

**ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



**BÁO CÁO
THỰC TẬP NGOÀI TRƯỜNG
HỌC KỲ 223 NĂM HỌC 2022-2023**

- **NGÀNH: Kỹ Thuật Máy Tính**
- **CHƯƠNG TRÌNH ĐÀO TẠO: OISP**
- **ĐƠN VỊ/ DOANH NGHIỆP NHẬN THỰC TẬP:**
Big Dolphin Co., Ltd.
- **CÁN BỘ HƯỚNG DẪN**
Thầy Lê Trung Khanh
- **SINH VIÊN THỰC HIỆN (SVTH)**
Lê Tuấn Hưng MSSV: 2052508

TP. HỒ CHÍ MINH, THÁNG 4, NĂM 2023

Contents

Week 1: Linux and GCC	3
1. Operating System	3
a. Definition of OS:.....	3
b. Structure of OS:	3
2. Linux.....	4
a. History of Linux:.....	4
b. Linux Directory Structure:.....	4
c. Linux File system:.....	5
d. Properties of common linux filesystem:	5
e. Linux permission for files and directories.	6
f. Linux basic command.	6
3. Open source license.....	7
Week 2: Makefile and bootloader	8
1. Makefile.....	8
2. C/C++ wrapper.....	9
3. Bootloader of OS:.....	11
Week 3: Advanced Makefile and pthread programming	13
1. String Substitution:.....	13
2. Pthread programming	14
3. Memory sharing between threads.....	17
Week 4-5-6: Build Linux:.....	18
1. Requirements.....	18
2. Configuration and build.....	19
3. Install the OS	20
4. Ch2root.sh	21
Week 7-8: Build OS for Rockchip RK3588.....	23
1. Export Configuration.....	23
2. Yocto SDK	25
Reference	26

Week 1: Linux and GCC

1. Operating System

a. Definition of OS:

The program that, after being initially loaded at computer by boot program, will manages all the other application programs in the computer.

Act intermediary between a user and computer hardware.

Using to allocate resource and control program.

The main different of OS from firmware is that OS will control the CPU to do tasks when CPU free.

b. Structure of OS:

There are various way to structure: Simple, Monolithic, Layered, Micro-Kernel.

Simple Structure:

- Include: Application program, System program, MS – DOS device driver, and ROM BIOS device driver.

Monolithic (UNIX):

- Include: System program and **Kernel** (above hardware, below system-call interface, which provides file system, CPU scheduling, memory management, etc.)

Layer:

- The OS is devide into a number of layers.
- Level 0: hardware, level N: user interface, Only higher level can access to the lower level. => security.

Microkernel:

- Bring non-essential components to the user interface.
- Each Micro-Kernels are inpendently isolated from the others.

2. Linux

a. History of Linux:

Linux is a free and open-source operating system that was created by Linus Torvalds in 1991, which base on the UNIX operating system.

List of popular Linux distribution: Ubuntu, Fedora, Debian, Kali linux, CentOS, etc.

b. Linux Directory Structure:

