



Embedded Operating System

Che-Wei Chang

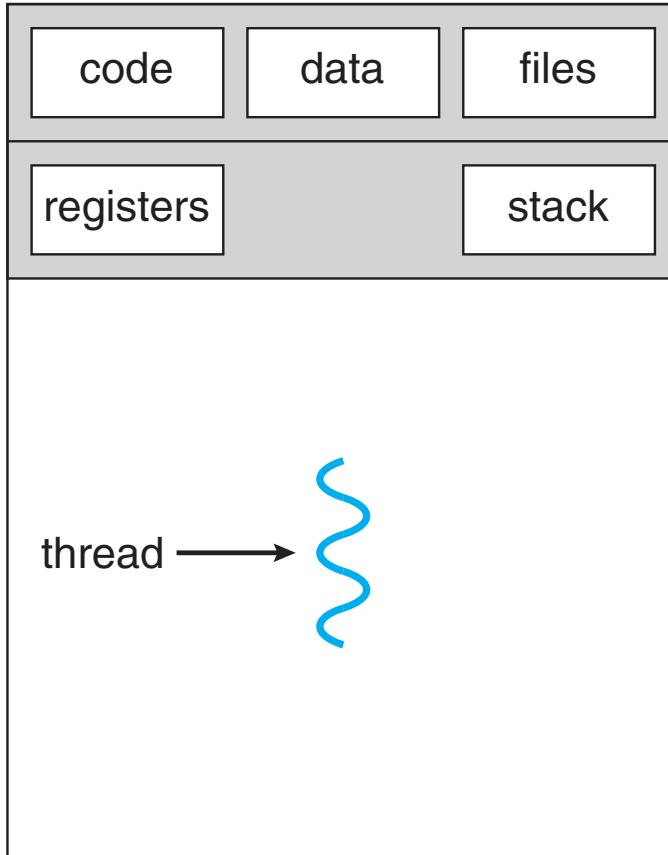
chewei@mail.cgu.edu.tw

Department of Computer Science and Information
Engineering, Chang Gung University

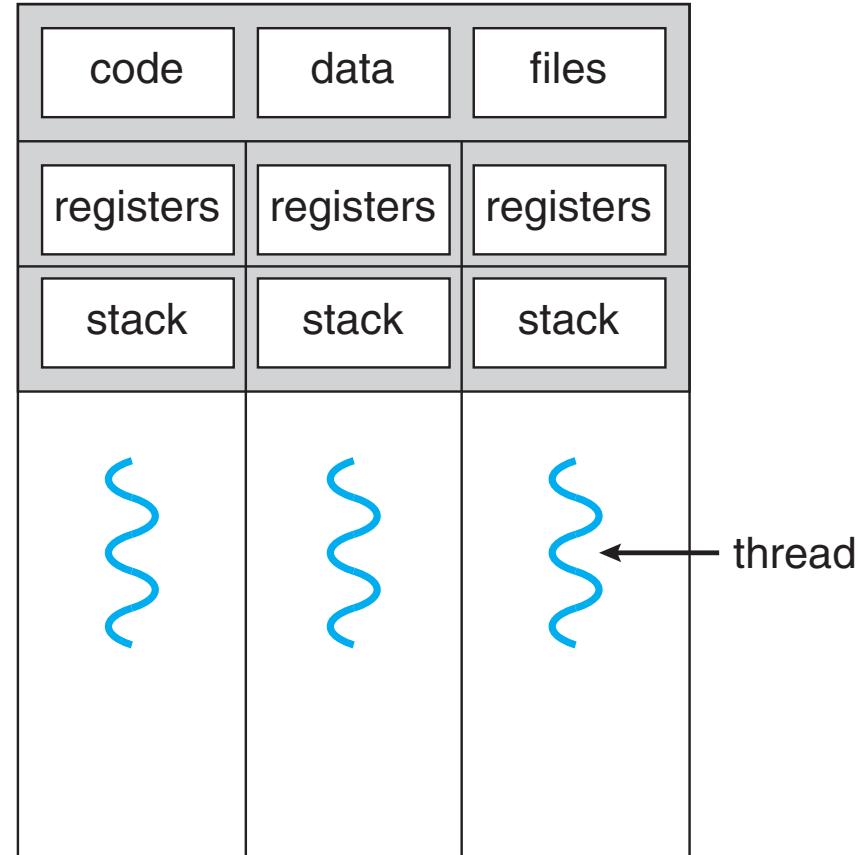


Concepts of Pthread

Single and Multithreaded Processes



single-threaded process



multithreaded process

Definition of Threads

- ▶ A thread is defined as **an independent stream of instructions** that can be scheduled to run
- ▶ To the software developer, the concept of a "**procedure**" that **runs independently from its main program** may best describe a thread
- ▶ To go one step further, imagine a main program that contains a number of procedures
 - Then imagine all of these procedures being able to be scheduled to run simultaneously and/or independently by the operating system
 - That would describe a "**multi-threaded**" program

