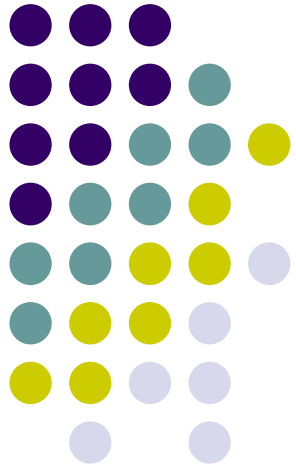
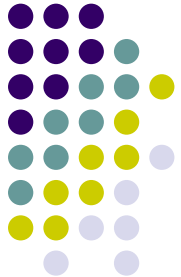
Turn-Key

Image building
Integrating connectivity, QoS, and security

Ying-Dar Lin



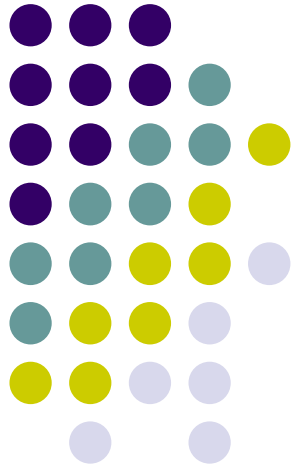
Agenda



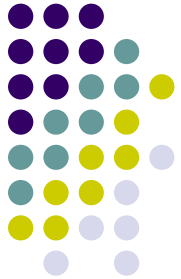
- Backgrounds
  - Turn-key, uCLinux, uClibc
  - ToolChain, **buildroot** system
  - CVS, SVN
- Introduction of the *Wall* Project
- Issues
  - Procedures for adding a package into the turn-key
  - Accelerate the turn-key building procedure
- Mini-project
- Project

Backgrounds

- Turn-key?
- uCLinux, **uClibc**, ToolChain?
- How to use the uClibc **Buildroot** system
- How to use the **CVS** Server/Client

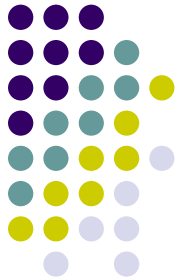


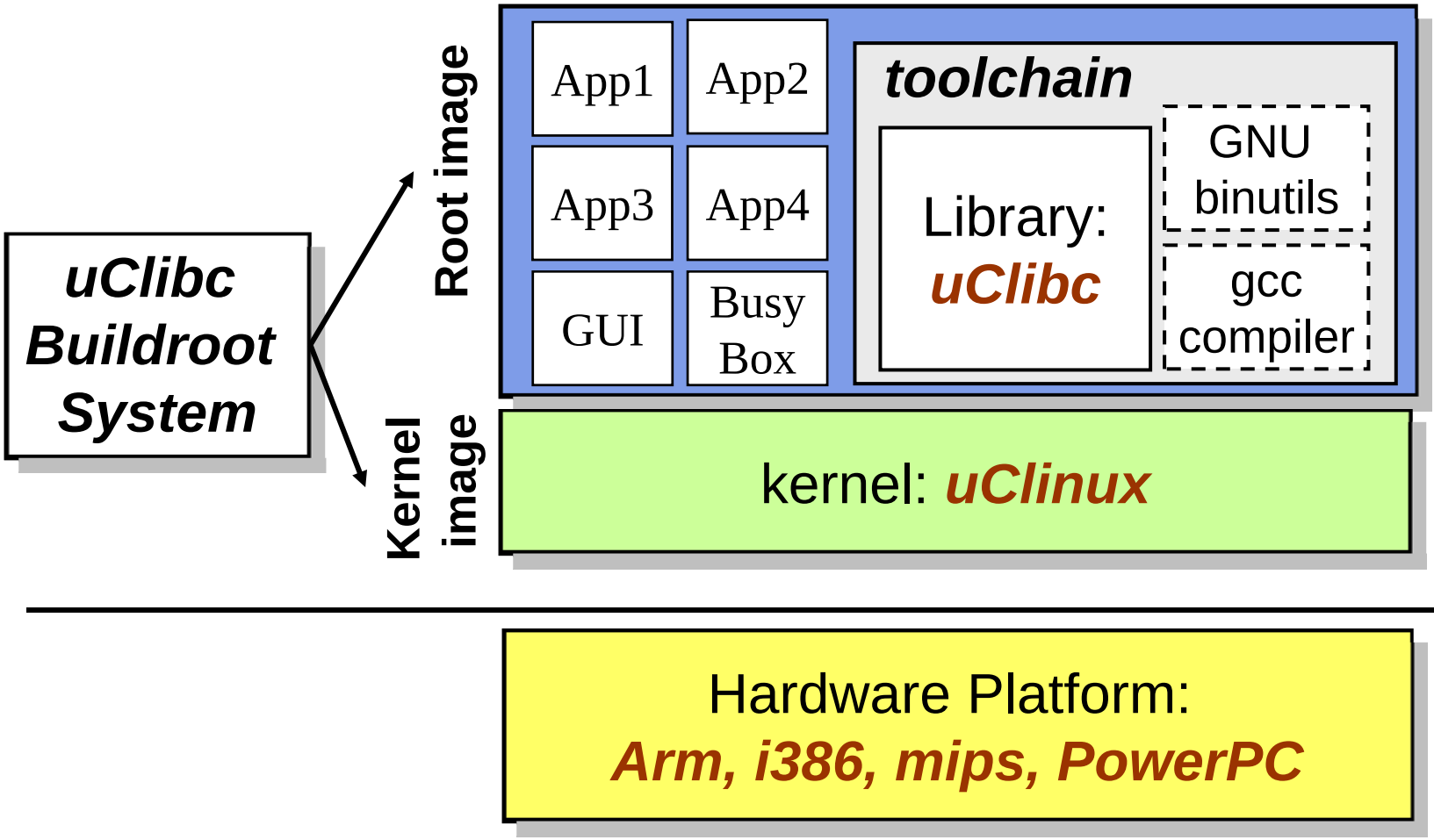
What is a Turn-key?



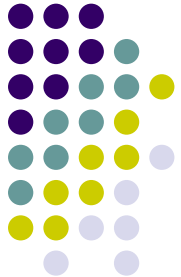
- Pre-built computer "packages"
- Perform a certain type of task
- Everything needed is put together in

A turn-key example:
A Linux-based System





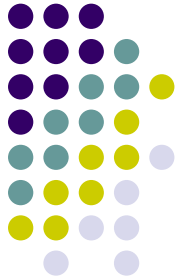
uClibc



- A C library for embedded Linux
- Much smaller than the GNU C Library (glibc)
- Compatible with glibc
- For MMU-less systems (μClinux)
- Also for standard Linux
- Supports shared libraries and threading
- Closed source commercial applications

uClinux

Embedded Linux/Microcontroller Project

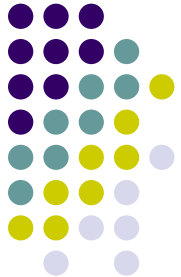


- A derivative of Linux kernel
- for microcontrollers without Memory Management Units (MMUs).
- Linux kernel releases for 2.0 2.4 and 2.6
- http://www.uclinux.org/

uCsimm

- a microcontroller module built specifically for the uClinux Operating System
- an inch high, with a standard 30-pin SIMM form factor
- a Motorola DragonBall 68EZ328 processor
- 2 MB of FLASH and 8Mb of DRAM
- a 10Base-T ethernet and RS 232 high-speed serial
- http://www.uclinux.org/ucsimm/

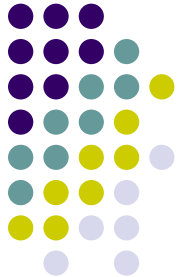
BusyBox



- Combines tiny versions of common UNIX utilities into a single small executable

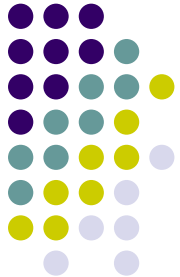
- Fewer options than their full-featured GNU cousins
- Size-optimization and limited resources
- Extremely modular and easy to customize
- http://busybox.net/about.html

Screenshot of BusyBox



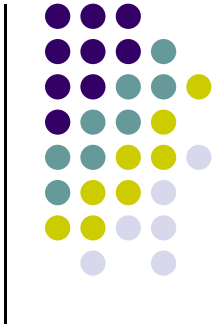
```
BusyBox v1.1.0 (2006.03.08-23:51+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# help

Built-in commands:
------------------
 . : alias bg break cd chdir continue eval exec exit export false
 fg hash help jobs kill let local pwd read readonly return set
 shift times trap true type ulimit umask unalias unset wait

#
```