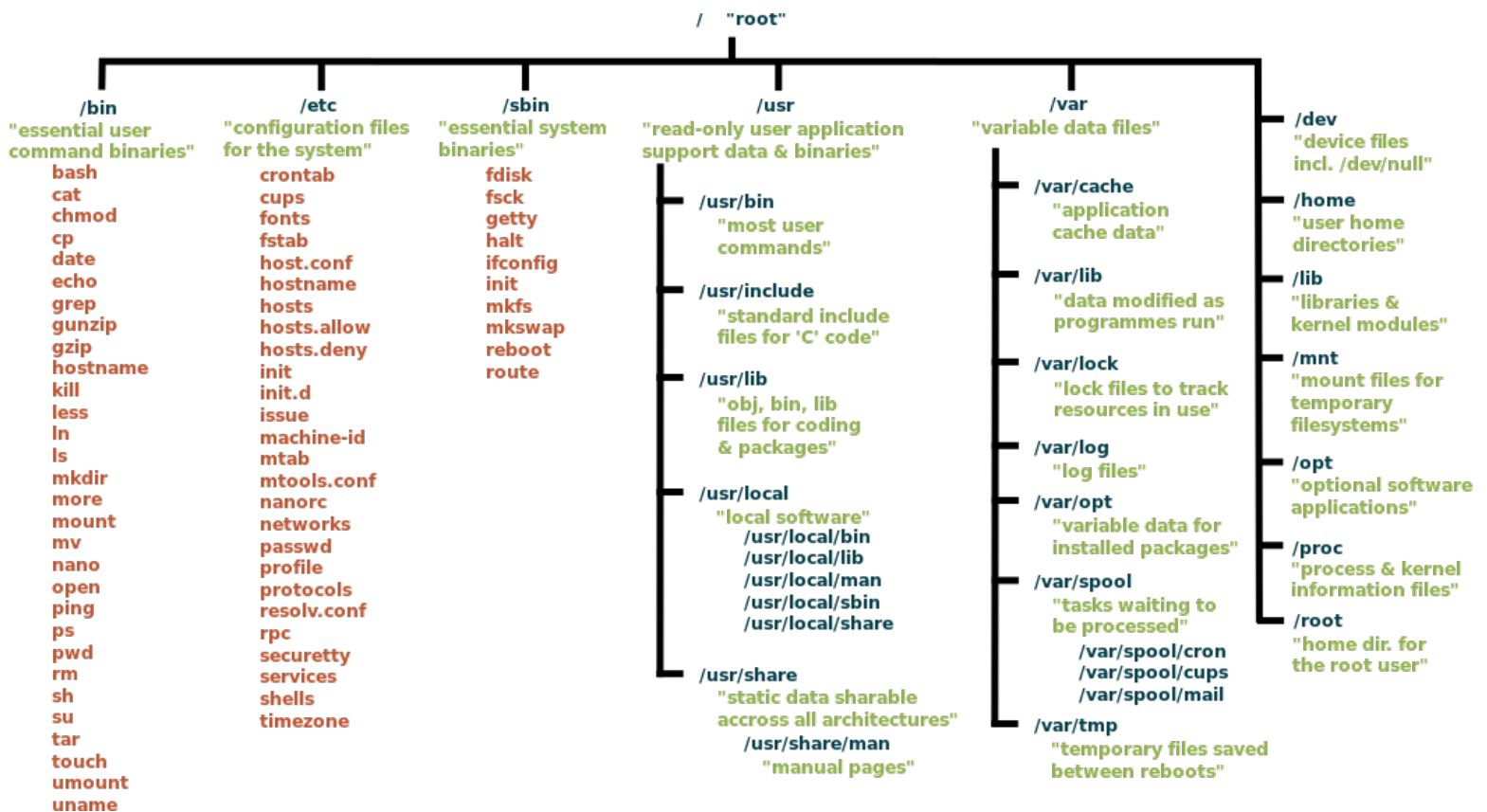


Fig 1. Linux file directory

Linux use the '/' to split the directory, the root folder start from '/', and use the Filesystem Hierarchy Standard (FHS) is a reference describing the conventions used for the layout of a UNIX system.

c. Linux File system:

- The method and data structure that the operating system uses to control how data is stored and retrieved.

Folder	Function
/bin	Basic programs
/boot	Contains the Linux kernel to boot and system maps and second-stage boot files.
/dev	Contains device files (CDRom, HDD, FDD....).
/etc	Contains system configuration files.
/home	Directory for users other than root.
/lib	Contains shared libraries for commands located in /bin and /sbin. And this directory also contains kernel modules.
/mnt hoặc /media	Mount point is the default for external file systems.
/opt	Directory containing additional software installations.
/sbin	System programs
/srv	Data used by servers on the system.
/tmp	Directory containing temporary files.
/usr	Directory containing permanent or important files to serve all users.
/var	Variable data handled by the daemon. This includes log files, queues, caches, caches, etc.
/root	Administrator personal files (root)
/proc	Used for the Linux kernel. They are used by the kernel to output data to user space.

d. Properties of common linux filesystem:

ext3: have an journaling (error-detection), can support from 2TiB to 32TiB.

ext4: can help defragmentation, and large size volume, provide fast transferring.

btrfs: better file system, Btrfs is a copy-on-write (COW) filesystem that focuses mainly on ease in repair and administration, have data integrity, remove duplicate data and handle more size. *performance

e. Linux permission for files and directories.