Reference: <https://computing.llnl.gov/tutorials/pthreads/>

A Process

- ▶ A UNIX process is created by the operating system
- ▶ It requires a fair amount of overhead for the creation and context switch among processes
- ▶ Processes contain information about program resources and program execution state, including:
 - Process ID, process group ID, user ID, and group ID
 - Working directory
 - Program instructions
 - Registers, stack, heap
 - File descriptors
 - Signal actions
 - Shared libraries
 - Inter-process communication tools
 - Message queues, pipes, semaphores, and shared memory

A Thread

- ▶ Threads use and exist within the process resources
- ▶ Threads are able to be scheduled by the operating system and run as independent entities
- ▶ Threads duplicate only the bare essential resources that enable them to exist as executable code
 - Stack pointer
 - Registers
 - Scheduling properties (such as policy or priority)
 - Set of pending and blocked signals
 - Thread specific data

User Threads and Kernel Threads

- ▶ User threads
 - Management done by user-level threads library
 - Three primary thread libraries:
 - POSIX Pthreads
 - Win32 threads
 - Java threads
- ▶ Kernel threads
 - Supported by the Kernel
 - Examples – virtually all general purpose operating systems, including: Windows, Solaris, Linux, Tru64 UNIX, Mac OS X
 - **Linux supports one-to-one mapping for a Pthread to a kernel thread**, which is the environment for this lab exercise

History of Pthread

- ▶ Historically, hardware vendors have implemented their own proprietary versions of threads which are not portable
- ▶ In order to take full advantage of the capabilities provided by threads, a **standardized programming interface** was required
- ▶ For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995)
 - ➔ Thus, it is called Pthread (POSIX thread)

Pthreads vs Processes

- For 50,000 process/thread creations, the time is measured in seconds

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Source: <https://computing.llnl.gov/tutorials/pthreads/>

Considerations for Designing Parallel Programs

- ▶ Problem partitioning
- ▶ Load balancing
- ▶ Communications
- ▶ Data dependencies
- ▶ Synchronization and race conditions
- ▶ Memory issues
- ▶ I/O issues
- ▶ Debugging efforts

Pthreads API

- ▶ Thread Management:
 - Routines that work directly on threads, such as creating, detaching, joining
- ▶ Mutex:
 - Routines that deal with synchronization, called a "mutex", which is an abbreviation for "mutual exclusion"
 - Mutex functions are provided for creating, destroying, locking and unlocking mutexes
- ▶ Condition Variable:
 - Routines to create, destroy, wait and signal based upon specified variable values
 - Functions to set/query condition variable attributes are also included



Tools for Using Pthread

Compiling Threaded Programs

Compiler / Platform	Compiler Command	Description
INTEL Linux	icc -pthread	C
PGI Linux	icpc -pthread	C++
GNU Linux, Blue Gene	pgcc -lpthread pgCC -lpthread	C
GNU Linux, Blue Gene	gcc -lpthread g++ -lpthread	C++
IBM Blue Gene	bgxlC_r / bgcc_r bgxlC_r, bgxlC++_r	GNU C GNU C++ C (ANSI / non-ANSI) C++

Creating and Terminating

- ▶ Routines
 - `pthread_create` (`thread,attr,start_routine,arg`)
 - `pthread_exit` (`status`)
 - `pthread_cancel` (`thread`)
 - `pthread_attr_init` (`attr`)
 - `pthread_attr_destroy` (`attr`)
- ▶ Initially, your `main()` program comprises a single, default thread
- ▶ All other threads must be explicitly created by the programmer
- ▶ The function `pthread_create()` creates a new thread and makes it executable
 - Return 0 for success

pthread_create() Arguments

- ▶ **thread:**
 - An identifier for the new thread returned by the subroutine
- ▶ **attr:**
 - An opaque attribute object that may be used to set thread attributes
 - You can specify a thread attributes object, or use NULL for the default values
- ▶ **start_routine:**
 - The C routine that the thread will execute once it is created
- ▶ **arg:**
 - A single argument that may be passed to the start_routine
 - It must be passed by reference as a pointer cast of type void
 - NULL may be used if no argument is to be passed.

An Example pthread_create()

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS      5

void *PrintHello(void *threadid)
{
    printf("\n%d: Hello World!\n", threadid);
    pthread_exit(NULL);
}