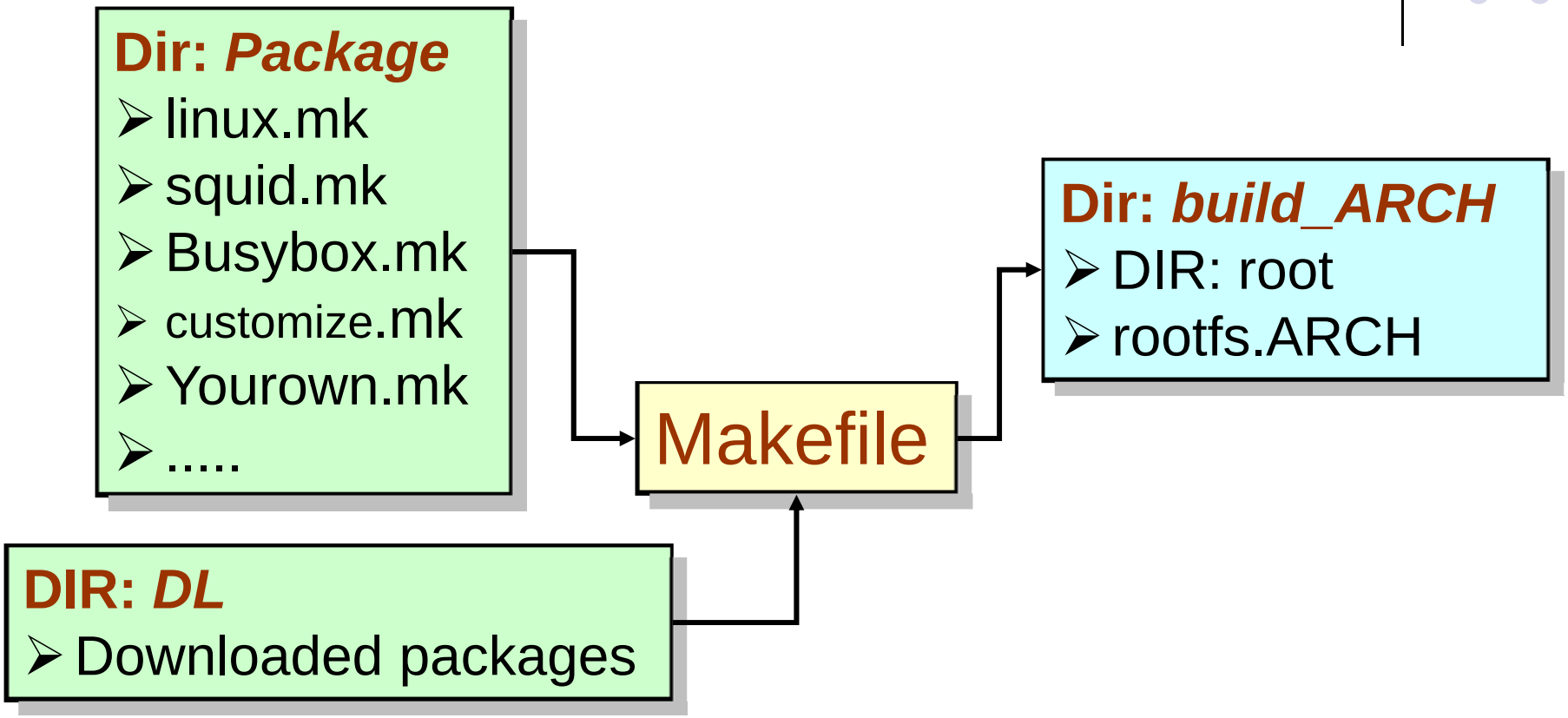
uClibc BuildRoot System



- Select the packages you want
- Build your own
  - uClibc-based root file system
  - Or Development system
- Cross-Platform: arm, i386, m68k, mips, ...
- A time-consuming procedure
- http://buildroot.uclibc.org/buildroot.html
- Pre-compiled devel. systems are available

Architecture in BuildRoot Sys

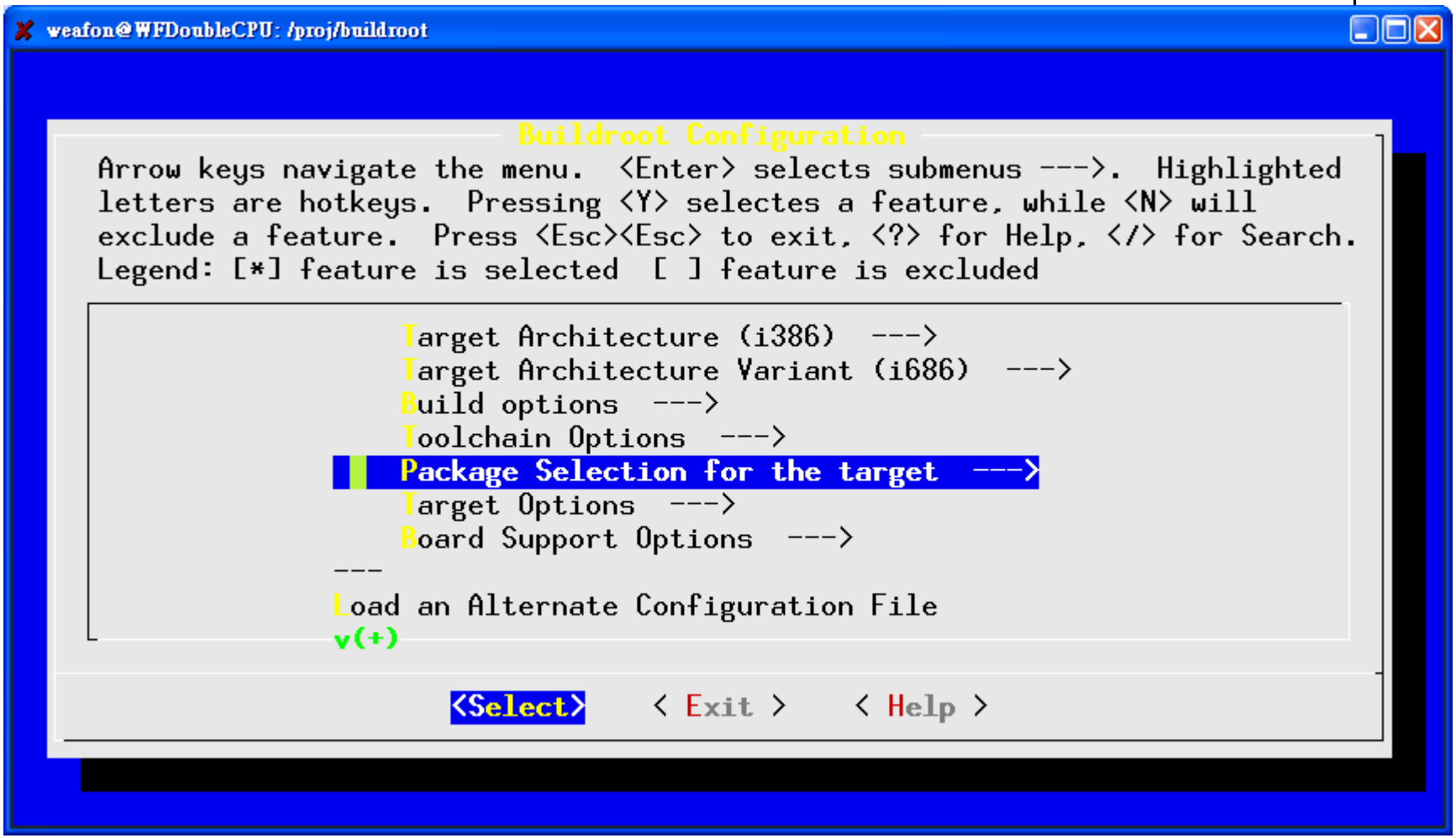




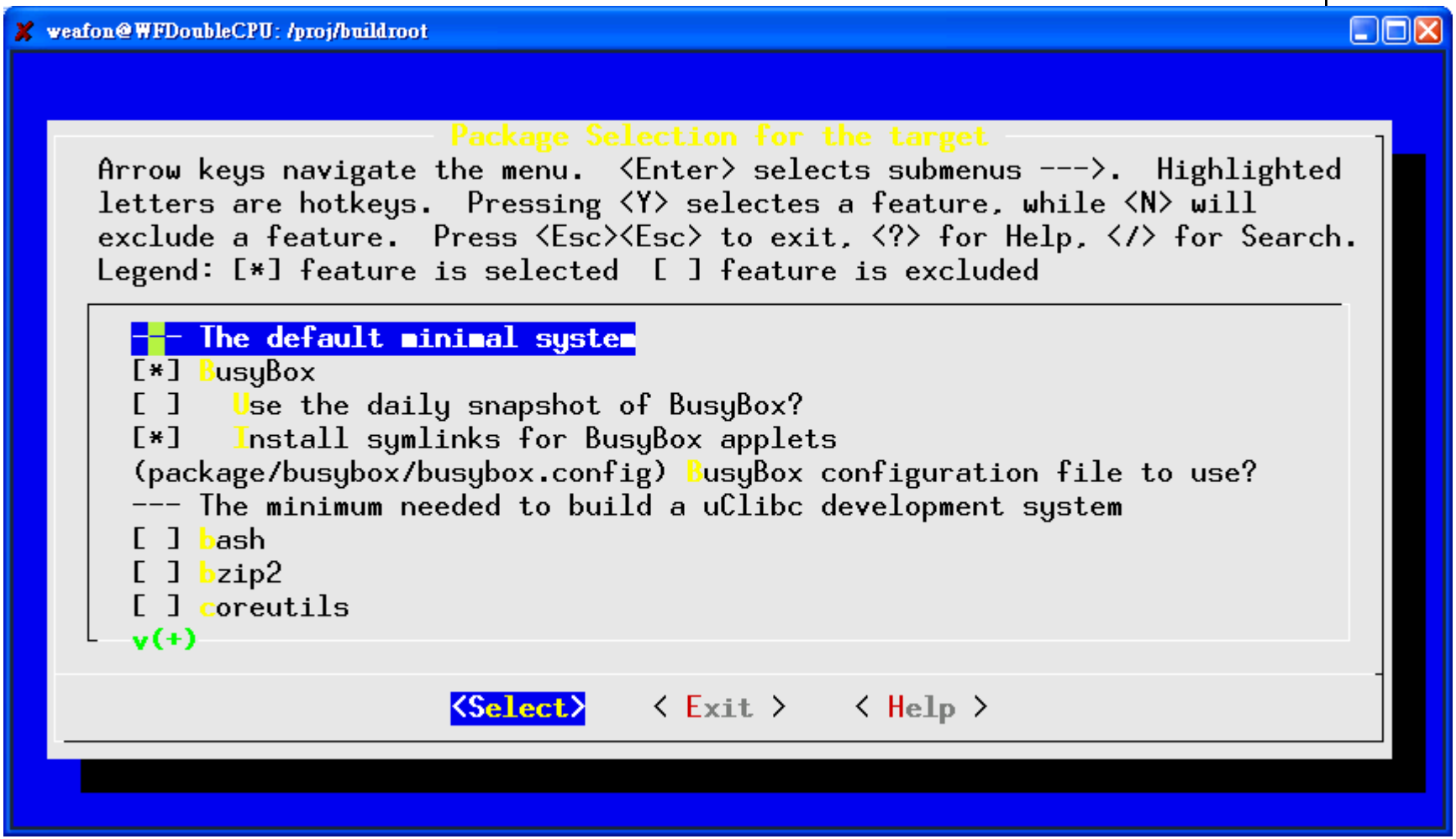
Makefile

```
.........
# OS
TARGETS :=linux
# The default minimal set
TARGETS +=busybox tinylogin
# a full development system
TARGETS +=ed file gawk tar grep bzip2

# your own packages configuration start
TARGETS +=libgmp freeswan
TARGETS +=iproute2 iptables
TARGETS +=dnsmasq
TARGETS +=tripwire
TARGETS +=zebra
.........
TARGETS +=customize #copy configuration
TARGETS +=ext2root  #build image
```