3 type of permission: READ (r), WRITE (w), EXECUTE (x)

Users are Owner, Group, Public.

- `-` - Regular file.
- `b` - Block special file.
- `c` - Character special file.
- `d` - Directory.
- `l` - Symbolic link.
- `n` - Network file.
- `p` - FIFO.
- `s` - Socket.

f. Linux basic command.

pwd: show current directory.

cd: navigate folder

ls: see the information of the folder.

cat: create file, concatenate 2 file.

cp: copy file to another folder

rmdir: delete empty directory

rm: remove

touch: create a file

grep: find the “word” inside the file.

sudo: super user do

df: disk space used

du: disk usage

head, tail: see first, last line of the file.

chmod: change the permission of the directory

chown: change owner of directory

kill: kill the PID

wget: download from internet from URL.

echo: add data to the file.

3. Open source license

Open-source licenses are categorized as copyleft or permissive.

4 popular OSL: GPL, Apache, MIT, BSD

- MIT License:

A permissive free and open-source software license. It allows users to use, copy, modify, merge, publish, distribute, sublicense, and sell copies of the software, either in source code or binary form, without restriction. The only real requirement of the MIT License is that users must include a copy of the license in any distribution of the software.

- Apache License:

A permissive free and open-source software license, one of the key features of the Apache License is its patent grant, which gives users a license to any patents that may be associated with the software.

- BSD License:

A permissive free and open-source software license, the key point that can be used in proprietary software, meaning that users are free to incorporate the software into proprietary products without releasing the source code.

- GPL License:

A copyleft license, which means that any derivative work or modification of GPL-licensed software must also be released under the GPL license and the requirement that source code need to be available.

Week 2: Makefile and bootloader

1. Makefile

Makefile is a file that it defines a set of tasks to be executed .

There are 2 parts: target and recipe.

- Target: might be a binary file that depends on prerequisites (source files). On the other hand, a prerequisite can also be a target that depends on other dependencies. ***target will not run if there exists the file with the same name***
- Recipe: a command to be run.

“.PHONY” make the target always run.

“.DEFAULT_GOAL :=” will define the default target, which mean that when we use “make” only, that target will be executed.

We can use variable ‘\$(name)’ to *simplify the* makefile file.