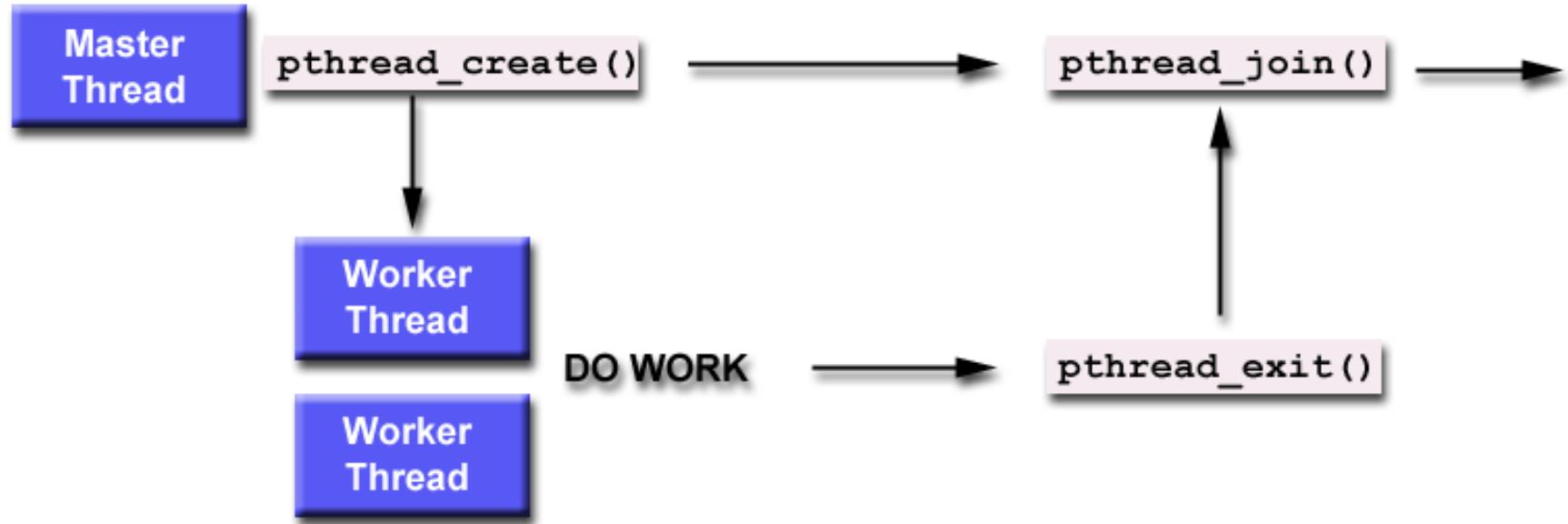
int main (int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t;
    for(t=0; t<NUM_THREADS; t++){
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

Function pointer for the routine

The type of threads

Thread: thread[t], attribute: default, function: PrintHello, input: t

Joining Threads



Function pthread_join()

- ▶ The Format:

→ int pthread_join(pthread_t thread, void **value_ptr);

- ▶ pthread_t thread is for the thread to wait

- ▶ void ** value_ptr is for the return value of the thread

- ▶ The int return value:

- 0 is for success

- Others are for errors

Mutex

- ▶ Create a mutex:
 - `pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;`
- ▶ Lock a mutex:
 - `pthread_mutex_lock(&count_mutex);`
 - It is a blocking lock
 - `pthread_mutex_trylock ((&count_mutex));`
 - It is a non-blocking lock
- ▶ Release a mutex:
 - `pthread_mutex_unlock(&count_mutex);`

An Example with Mutex (1 / 2)

```
#include <stdio.h>
#include <pthread.h>
#define TCOUNT 10
#define NUM_THREADS 3
int count = 0;
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
int thread_ids[3] = {0, 1, 2};
```

```
int inc_count(void *idp)
{
    int i;
    for(i=0; i < TCOUNT; i++)
    {
        pthread_mutex_lock(&count_mutex);
        count++;
        pthread_mutex_unlock(&count_mutex);
        printf("inc_counter():thread %d, old count %d, new count %d, \n", (int)idp, count-1, count);
        sleep(1);
    }
    return 0;
}
```

Initialize a mutex

count++ should be protected by a mutex

An Example with Mutex (2/2)

```
int main()
{
    int i;
    pthread_t threads[3];
    pthread_create(&threads[0], NULL, (void *)&inc_count, (void *)thread_ids[0]);
    pthread_create(&threads[1], NULL, (void *)&inc_count, (void *)thread_ids[1]);
    pthread_create(&threads[2], NULL, (void *)&inc_count, (void *)thread_ids[2]);

    for(i=0; i<NUM_THREADS; i++)
    {
        pthread_join(threads[i], NULL);
    }
    printf("done ... terminate with kill command or CRTL+C\n");
    return 0;
}
```

The diagram illustrates the annotations for the code:

- Thread ID:** Points to the first argument in each `pthread_create` call, which is `&threads[0]`, `&threads[1]`, and `&threads[2]`.
- Default Attribute:** Points to the second argument in each `pthread_create` call, which is `NULL`.
- Function Pointer:** Points to the third argument in each `pthread_create` call, which is `(void *)&inc_count`.
- Input Value:** Points to the fourth argument in each `pthread_create` call, which is `(void *)thread_ids[0]`, `[1]`, and `[2]`.
- Wait the previous three threads:** Points to the `i < NUM_THREADS` condition in the `for` loop, indicating that the threads will be joined sequentially.



Preparation

Notices

- ▶ No food, no drink
- ▶ The evaluation boards are quite expensive
- ▶ Do not do anything else to crash the PC
- ▶ Do not update the OS nor tools to keep the consistency
- ▶ Remember the number of your evaluation board
 - Check the items before you use them
 - Check the items before you return them
- ▶ No rubbish

What are We Going to Do?