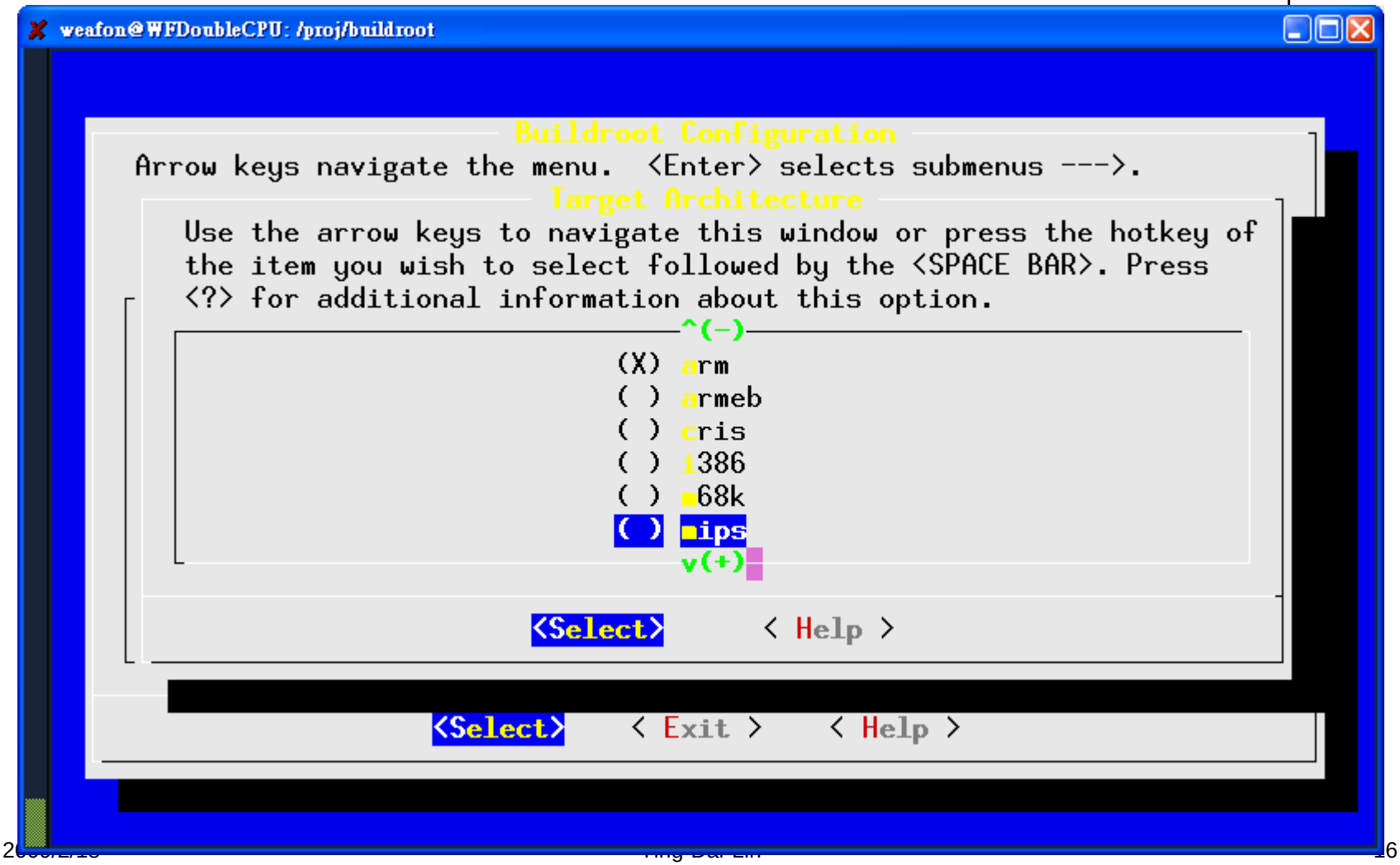
make menuconfig: main menu



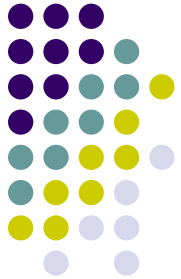
make menuconfig: Package Selection



make menuconfig: Platform Selection

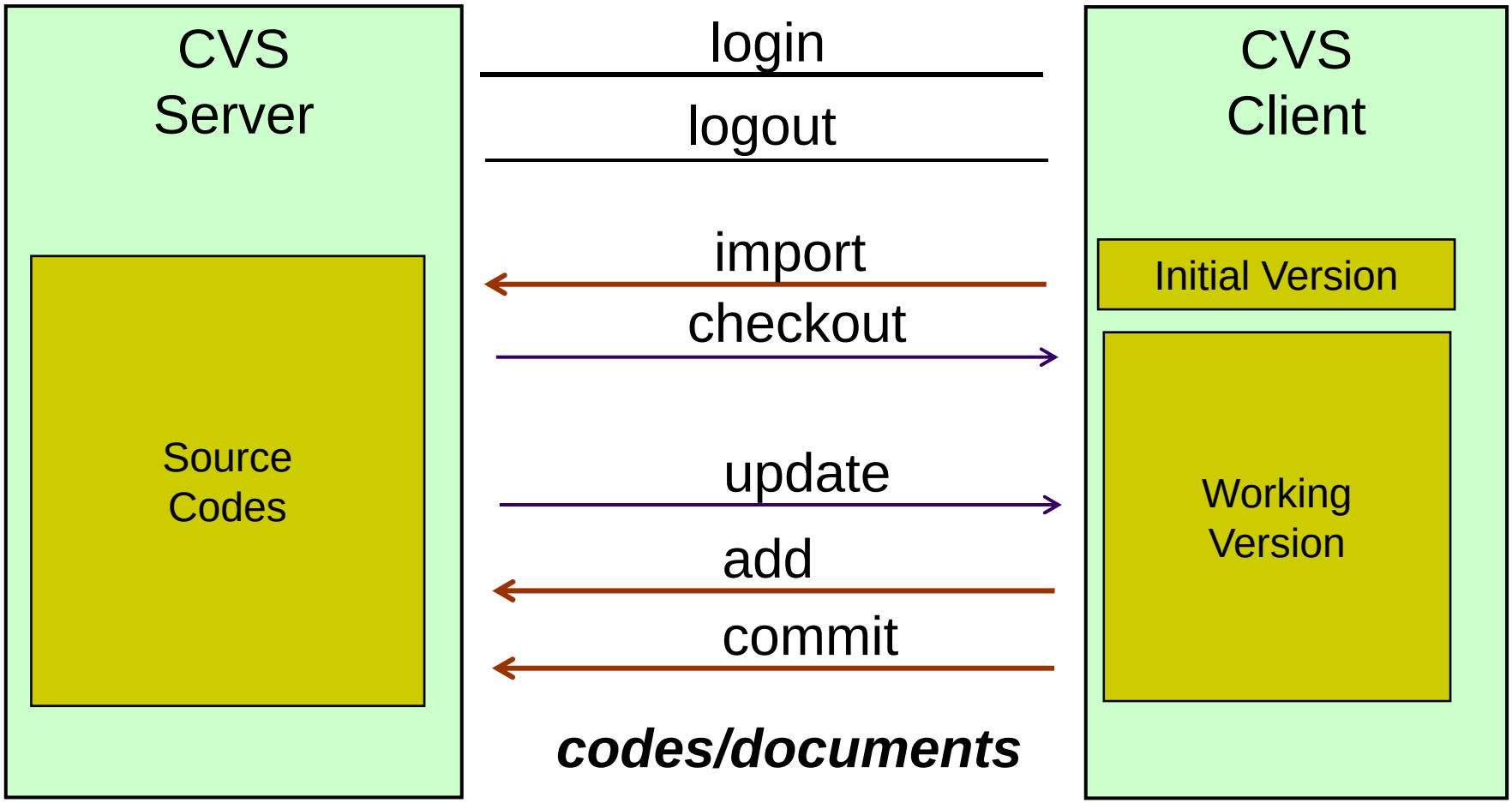


Basic Functions in Source Versions Control System



- **Checkout:**
  - Distribute source codes
- **Version:**
  - File history, comparison, rollback
- **Update/Commit:**
  - Team Work
- **Branch:**
  - An experimental revision
  - Different platforms/purposes