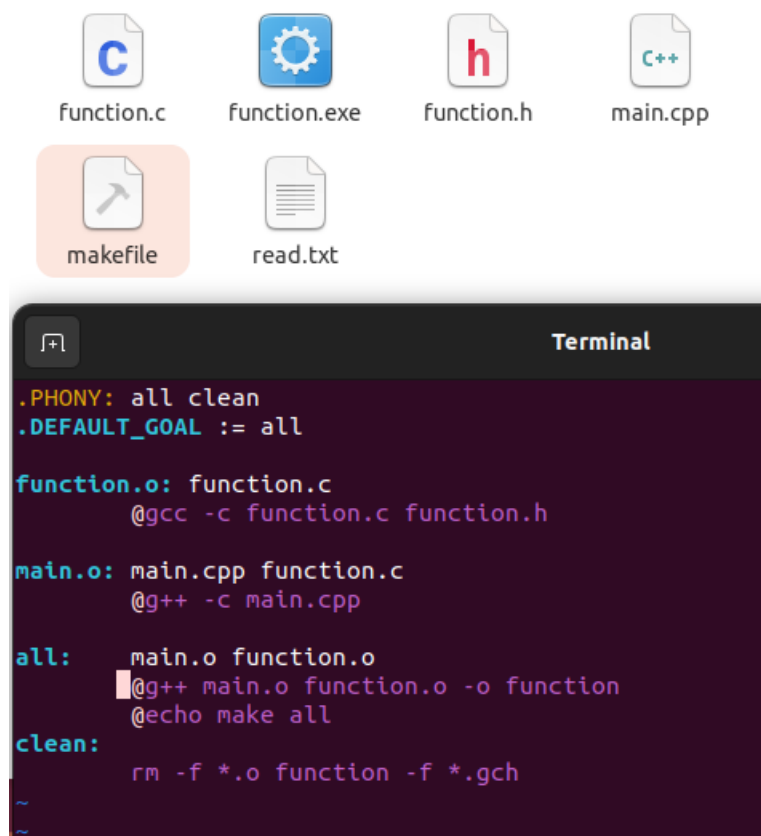


Fig 2. Example of a makefile file

2. C/C++ wrapper

C compiled process:

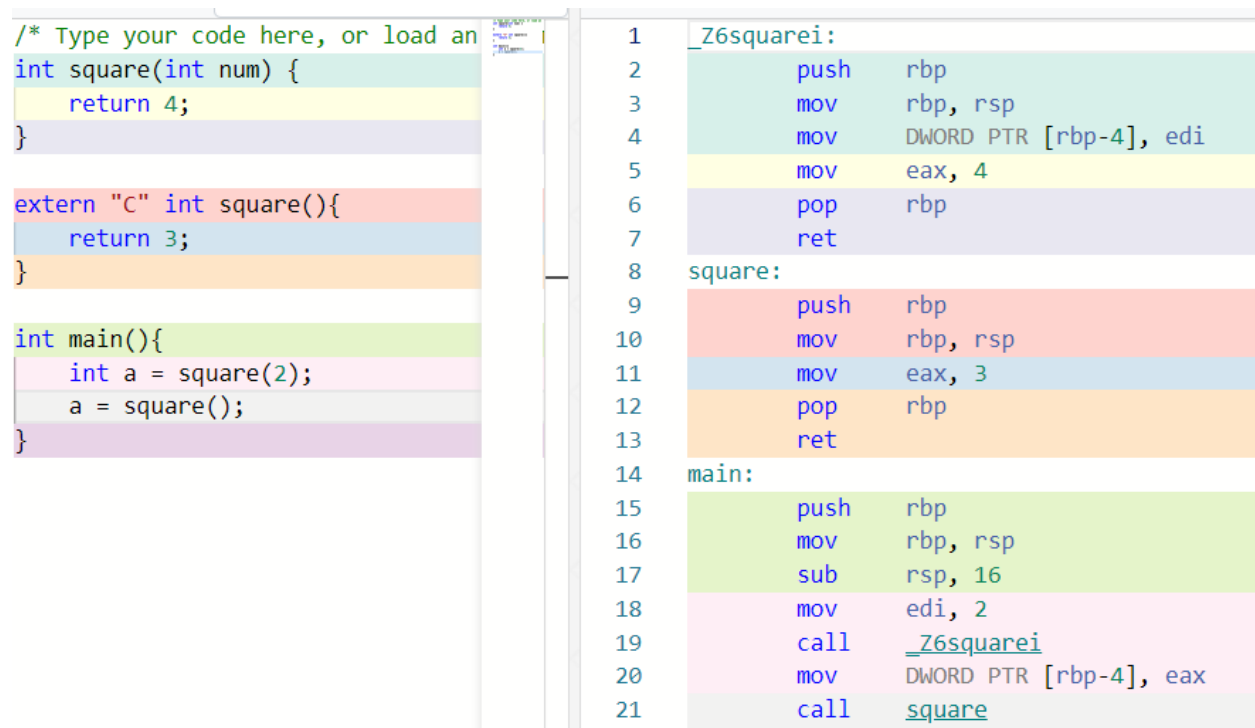
- Preprocessed source file:
 - o gcc -E [-o hello.i] hello.c
- Assembly code:
 - o gcc -S [-o hello.S] hello.c
- Binary file
 - o gcc -c [-o hello] hello.c
- Executable file
 - o gcc [-o hello] hello.c

***Extra:** How to compile C/C++ in the source code with g++.

First, we need to understand the mangle.