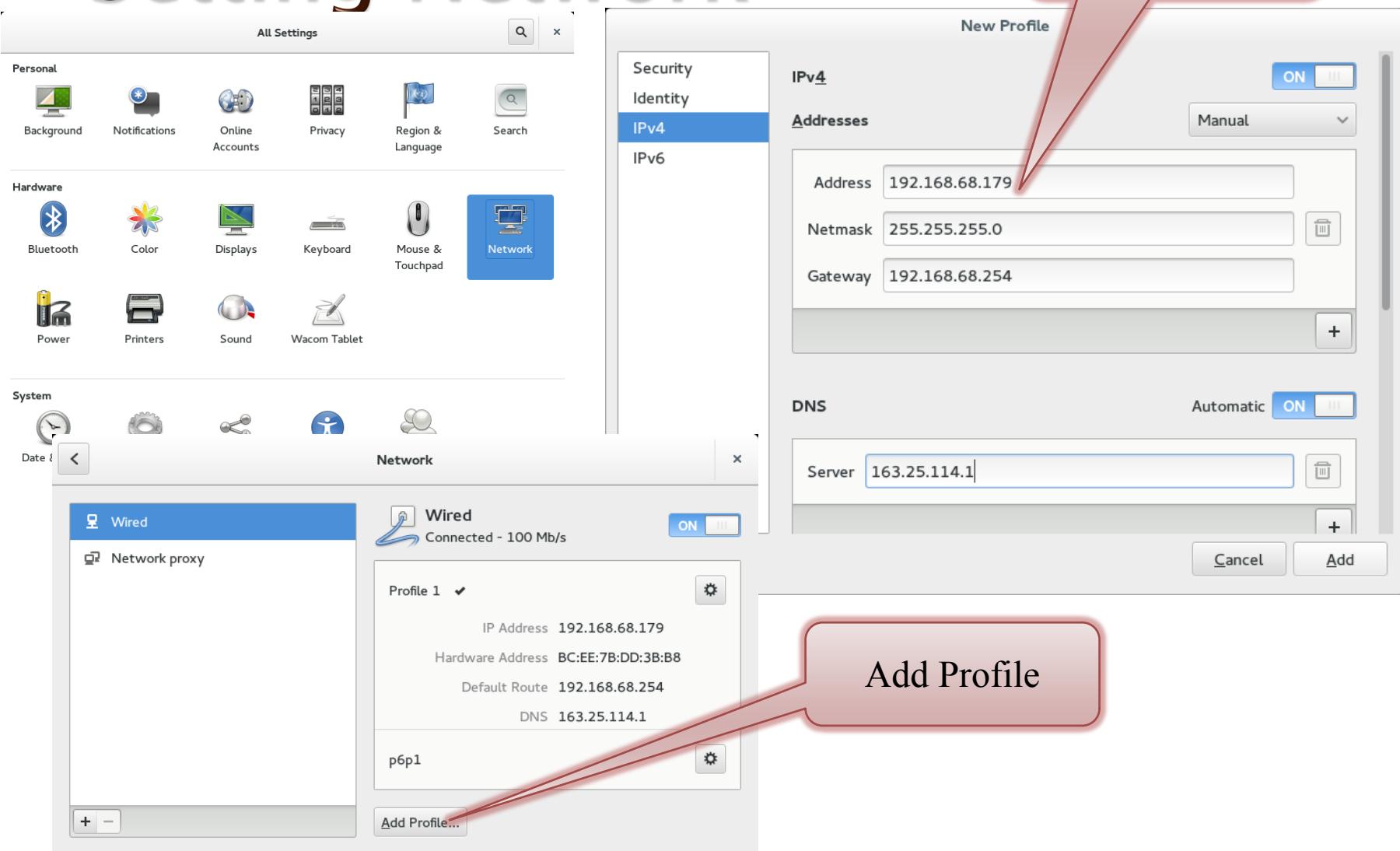
- ▶ Build the Cross Development Toolchain
- ▶ Build the Linux Kernel
 - ➔ Check Point 1: uImage
- ▶ Setup a TFTP Server
- ▶ Setup NFS Server
 - ➔ Check Point 2: Test the Services
- ▶ Setup the Target Board
- ▶ Download the Linux Kernel
 - ➔ Check Point 3: Try the Linux Kernel
- ▶ Write Multi-thread programs on TI OMAP Evaluation Board
 - ➔ Check Point 4: Test it
 - ➔ Check Point 5: Modify the program
 - ➔ Final Check Point: Use Mutex to protect your program

Fedora Linux

- ▶ The Fedora Project was created in late 2003
- ▶ We are using the version 20
- ▶ Package manager: RPM
- ▶ Update method Yum
- ▶ Default user interface: GNOME 3
 - Password: 123456
 - Select the language: Taiwan
 - WindowsKey+Space to change the input language
 - Activities → Search: terminal → to get the terminal
 - Edit → Profile Preferences → Colors → Uncheck “use colors from system theme”
 - Click the icon at the right-top corner for network setting