Concurrent Versions System (CVS)

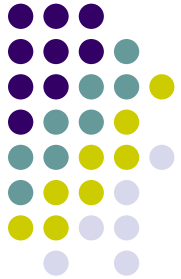


Cases for using CVS (I): Download Open Source Packages

- Only need a CVS client

- Operating procedures:
  - Set *CVSROOT*: to indicate where the server locates
    `CVSROOT = :pserver:username@140.113.xx.xx:/CVSROOT`
  - Login
  - Checkout: download all codes
  - Update: update the latest codes

Cases for using CVS (II):
Maintain Your Own Packages

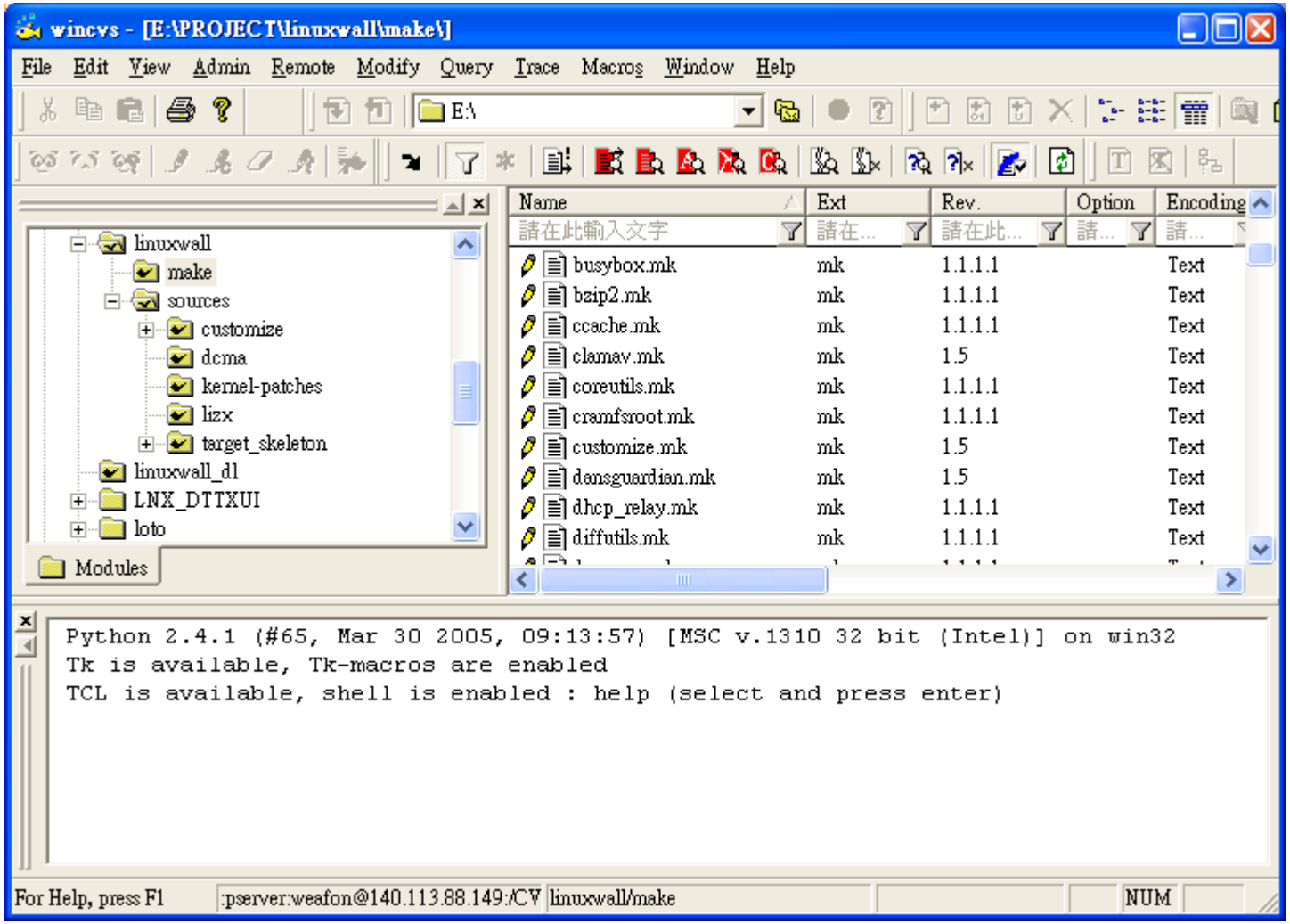


- Need the CVS server and client
- Operating procedures:
  - Create Repository, where you all sources are stored
  - Set *CVSROOT*
  - Login
  - Import
  - Checkout
  - Commit, Update, Add

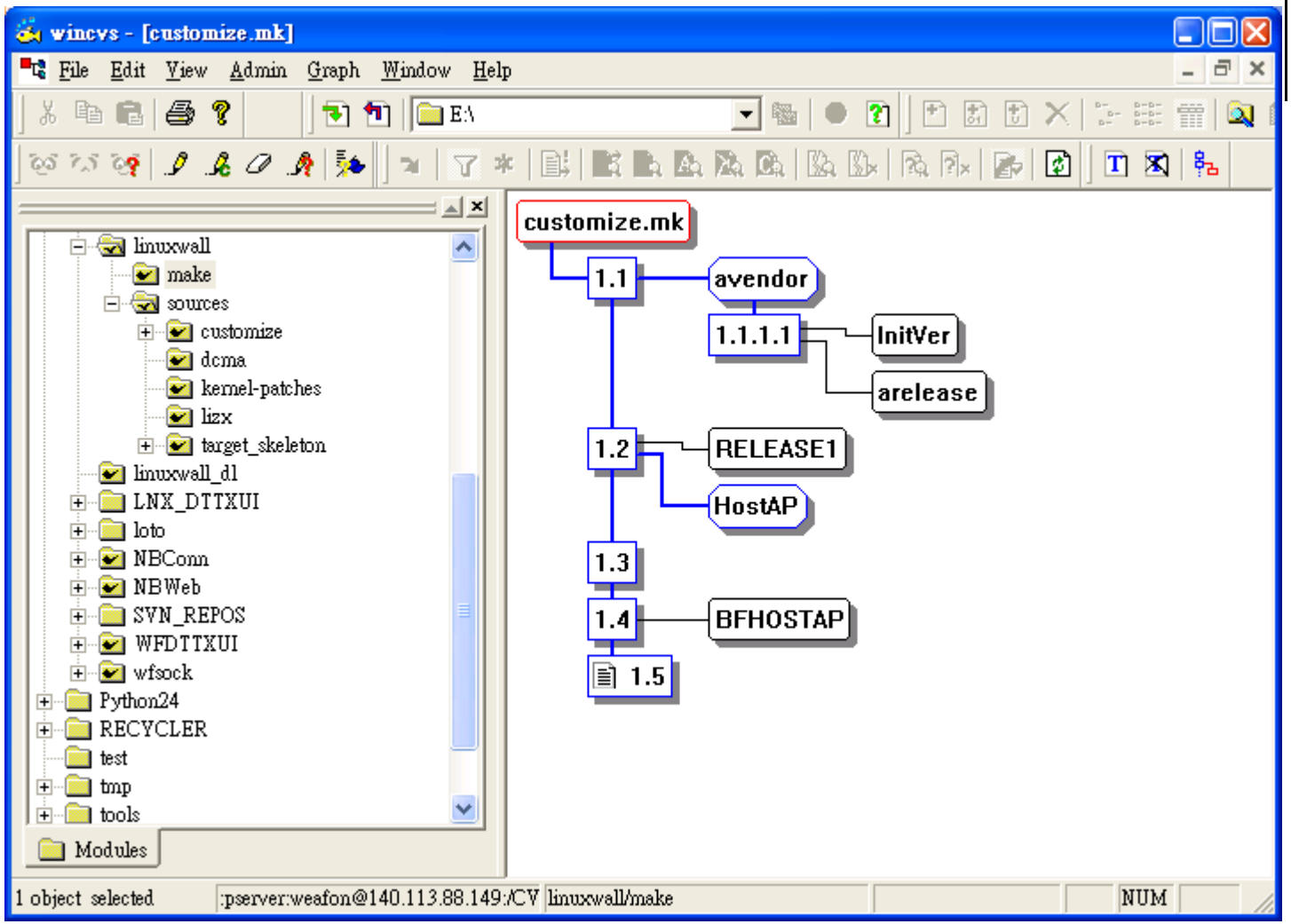
Tips for using CVS

- CVS is also available on MS Windows
  - e.g. Server: CVSNT, Client: WinCVS
  - http://www.wincvs.org/
- Login into the CVS: not another shell
- CVSROOT is an environment variable
  - The configuring is shell-dependent
  - TCSH: `setenv CVSROOT :pserver:`*`username`*`@140.113.88.149:/CVSROOT`
  - BASH: `export CVSROOT=:pserver:`*`username`*`@140.113.88.149:/CVSROOT`
- "cvs add" does not copy files to the server.
  "cvs commit" is needed after "add"
- "cvs add *dirname*" does not add all files
  - you have to cd to this directory
  - keyin "cvs add *"
- http://www.nongnu.org/cvs/

