries /usr/bin

ESP Environment Variables

- `ESPmode=real` #operating mode
- `ESPhome=/home/esp/esp2` #top dir of ESP app
- `ESPpath=/home/esp/esp2/mission:.`

  #where to search for ESP mission scripts

- `ESPconfigPath=` #path to config files
- `ESPlog=/var/log/esp` #where to write files
- `ESPname=bruce` #name of ESP machine
- `RUBYLIB=/home/esp/esp2/lib:/home/esp/esp2/utils`
- `PATH=...:/opt/mbari/bin:$ESPhome/bin`

Setting up ESP Environment

- ESPenv script initializes ESP environment
  - *Must be run in the current shell to have any effect!*

```
$ . ESPenv shallow eddie  #for ESPeddie
```

    - 1st argument is the ESP type (for 2G, usually “shallow”)
    - 2nd argument is the machine name
      - *if omitted, the ESPname is derived from machine’s hostname*
    - additional arguments documented in script

Environment Variables (cont'd)

Viewing:

- `$ export` #shows all variables
- `$ env` #shows all exported variables
- `$ env | grep ESP` #shows only ESP variables
- `$ echo $ESPname` #shows the ESP's name

Changing env var for only one command:

- `$ ESPmode=real showlog`
- ESPmode reverts to its original value after above showlog command completes
- showlog could be replaced with any cmd

The Unix Shell

- Starts and stops commands
- May run multiple commands in parallel
- May pipe a command's output into the input of another:
  - cat /etc/passwd | sort | head #output first few sorted accounts
- Is a complete scripting language in itself
  - Many ESP utilities are implemented in Unix shell script
    - showlog is this script:

```
#!/bin/sh
#  tail a log -- the user's ESP log if no file name specified
#  defaults to -f if no tail options given
: ${logDir:=${ESPlog-`echo ~ftp`/$USER}}
unset opts
mode=$ESPmode
for arg do
  case $arg in
    -*) opts="$opts $arg"
    ;;
    *)  mode=$arg
    ;;
  esac
done
eval exec tail ${opts--f} "$logDir/$mode.out"
```

Unix Shell Interactive Scripting

- `showlog --help`
- `showlog -3`
- `while sleep 2; do date; done`
- `export ESPmode=time`
- `while sleep 2; do date; done >>$ESPlog/$ESPmode.out`
- `showlog -10 -v`
- `while sleep 2; do date; done >>$ESPlog/$ESPmode.out &`
- `jobs`
  - `[1] + Running  while sleep 2; do date; done`
- `showlog` #clock now running in the background
- `kill -STOP %1` #pause running job 1
- `kill -CONT %1` #resume job 1
- `kill %1` #terminate running job 1

Starting a Detached Process

- Try:
  - create a new bin/countTime command

```
#!/bin/sh
while sleep 2; do date;done
```

  - $ chmod +x bin/countTime

  - export ESPmode=time
  - countTime
  - start countTime
  - jobs #not known to this shell
  - showlog #clock now running detached from shell
  - kill %1 #does not work – what does?

Stopping a Detached Process

- After having run: `start countTime`
  - from previous slide
  - `$ pstree -Gp $USER`

    `countTime(6353)———sleep(6502)`

    `sh(5458)`

    `sh(5514)———pstree(6503)`
  - The numbers above are PIDs or “process identifiers”
  - `kill 6353` *#or* `killall countTime`
  - wait 3 seconds
  - `kill 6353` *#or* `killall countTime`
  - If 2nd kill attempt does not complain that process does not exist,
    - `$ kill -KILL 6353` *#or* `killall -KILL countTime`
  - `-KILL` is a last resort
    - as the process gets no opportunity to clean up after itself

Carefully Altering Text Files

- Save the original version before making changes
- cd to directory containing file *cfg* to be changed
- `mkdir original`
- `ls original` #verify that original/*cfg* does already exist
- `cp cfg original` #only if original/*cfg* does not exist yet
- `chmod +w cfg` #make *cfg* file writable if not already so
  - files under `esp2` subdirs are read-only
    - to avoid accidental modification
- Later, you can refer to *original/cfg*
  - or move it back over *cfg* to revert all changes
    - `mv original/cfg .`
- Provides you a record of all the files you changed

Comparing Files

- Unix standard ‘diff’ utility will compare to text files
  - or whole directories (if you read this man page)

    `https://linux.die.net/man/1/diff`
- If you saved the original,
  - to compare cfg with it:

    `$ diff original/cfg .`
  - {see ‘Carefully Altering Text Files’ slide}

Editing Files on the ESP

- Only text editor available is ‘vi’
  - just google “vi cheat sheet”
- Save an unmodified version of before you first edit file
- Do not edit Ruby files (under esp2 directory) as root user
- Modifying system configuration files...
  - may make the system unbootable
  - may take down networking
- If system will not start:
  - you may be able to recover by booting
    - from copy of the OS provided on your SDcard
    - issuing commands from the ESP’s serial console port

Getting to the Bootloader Prompt

- Connect to ESP serial console from another computer
  - via a USB<->serial adapter cable
  - 115200 baud, 8 data bits, 1 stop bit, no parity

    `picocom -b115200 /dev/ttyUSB0` *#may be USB1*
- Cycle power to the ESP
  - or press the tiny reset button on the side of the processor board near its console serial connector
- Press <Control-C> in the terminal emulator window
  - the instant console output appears
  - press Control-C once every 500ms until you see:

    `EA3141-MBARI:`
    - The ESP U-Boot bootloader prompt

Booting from the SDcard

- SDcards installed in MBARI ESPs normally
  - have Gigabytes of unused data storage space
  - include a complete copy of the ESP's Linux OS
- To boot from the SDcard
  - at the ESP bootloader prompt, type:

```
EA3141-MBARI:  run sdBoot
```

- Beware that the system will likely be missing many or all
  - of the changes you made to it
  - may revert to its original networking configuration
- Avoid making unneeded changes to this backup OS
  - There is no 3rd copy of the Operating System!