Setting Network



vi— A Screen-Oriented Text Editor

- ▶ vi is widely supported by Unix-like operating system
- ▶ Normal mode
 - Move, search, copy, paste, delete,...
 - Press i, I, a, A, o, O,... to change to the insert mode
 - Press : for the command mode
- ▶ Command mode
 - Save, quit, load, split,...
 - After enter the command, it will be back to the normal mode
- ▶ Insert mode
 - Move and input anything
 - Press ESC to go back to the normal mode

vi Commands

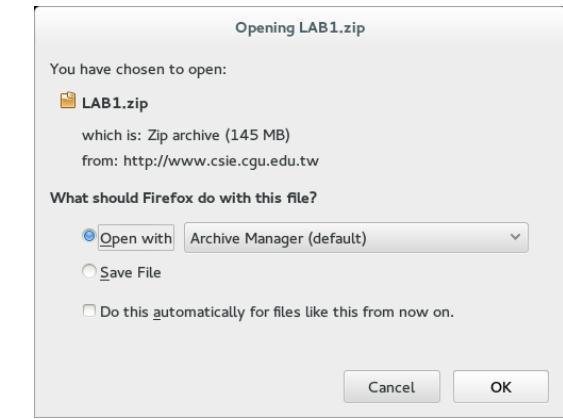
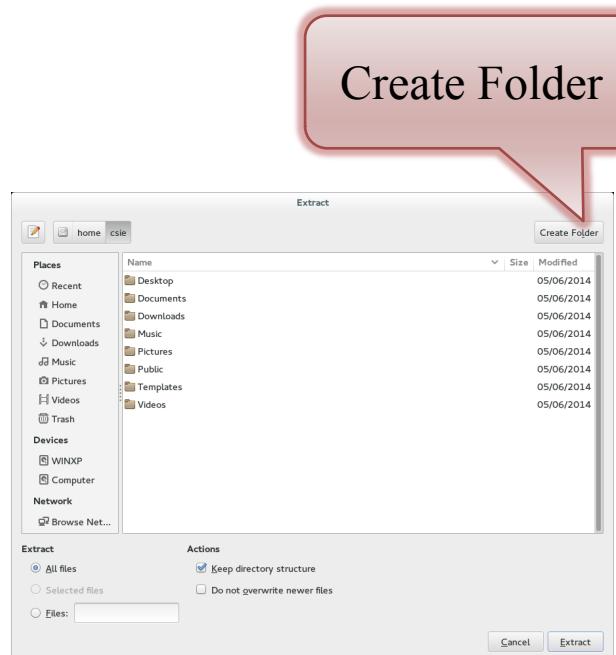
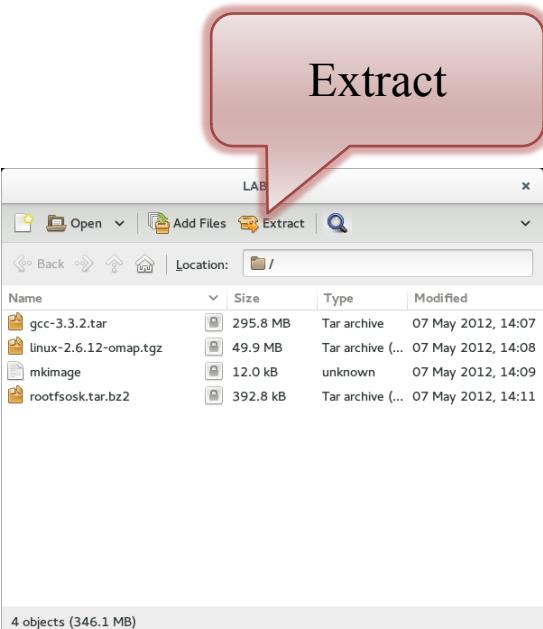
- ▶ Press ‘i’ to get the insert mode
- ▶ Key-in anything
- ▶ Press ‘ESC’ to go back the normal mode
- ▶ Press ‘:→w→q→ENTER” to save and quit
- ▶ Please search for some tutorial of vi and study by yourself



Build the Linux Kernel and Setup Services on TI OMAP

Download Files

- ▶ Download the tools from the course website and extract the files



Download Files

- ▶ You will need the following files
 - linux-2.6.12-omap.tgz → the kernel source code
 - gcc-3.3.2.tar → some gcc extension for this lab
 - mkimage → some script which is used when compiling kernel
 - rootfsosk.tar.bz2 → the content of the root filesystem
- ▶ You need the root privilege for the following actions
 - *su* (the password is 123456) → change to root
 - *cd /home/csie/LAB1*
 - *cp linux-2.6.12-omap.tgz /opt/linux-2.6.12-omap.tgz*
 - *cp gcc-3.3.2.tar /opt/gcc-3.3.2.tar*
 - *chmod +x mkimage*
 - *cd /opt*
 - *tar xvf gcc-3.3.2.tar*
 - *tar zxvf linux-2.6.12-omap.tgz*
 - *cp /home/csie/LAB1/mkimage /opt/usr/local/arm/3.3.2/bin/mkimage*