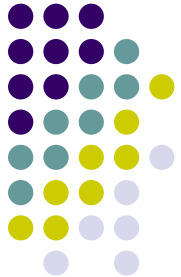
Screenshot of WinCVS client



Screenshot of WinCVS client



SubVersioN (SVN)



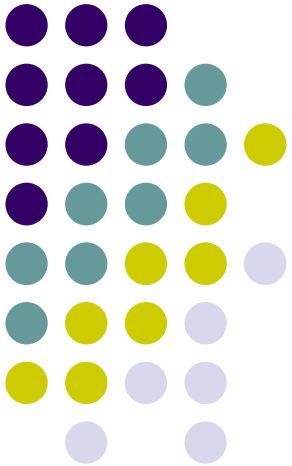
- Most current CVS features
- Directories, renames, and file meta-data are versioned
- Commits are truly atomic
- Choice of database or plain-file repository implementations
- Efficient handling of binary files
- http://subversion.tigris.org/

Wall Project

十機一體閘道器於Linux平台

Security and QoS gateway

From 7-in-1/NetBSD To N-in-1/Linux



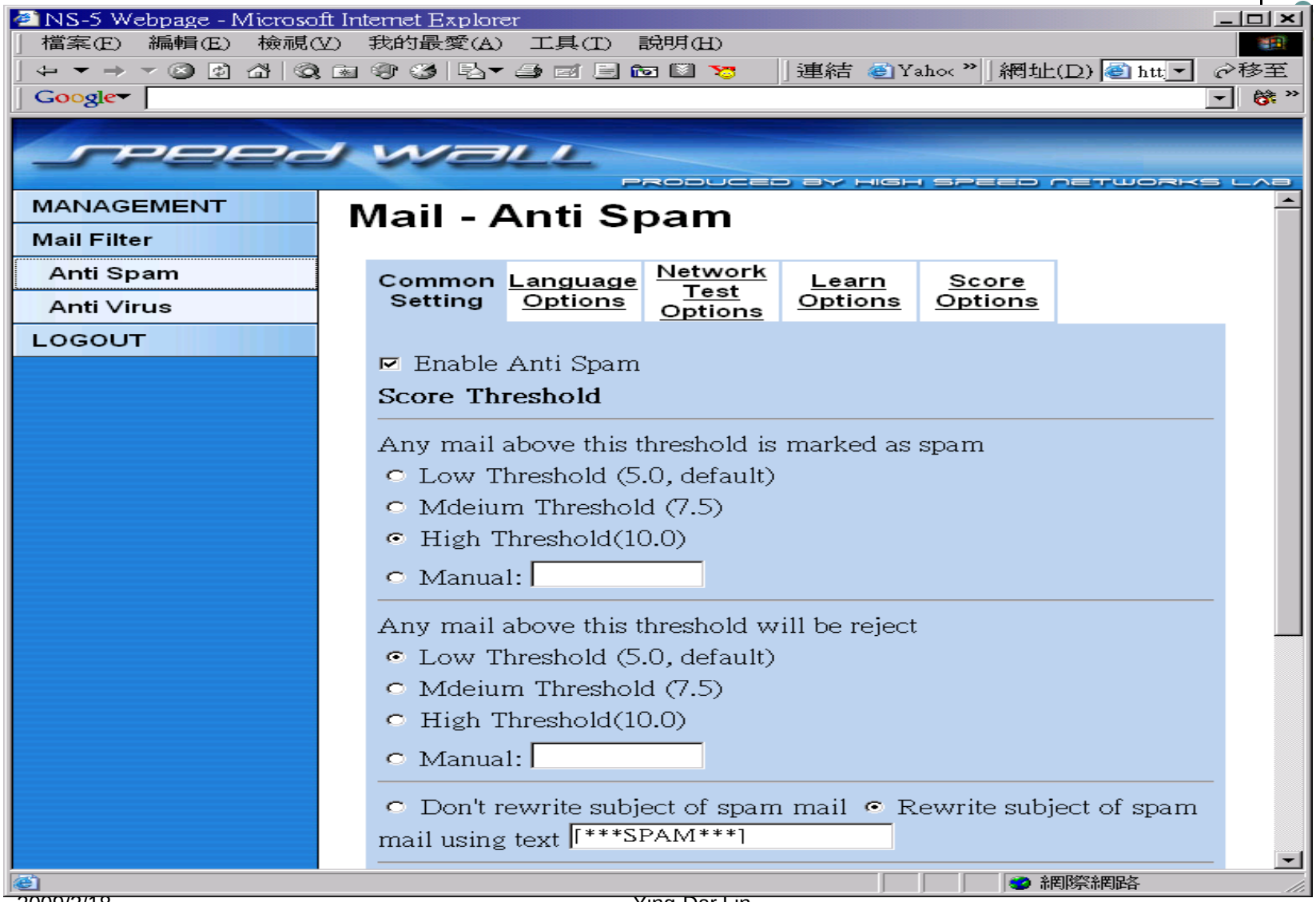
Evolution of Wall

- **7-in-1** (NetBSD)
  - Solving problems on *TCP/IP layer*
  - NAT, Firewall, VPN, Router, IDS, CF, BW magt.
- **10-in-1** (NetBSD)
  - *Content-aware*
  - Anti-Virus, Anti-Spam, CF/Keyword
  - Reducing System Overhead
- **N**-in-1 (**Linux**)
  - *Easy* to add new module
  - UPnP, APP Firewall, SSL-VPN,
  - Transparent Proxy, MSN log, POP3 proxy, Log rotation

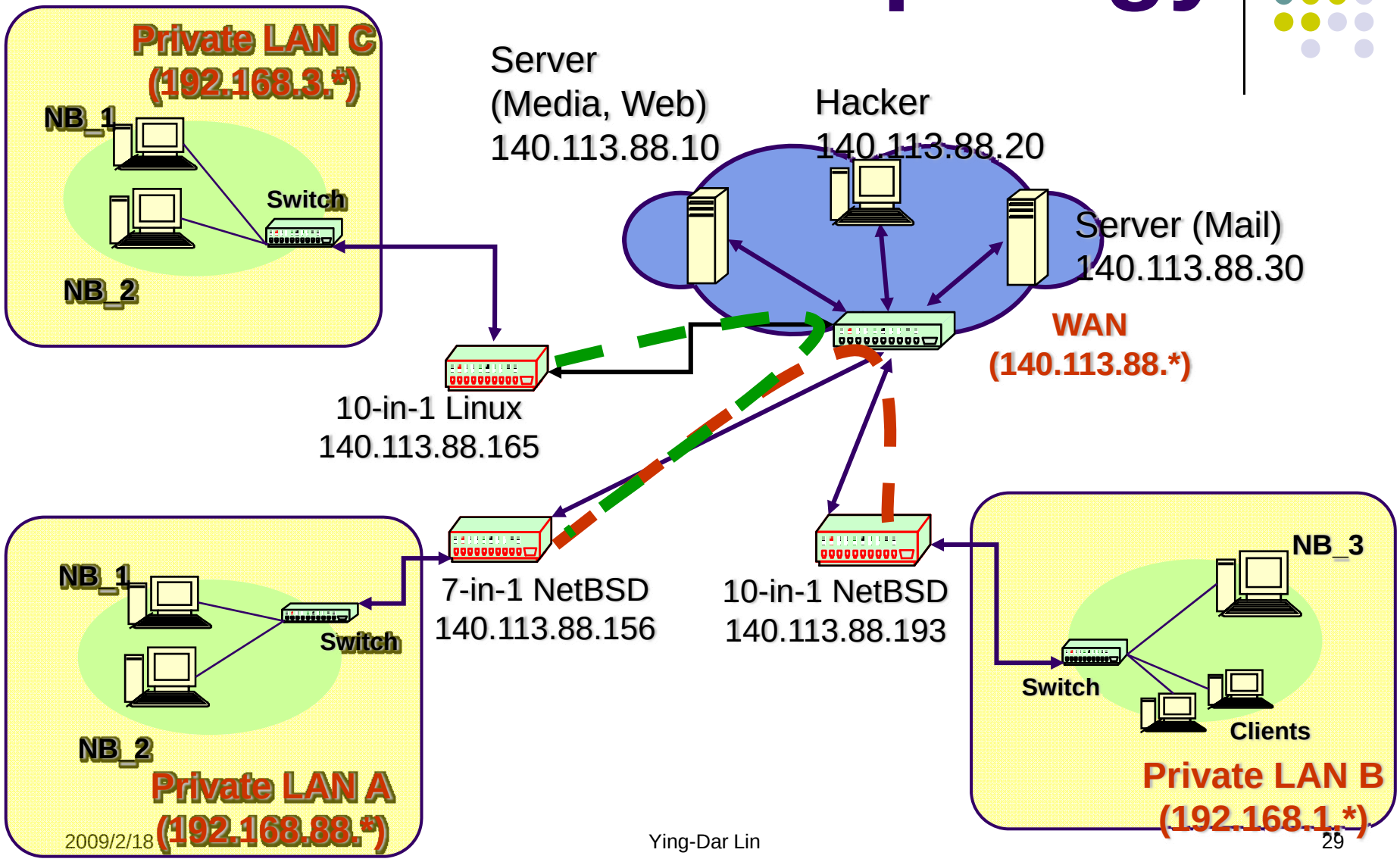
Specification

Connection	LAN, DMZ, WAN, DHCP, DNS relay, Dynamic DNS *Link load balance, Bridge mode*
Security	IPSEC, PPTP, L2TP, SSL-VPN
Firewall	NAT, Firewall, APP firewall, UPNP, *Traffic profiling*
Mail	Anti-Spam, Anti-Virus, POP3 proxy
Web	URL, URL keyword, content keyword, Transparent Proxy
IM	MSN log
BW Control	TC
Management	Web, SSL, FTP, *Log rotation, CRON*
Platform	I386, *IXP (simple version)*