In C++, there is a name mangling, which mean that the function name are decoded
=> C++ support overloading function.

But in C, the funtion name are kept the same. So to combine C and C++, we will use the extern "C" to notice that these function do not need to be mangled.



```
/* Type your code here, or load an existing file */
int square(int num) {
    return 4;
}

extern "C" int square(){
    return 3;
}

int main(){
    int a = square(2);
    a = square();
}
```

```
1  _Z6squarei:
2      push    rbp
3      mov     rbp, rsp
4      mov     DWORD PTR [rbp-4], edi
5      mov     eax, 4
6      pop     rbp
7      ret
8  square:
9      push    rbp
10     mov     rbp, rsp
11     mov     eax, 3
12     pop     rbp
13     ret
14  main:
15     push    rbp
16     mov     rbp, rsp
17     sub     rsp, 16
18     mov     edi, 2
19     call    _Z6squarei
20     mov     DWORD PTR [rbp-4], eax
21     call    square
```

Fig 3. Function mangled in C++

Then, to compile the C/C++ we need to use extern “C” both when calling:

- C from C++: tell g++ to expect unmangled symbols produced by gcc
- C++ from C: tell g++ to generate unmangled symbols for gcc to use

The example code should be:

```
#ifdef __cplusplus
extern "C" {
#endif
    int square();
#ifdef __cplusplus
}
#endif
```

3. Bootloader of OS:

Bootloader is a program that loads an operating system when a computer is turned on.

In linux, there are 6 distinct stages in typical booting process.

- 1: POST and BIOS (Basic Input/Output System).
POST do the integrity check (hardware check)
BIOS stored on ROM will search the MBR (independent with OS).
- 2: MBR (Master Boot Record) locate and load the bootloader into RAM.
- 3: GRUB, LILO, LOADIN: Load the kernel into memory.
- 4 - 5: Kernel initialization:
- 6: Run level programs