Prepare the Compiling Environment

- ▶ Set Path
 - *export PATH=\$PATH:/opt/usr/local/arm/3.3.2/bin* → for every terminal session, before you compile the kernel
 - *export LANG=en*
- ▶ Install Tools
 - *yum -y install gcc* → compiler tools
 - *yum -y install glibc.i686* → library for 32bit Linux kernel
 - *yum -y install minicom* → minicom is the utility for the serial port connection

Build the Linux Kernel

- ▶ Go to the kernel source directory (be the root)
 - *cd /opt/linux-2.6.12*
- ▶ Set the kernel configuration
 - *make omap_osk_5912_defconfig*
- ▶ Compile the kernel
 - *make ulmage*
- ▶ Prepare the root filesystem
 - *cp /home/csie/LAB1/rootfsosk.tar.bz2 /tmp/rootfsosk.tar.bz2*
 - *cd /tmp*
 - *tar jxvf rootfsosk.tar.bz2*

Check Point 1

- ▶ Now, you should have the compiled kernel
- ▶ The kernel image is at:
`/opt/linux-2.6.12/arch/arm/boot/uImage`
- ▶ The root filesystem for the evaluation board is at:
`/tmp/roorfs2.6`

Set the Network Services

- ▶ Disable the Firewall (it is not a good idea, only for this lab exercise)
 - *systemctl stop firewalld*
 - *systemctl disable firewalld*
- ▶ Set the TFTP Service
 - *yum -y install tftp-server tftp* → tftp is used to download the kernel image
 - *vi /etc/xinetd.d/tftp*
 - Find `disable = yes`
 - Change it to `disable = no`
 - ~~*/sbin/chkconfig xinetd on*~~
 - *systemctl start tftp.socket*
 - ~~*/sbin/service xinetd start*~~
 - *systemctl enable tftp.socket*
- ▶ Set the NFS Service
 - *yum -y install nfs-utils* → nfs for the root filesystem
 - *vi /etc/exports*
 - Add the line `/tmp/rootfs2.6 *(rw,fsid=1,no_root_squash)`
 - *exportfs -rv*
 - *systemctl start rpcbind.service*
 - *systemctl start nfs-mountd.service*

Test the Network Services

- ▶ You need a friend for the following test
 - One be the server and the other be the client
 - Switch the roles and do it again
- ▶ Test TFTP
 - Server side:
 - *vi /var/lib/tftpboot/testfile* → and then key something
 - Client side:
 - *tftp 192.168.68.xxx* (xxx is for the server IP)
 - *get testfile*
 - *quit*
 - *cat testfile*
- ▶ Test NFS
 - Server side:
 - Client side:
 - *mkdir /home/csie/nfstest*
 - *mount -t nfs 192.168.68.xxx:/tmp/rootfs2.6 /home/csie/nfstest*
 - *cd /home/csie/nfstest*
 - *ls*
 - *cd /*
 - *umount /home/csie/nfstest*

Check Point 2

- ▶ Now, you have enabled the TFTP and NFS services on your PC
- ▶ TFTP and NFS are properly working now

Set the Minicom (1 / 3)

- ▶ Enter the setting menu
 - *minicom -s*

```
+----[configuration]-----+
| Filenames and paths
| File transfer protocols
| Serial port setup
| Modem and dialing
| Screen and keyboard
| Save setup as dfl
| Save setup as..
| Exit
| Exit from Minicom
+-----+
```

- ▶ Serial port setup → press the letter to change it

```
+-----+
| A -  Serial Device      : /dev/ttyS0
|
| C -  Callin Program     :
| D -  Callout Program    :
| E -  Bps/Par/Bits       : 115200 8N1
| F -  Hardware Flow Control : No
| G -  Software Flow Control : No
|
| Change which setting?  |
+-----+
```

Set the Minicom (2/3)

► Modem and dialing

```
+-----[Modem and dialing parameter setup]-----+
| A - Init string .....
| B - Reset string .....
| C - Dialing prefix #1....
| D - Dialing suffix #1....
| E - Dialing prefix #2.... ATDP
| F - Dialing suffix #2.... ^M
| G - Dialing prefix #3.... ATX1DT
| H - Dialing suffix #3.... ;X4D^M
| I - Connect string ..... CONNECT
| J - No connect strings .. NO CARRIER           BUSY
|                               NO DIALTONE          VOICE
| K - Hang-up string ..... ~~~++~ATH^M
| L - Dial cancel string .. ^M
|
| M - Dial time ..... 45      Q - Auto bps detect ..... No
| N - Delay before redial . 2      R - Modem has DCD line .. Yes
| O - Number of tries ..... 10     S - Status line shows ... DTE speed
| P - DTR drop time (0=no). 1      T - Multi-line untag .... No
|
| Change which setting? [ ]  Return or Esc to exit. Edit A+B to get defaults.
+-----+
```

```
+-----[configuration]----+
| Filenames and paths
| File transfer protocols
| Serial port setup
| Modem and dialing
| Screen and keyboard
| Save setup as dfl
| Save setup as..
| Exit
| Exit from Minicom
+-----+
```