Configuring Wall : Anti-Spam

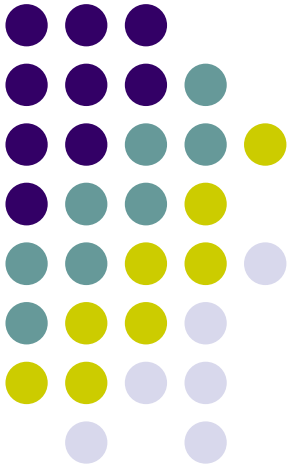


LAB Network Topology

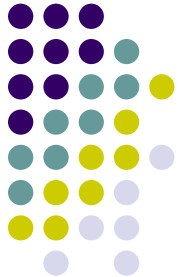


Issues

- Adding a package into the turn-key
- Accelerating the building procedure



Procedures for Adding a Package

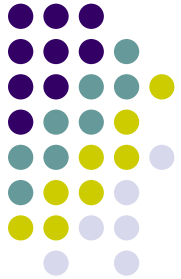


A. Select the open source package
B. Compile it with uClibc (Built in the uClibc development system)
C. Checkout from CVS and Write the xxxx.mk
D. Trim the code size
E. Set the configuration of the package
F. Build and Install the final image
G. Test its operation
H. Add and Commit all files into CVS

B. Compile the package in the development system (I)

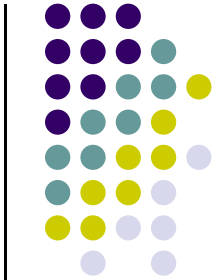
1. *Prepare development system*
   - Download the pre-complied system
   - or Build your own system
2. *Mount the development system*
   - mkdir rootfs
   - su root
   - mount -o loop rootfs.ARCH.FS rootfs
3. *Download the selected package*
4. *Unpack into a directory under rootfs*
   e.g. *rootfs/usr/local/src*

B. Compile the package in the development system (II)



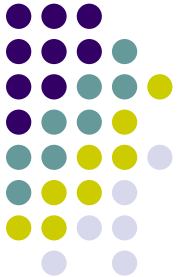
5. Switch into the development system
   - chroot rootfs bin/su –
   - using the bin utils and libraries in the image rootfs.ARCH.FS
6. configure, make, install the package
7. Test and Debug the package
8. Handle all the errors

C. How to Write the xxx.mk?



1. xxx.mk tells the buildroot
   how to make your package
2. Study other xxxx.mk files.
   - http://www.uclibc.org/cgi-bin/cvsweb/buildroot/make/
   - openssh.mk is a good sample.
3. Five parts are included in xxx.mk.
   - *unpacked*: download and unpack
   - *configured*: set the options for make
   - *strip*: downsize the code
   - *make* and *install*
   - clean: clean the built results or intermediate files

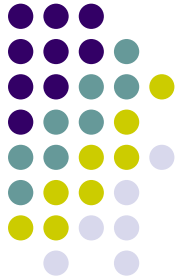
Important Variables Referred in 'xxxxx.mk' (I)



BUILD_DIR	All downloaded packages are unzipped in this working dir.
TARGET_DIR	after compiling, the results should be copied to this directory.
SOURCE_DIR	All patch files are put in this dir.
DL_DIR	All downloaded package are put in this dir
STAGING_DIR	Libraries and header files in this dir can be used as building
TARGET_CROSS	The cross compiler
TARGET_CC	$(TARGET_CROSS)gcc
STRIP	The command to downsize the executing code

All the variables are defined in Makefile

Important Variables Referred in 'xxxxx.mk' (II)



XXXXX_SITE	Tell buildroot where to download the package
XXXXX _DIR	Tell buildroot where to be the working dir
XXXXX _SOURCE	the tar file name of the package
XXXXX _PATCH	the path and filename of patch file (optional)

the four variables are defined in xxx.mk

Tips for Writing xxx.mk

- Remember to add

```
$(TARGET_CONFIGURE_OPTS) \
LD=$(TARGET_CROSS)gcc \
CFLAGS="$(TARGET_CFLAGS)”
```

  in the ‘configured’ part and set

```
CC=$(TARGET_CC)
```

- If some necessary libraries are missing,
  - build the missing libraries as you build a package,
  - then copy them into the dir specified by $(STAGING_DIR).
- if you fail to cross-compile,
  - chroot to the development image
  - build the image manually
  - pack to a tar file
  - in xxx.mk you just ask unpack the tar file

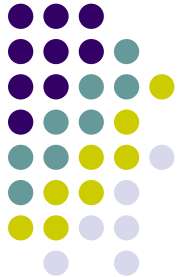
D. Trim the Image Size (I)

- Cut symbol information by STRIP
- Cut help documents
- Remove the static libraries and include files

```
-$(STRIP) --strip-unneeded $(SQUID_DIR)/src/squid
rm –rf  $(TARGET_DIR)/man
rm -rf $(TARGET_DIR)/include/*.h
rm -rf $(TARGET_DIR)/lib/*.a
```

A partial code of squid.mk

Downsize for "squid-2.5.STABLES"



Original Size: 9668KB

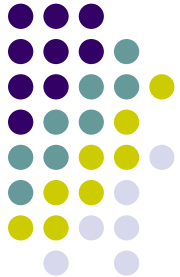
➔9660KB (make optimization and remove .note & .comment)

➔9648KB (remove man file)

➔6140KB (remove html doc)

- Squid program 5504KB

Downsize for gawk-3.1.2



- Original Size: 3172KB

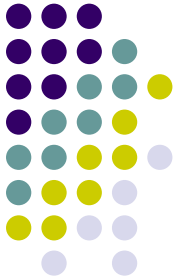
➔3170KB (make optimization and remove .note & .comment)

➔3074KB (remove man files)

➔1890KB (remove info doc)

➔1386KB (remove example files)

Size Distribution (I)



Total 139MB			
bin	1MB	sbin	0.25MB
etc	2.9MB	tmp	0.018MB
home	0.006MB	share	1.1MB
local	0.011MB	www	3.0MB
lib	0.883MB	libexec	0.032MB
usr	128MB		

Size Distribution (II)

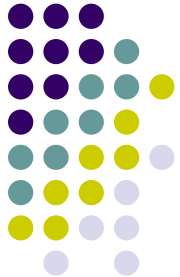
~~xxx.a~~
xxx.so

Files and Dirs in the *usr* dir			
lib	***31MB***	***docs***	***11MB***
sbin	4.2MB	local	45MB
share	2.4MB	postfix	12MB
include	***9.6MB***	bin	14MB

perl

Before Downsizing: 139 MB
After Downsizing:
➔ 128MB (-11MB for doc)
➔ 118.4MB (-9.6 MB for include)
➔ 102.6MB (-15.8 MB for *.a)

G. Build and Install the Image



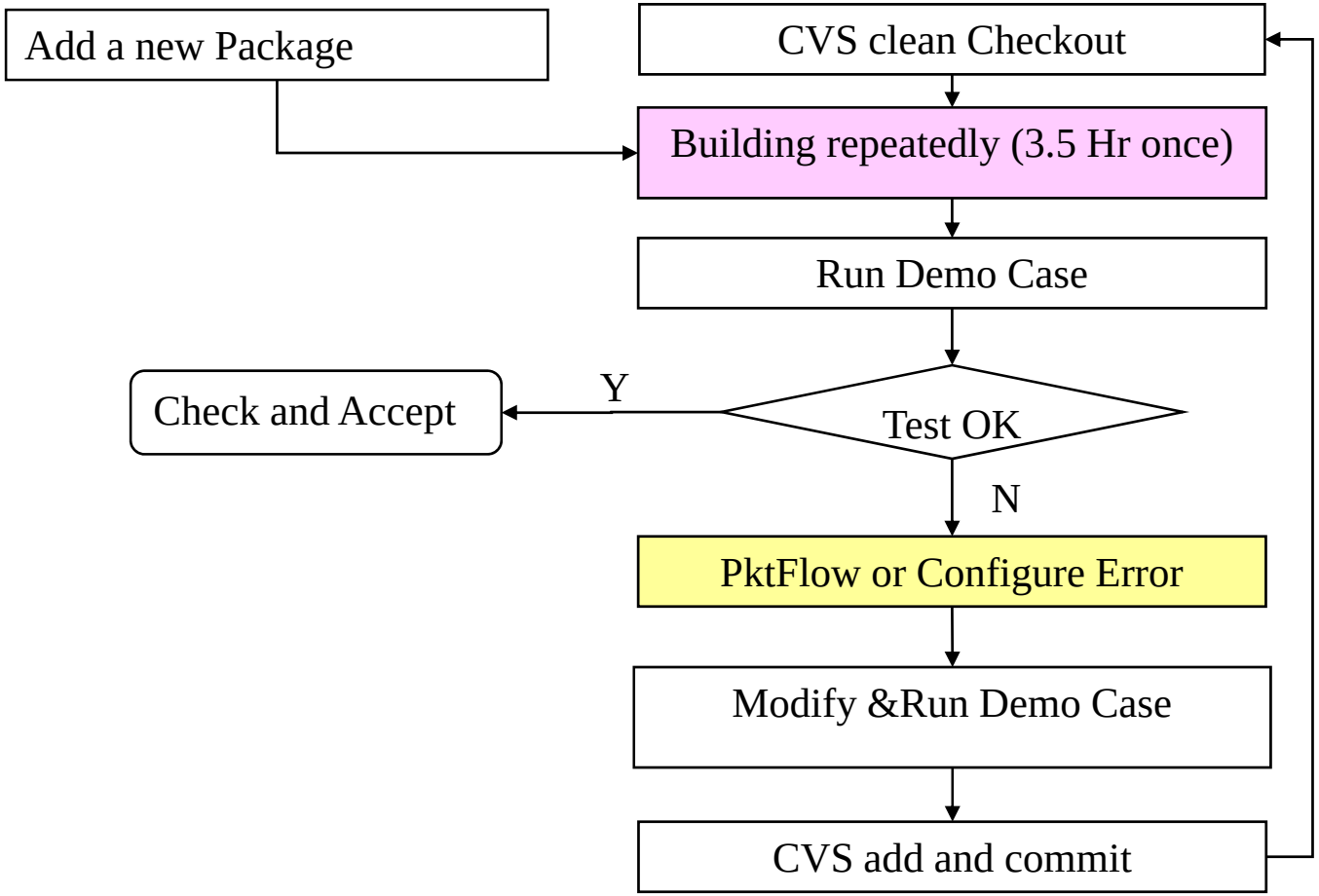
To test your image, you can install the images into a hard disk and then boot on a i386 PC