Set the Minicom (3/3)

- ▶ Save and leave the setting interface

```
+----[configuration]-----+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl         |
| Save setup as..          |
| Exit                      |
| Exit from Minicom        |
+-----+
```

```
+----[configuration]-----+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl         |
| Save setup as..          |
| Exit                      |
| Exit from Minicom        |
+-----+
```

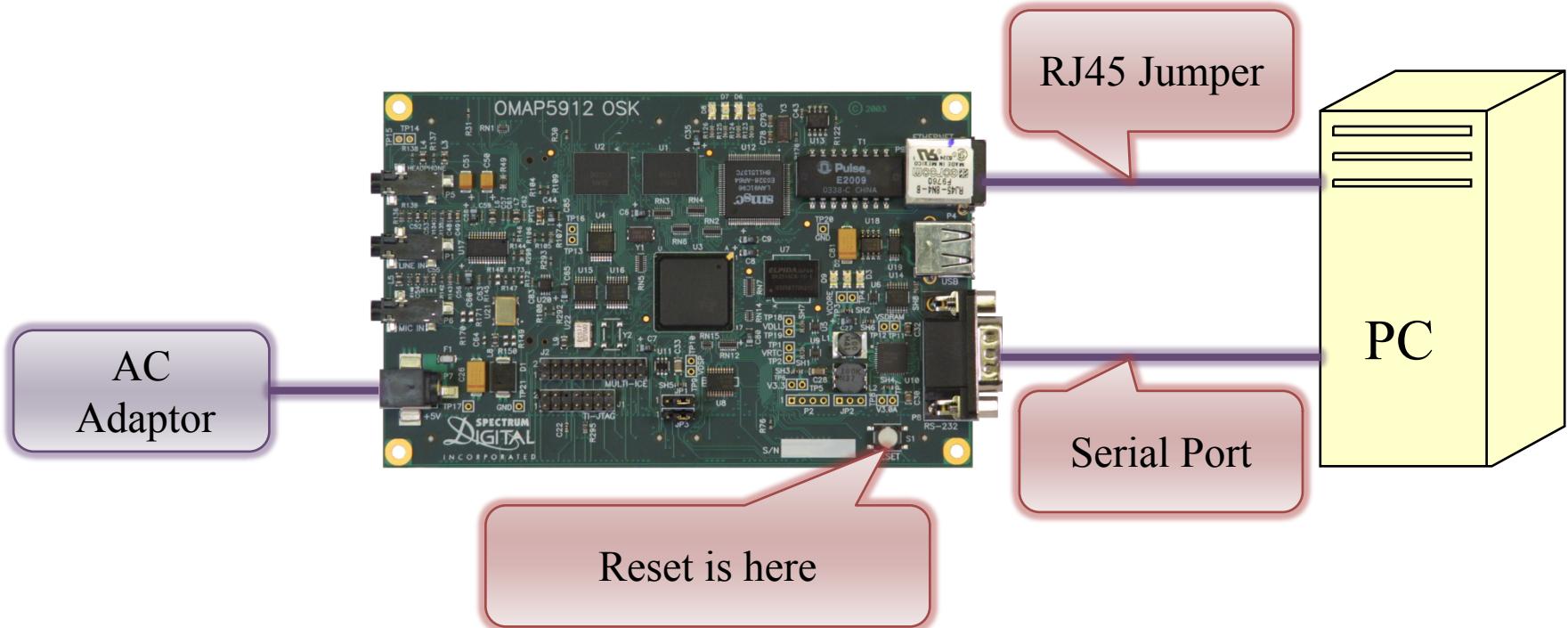
- ▶ Start and quit minicom
 - Start *minicom*
 - Quit *CTRL+A* → *Q*

```
Welcome to minicom 2.6.2
OPTIONS: I18n
Compiled on Aug  7 2013, 13:32:48.
Port /dev/ttyS0, 21:18:16

Press CTRL-A Z for help on special keys
```

Prepare for the Booting

- ▶ Copy the boot image for TFTP booting
 - `cp /opt/linux-2.6.12/arch/arm/boot/uImage /var/lib/tftpboot/uImage`
- ▶ Set the evaluation board as follows



Boot the Evaluation Board

- ▶ Start mimicom
 - *minicom*
- ▶ Press the reset button on the board
 - After the reset, immediately press any key on minicom terminal
 - You will get the following prompt

```
OMAP5912 OSK #
```

Download the New Kernel

- ▶ Set the boot configuration
 - *set ipaddr 192.168.68.yy* (evaluation board IP)
 - *set serverip 192.168.68.zz* (PC IP)
 - *set netmask 255.255.255.0*
 - *set gatewayip 192.168.68.254*
 - *set ethaddr 00-0e-99-xx-xx-xx*
 - *set bootargs console=ttyS0,115200n8 rw ip=192.168.68.yy root=/dev/nfs nfsroot=192.168.68.zz:/tmp/rootfs2.6,v3*
 - *printenv* → double check the setting

```
OMAP5912 OSK # printenv
bootdelay=3
baudrate=115200
bootfile="uImage"
bootcmd=bootm 0x100000
ipaddr=192.168.68.123
serverip=192.168.68.186
netmask=255.255.255.0
gatewayip=192.168.68.254
ethaddr=00-0e-99-02-0d-0b
stdin=serial
stdout=serial
stderr=serial
bootargs=console=ttyS0,115200n8 rw ip=192.168.68.123 root=/dev/nfs nfsroot=192.168.68.186:/tmp/rootfs2.6,v3

Environment size: 337/131068 bytes
OMAP5912 OSK #
```

- *saveenv* → if everything is correct → be careful, do not crash the entire system

Boot the New Kernel and Mount the NFS Root Filesystem

- ▶ Download the kernel: `tftpboot 0x10000000 ulimage`

- ▶ Boot the OS: *bootm 0x10000000*

```
Looking up port of RPC 100003/3 on 192.168.68.186
Looking up port of RPC 100005/3 on 192.168.68.186
VFS: Mounted root (nfs filesystem).
Freeing init memory: 112K
init started: BusyBox v1.00-pre8 (2004.03.05-22:18+0000) multi-call binary

*****
Starting System Init for OMAP59120SK
*****
```



Check Point 3

Done!

Or Bugs!?

Common Mistakes

- ▶ *su* and *export* should be used whenever a new terminal is created
 - If you extract the root file system by the user csie, there will be an error when you boot the board to mount the NFS root file system
 - Reboot the computer and do everything again
 - If you do not export the path of the tools, you will get some error when you compile the kernel module
- ▶ Please read the error message if you type something wrong
- ▶ UART: it should be connected to the bottom port
- ▶ Ethernet: do check the IP is correct
- ▶ Some evaluation boards were tested to be good: 1, 5, 7, 10, 11, 12, 15, 19, 20



Pthread Programming on TI OMAP 5912

Cross Compile a Program

- ▶ Copy the file in cp4 to /tmp/rootfs2.6
- ▶ Make it: *make*
- ▶ The Makefile is like:

```
PREFIX=/opt/usr/local/arm/3.3.2
```

```
CC=$(PREFIX)/bin/arm-linux-gcc
```

```
CFLAGS= -I$(PREFIX)/include -L$(PREFIX)/lib
```

```
all:
```

```
    $(CC) -o cp4.out mutex_thd.c -static -lpthread $(CFLAGS)
```

- ▶ You now can *./cp4.out* on the OMAP evaluation board to execute the program on the board

Check Point 4

- ▶ Please read the source code and the make file
- ▶ The example code is executed on the board
- ▶ Does it execute as you expect?

Exercise for Thread Creation and Join

- ▶ Copy the director cp5 to /tmp/rootfs2.6
- ▶ Now, you are a system programmer to improve the performance of a single-thread program
 - The main program is main_single_thread.c
 - The outsourcing program is in an object file format:
 - functionsARM.o is for ARM processor
 - functionsX86.o is for X86 processor
 - myFunctions.h is the header file
 - You have to write the Makefile and modify the program into a multi-thread version
- ▶ Hints
 - \$(CC) -o cp5.out objFile.o cFile.c -static -lpthread \$(CFLAGS)
 - pthread_create(&thread[7],NULL,(void *)&function7,NULL);
 - pthread_join(thread[i],(void **)&results[i]);

Check Point 5

- ▶ Do you understand the meaning of thread creation and join?
- ▶ Does the program executes as you expect
- ▶ Measure execution time of the single-thread version and multi-thread version programs
- ▶ Is there any difference?

Exercise for Mutex

- ▶ Now copy the director final to /tmp/rootfs2.6
- ▶ Write the Makefile
- ▶ Change the program to use multiple threads
- ▶ Note that some functions use the same data
- ▶ Mutex should be used to protect the shared data
- ▶ Hints:
 - Create multiple threads → join them → use mutex to protect shared data
 - Minimized the code protected by mutex

Final Check Point

- ▶ Measure the time for the program
- ▶ Does the program works well
- ▶ Is it efficient?

Grading this Exercise

- ▶ Check point 1: 10%
- ▶ Check point 2: 10%
- ▶ Check point 3: 10%
- ▶ Check point 4: 10%
- ▶ Check point 5: 10%
- ▶ Final check point: 10%
- ▶ Report before the exercise: 20%
 - Two page A4, 12 pt font
 - Deadline is 10:00 2019/12/9
 - File name: EOS-Lab1-Study-Student_ID
 - File type: PDF or Word
 - Send it to my email: chewei@mail.cgu.edu.tw
 - Email title: EOS Lab1 Study Student_ID
- ▶ Report after the exercise: 20%
 - Bonus: test multiple thread programs on your PC: 20%
 - Two page A4, 12 pt font
 - Deadline is 10:00 2018/12/16
 - File name: EOS-Lab1-Report-Student_ID
 - File type: PDF or Word
 - Send it to my email: chewei@mail.cgu.edu.tw
 - Email title: EOS Lab1 Report Student_ID