1	Run ‘make’ to compile and build the kernel and the final image.	
2	fdisk a clean disk to 2 partitions	one for kernel and one for rootfs.i386
3	format and mount the 2 partitions	wall_kernel wall_image
4	mount imgfile rootfs.i386.ext2	mount –o loop rootfs.i386.ext2 xxxx
5	cp bzImage to wall_kernel/boot cp –af xxxx/* wall_image	
7	install boot loader	grub mkdir wall_kernel/boot/grub cp /boot/grub/* wall_kernel/boot/grub write menu.lst in boot/grub ln –s menu.lst grub.conf

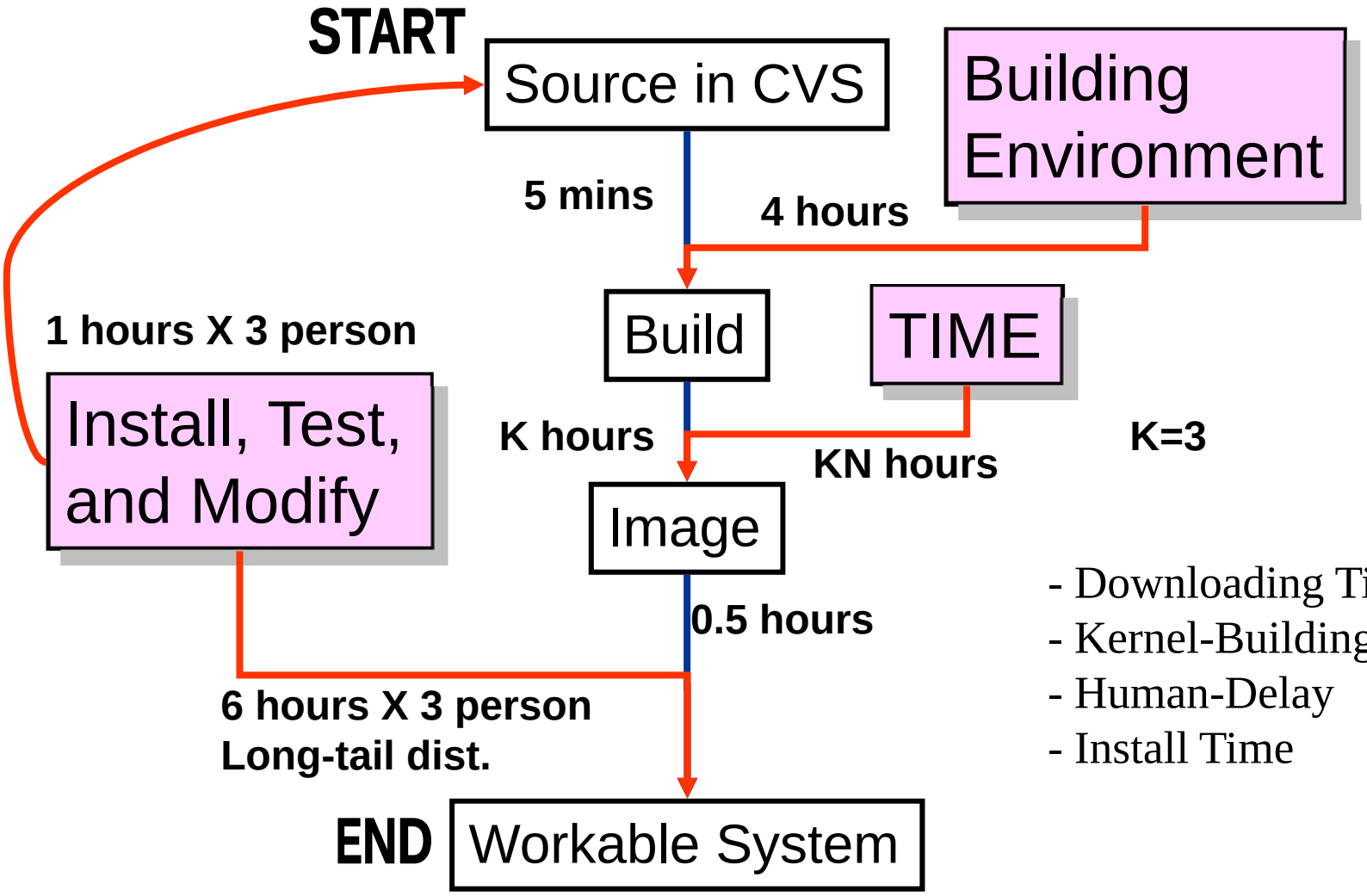
H. Add and Commit all files

- Add the files and dirs into cvs/svn
  - Package packed file (dl/xxxx.gz)
    - **`cd dl`**
    - **`cvs add xxx.gz`**
  - Package make file (package/abcd/abcd.mk)
    - **`cd package`**
    - **`cvs add abcd`**
    - **`cd abcd`**
    - **`cvs add abcd.mk`**
  - Configuration files (package/*customize/source*)
- Commit
    - **`cd buildroot`**
    - **`cvs commit`**

Testing the installed Image



Bottlenecks for getting an workable system



- Downloading Time
- Kernel-Building Time
- Human-Delay
- Install Time

Consumed Time in Building Image

Compiled Items	Time
All packages	154m 47.393s
Kernel only	68m 22.161s
Squid	2m 49.102s
Zlib opssl/ssh	10m 55.443s
p3scan(only tar)	8.04s
Executor	59.936s
Dansguardian	17.687s
gcc	2m 6.387s
Pureftpd	2m 7.61

IPC (CPU 700MHz/256MB)

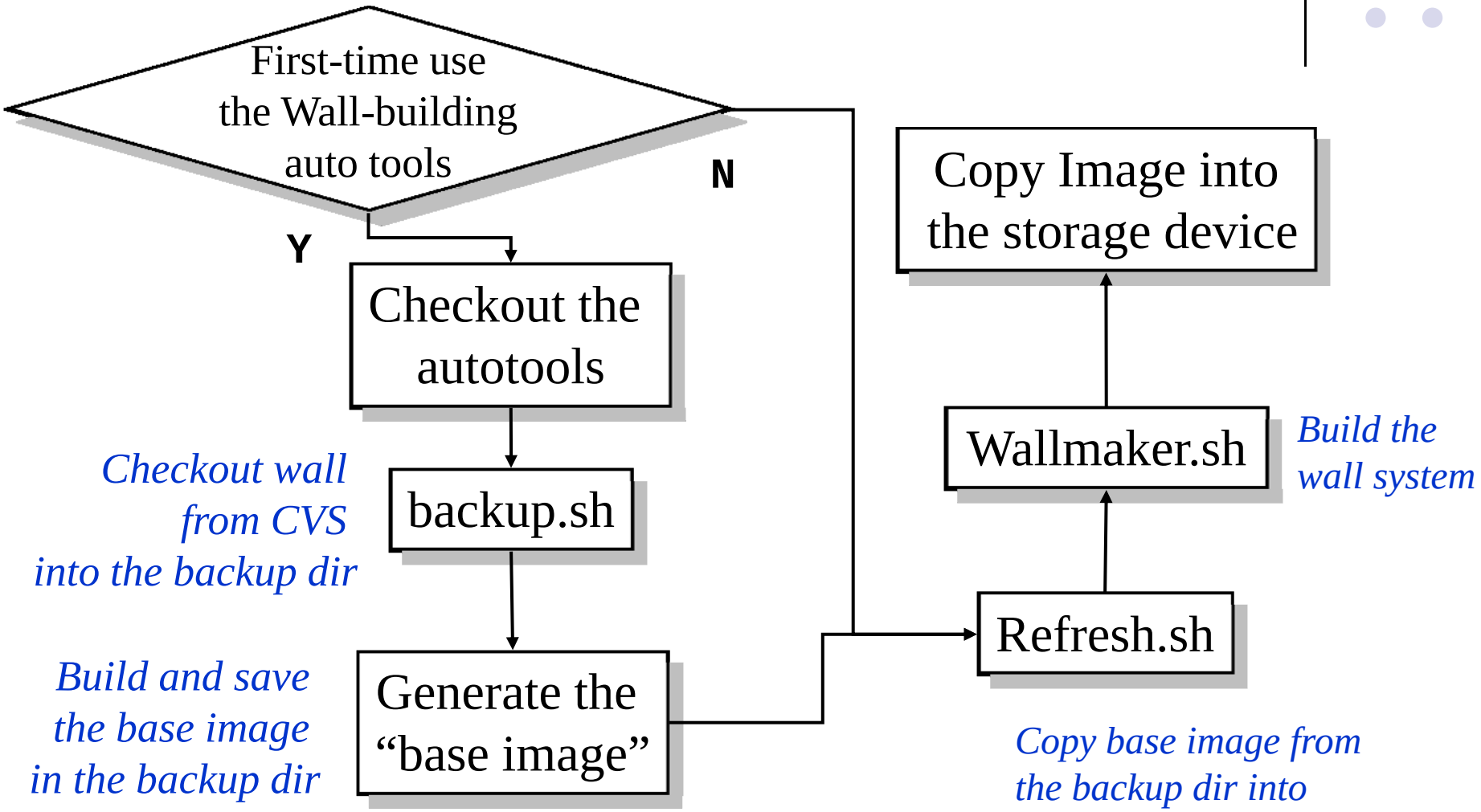
Packages not related to kernel

Squid	Web proxy cache
Zlib	lossless data-compression library
Openssl	Toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols
Openssh	the SSH protocol suite of network connectivity tools
P3scan	full-transparent proxy-server for email clients
Gcc	the GNU Compiler Collection
Bridge tools	IEEE 802.1d ethernet bridging
Dansguardian	web content filter. It filters the actual content of pages based on many methods including phrase matching, PICS filtering and URL filtering.
Pureftpd	a free (BSD), secure, production-quality and standard-conformant FTP server.

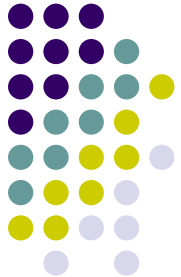
Packages related to kernel

Iptables	the user-space command line program used to configure the packet filtering ruleset and NAT
Freeswan	an implementation of IPSEC & IKE for Linux
Hotplug	lets you plug in new devices and use them immediately
iproute2	a collection of utilites for controlling TCP / IP networking and Traffic Control
pcmcia	a complete PCMCIA support package
linux	Kernel patch
Uclibc	Small c library

Automation Building Procedure

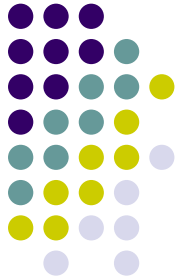


Mini-Project: A Linux-based HTTP server



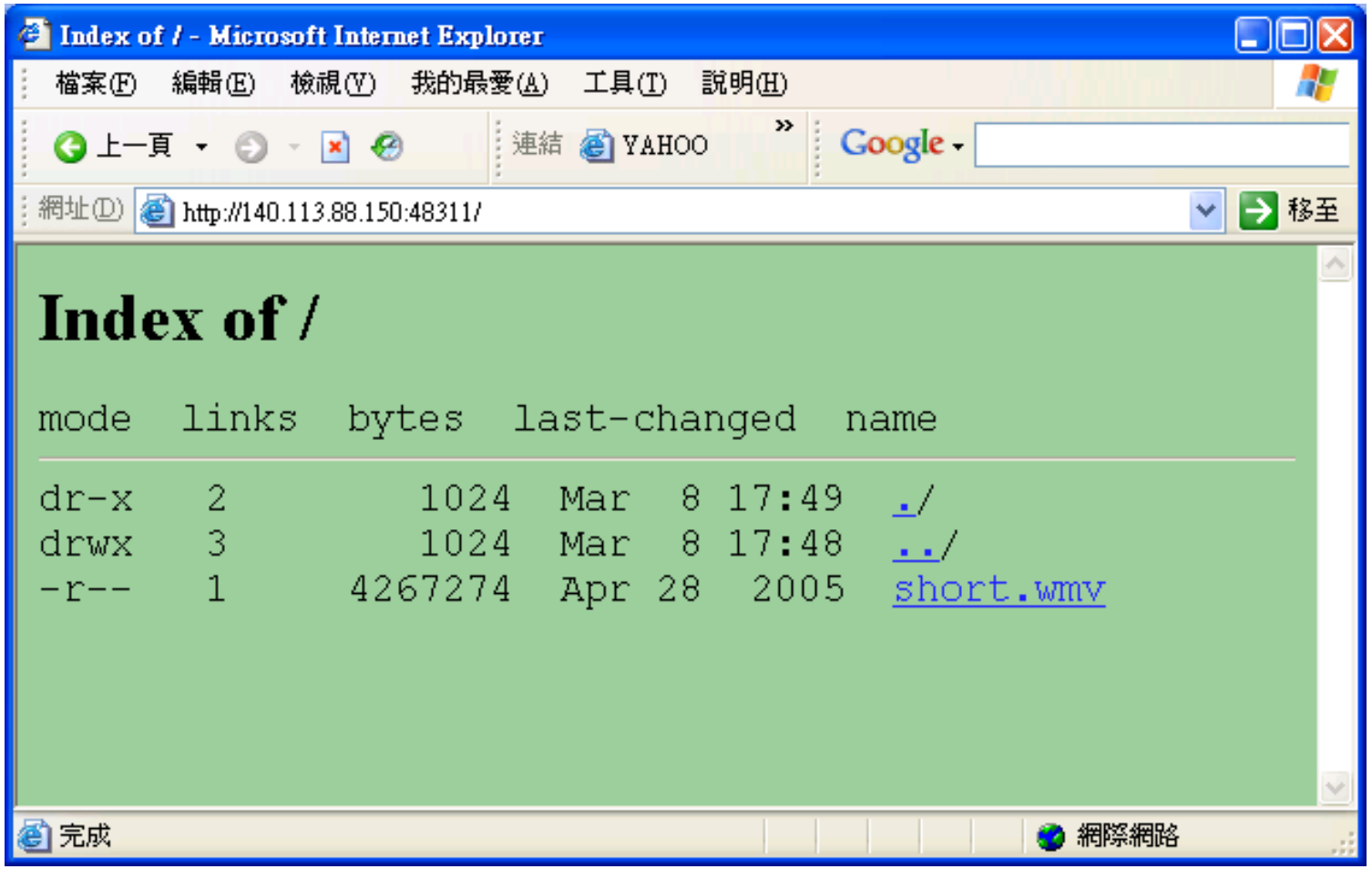
- Purpose:
  - Download source codes by CVS or SVN
  - Be familiar with the uLibc BuildRoot System

System Spec of Mini-Project:

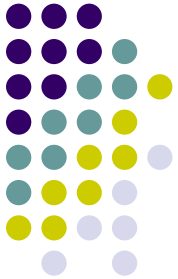


- A http server with port=48311
- The root dir of the server is /tmp/homepage
- The dir should have an video file
- The size of the video file > 4MB
- A user can play the file at another computer.

A example of the result

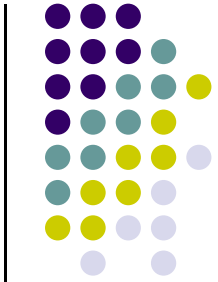


Procedures of Mini-Project (I)



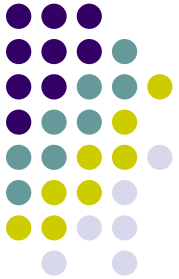
- Install CVS or SVN client
- Download BuildRoot from uCLibc website
- Enable related items
- Copy a 4MB video file
  into the package/customize/source/xxx/xxx
- Build root image
- Chroot to the built image file
- Run the thttpd (thttpd –help)
- Connect from another PC

Discussions of Mini-Project



1. List your procedures and the used commands.
2. Show the screenshots for your results (3 pics).
3. Which packages are built in the image?
4. If without the video file, what is the length of the image file?
5. Do you select the “customize” package? What is its purpose?

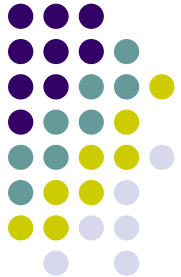
Discussions of Mini-Project



6. After all procedures, show the dirs and files in the *package/customize* directory (ls –alR package/customize)

7. Besides SVN and BuildRoot, do you install any packages in your build system for the mini-project? What are their purposes?
8. How many times “make” do you type before successfully getting the final image? Why?
9. How much time do you spent for the project?

Topic of Term-Project: A Linux-based Firewall



- Purpose:
  - Build a project under your own cvs or svn server
  - Be familiar with the uCLibc BuildRoot system
  - Learn how to add packages into BuildRoot sys.
  - Simple configuration on iptables

System Spec of Term-Project:

- Your own CVS or SVN server
  - Have your own BuildRoot project
  - Some files are revised and have multiple versions
- Build a firewall which blocks all direct connections
  - is running on a (virtual) PC
  - provide a private LAN
- Provide a http proxy
  - Filter the pornography web pages
  - You may need the two open source packages: Squid and Dansguardian
- Downsize the image
- Alternating Platforms

Discussions of Term-Project

1. List your procedures and used commands. How you create a project in your cvs or svn server? How you add packages into BuildRoot sys?
2. Print your own xxxx.mk and explain it.
3. Show the screenshots for your results.
4. Which packages are built in the image?
5. The size of the built image? Before and after downsize? Describe what are removed after downsizing.
6. How can you provide the pornography filter?
7. How many times “make” do you type before getting the final image? Why?
8. How much time do you spent for the project?