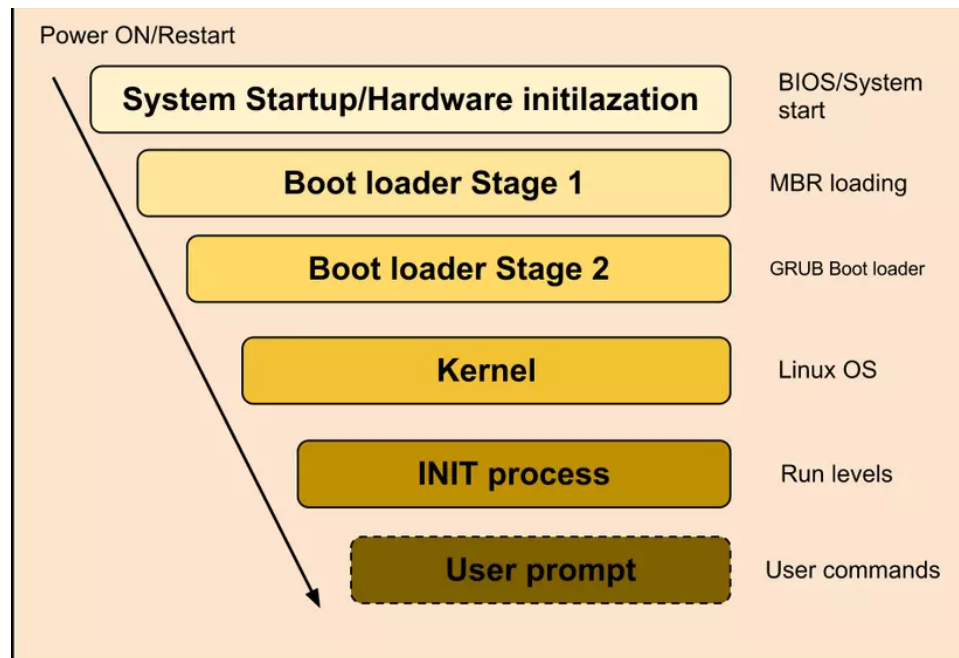


Fig 4. Bootloader progress

Embedded Linux: Usually use u-boot, the process is a bit same with above.

- 1: BootRom: the program stored on ROM, to initialize the system at hardware level.
- 2: SPL: same with MBR, locate and load bootloader into RAM.
- 3: uboot: load kernel into RAM

4: Linux kernel: The remain part of initialize the OS.

Compare between GRUB and LILO:

Features	GRUB	LILO
Definition	A boot loader offered by the GNU project is called GRUB.	It is a Linux bootloader that replaced loadlin as the default boot loader for most Linux OS in the years following its success.
Introduction	It was introduced in 1995.	Werner Almesberger was the first to introduce the LILO from 1992 to 1997.
Supporting OS	It supports multiple OS, including Windows, macOS, Linux, Unix, BSD, and Solaris.	It supports only a single operating system which is Linux OS.
Complexity	It is more complex than LILO.	It is simple and easy to use.
GUI Menu Choice	It includes a GUI menu choice.	It doesn't include a GUI menu choice.
Development	It is developed by GNU Project.	Werner Almesberger, John Coffman and Joachim Wiedorn are three developers that developed LILO.
Type	It is a new default boot loader.	It is an old default boot loader.
Network Booting	It supports network booting.	It doesn't support network booting.
Interactive Command Interface	It supports an interactive command interface.	It doesn't support an interactive command interface.

Fig 5. GRUB and LILO comparison

U-boot can be first and second stage of booting.

Week 3: Advanced Makefile and pthread programming

1. String Substitution:

\$(subst from, to, text): replace a part of text.

\$(patsubst pattern, replacement, text):

Ex:

\$(patsubst %.c, %.o, abc.c ok.c) => abc.o ok.o

Which mean that, for the 3rd argument, anything have suffix .c will create an file with suffix.o.

Or we can use \$(objects:.c=.o)

Then, when we want to create an object for each .c or .cpp file. We can write like:

SOURCES := \$(wildcard *.c)

OBJECTS := \$(patsubst %.c, %.o, \$(SOURCES))

```
.PHONY: all clean
.DEFAULT_GOAL := all

SRC := src
OBJ := obj

SOURCES := $(wildcard $(SRC)/*.c $(SRC)/*.cpp)
OBJECTS = $(patsubst $(SRC)/%.c, $(OBJ)/%.o, $(SOURCES))
OBJECTS := $(patsubst $(SRC)/%.cpp, $(OBJ)/%.o, $(OBJECTS))

CC := gcc
CXX := g++
CFLAGS := -Wall -Werror
CXXFLAGS := $(CFLAGS)

TARGET := function

all: $(TARGET)

$(TARGET): $(OBJECTS)
    @$(CXX) $^ -o $@
    @rm -f *.o *.gch

$(OBJ)/%.o: $(SRC)/%.cpp
    @$(CXX) -I$(SRC) -c $< -o $@

$(OBJ)/%.o: $(SRC)/%.c
    @$(CC) -I$(SRC) -c $< -o $@

clean:
    rm -f $(OBJECTS) $(TARGET) -f *.gch
```

Fig 6. Makefile file with wildcard

There are something to explain:

First `% .o : %cpp`, and `% .o : %c`. which means, for each prerequisite with suffix `.c` and `.cpp`, has a corresponding target with `.o`. `$<` and `$@` is the symbol that it will automatic match the `gcc` and `g++` command.

2. Pthread programming

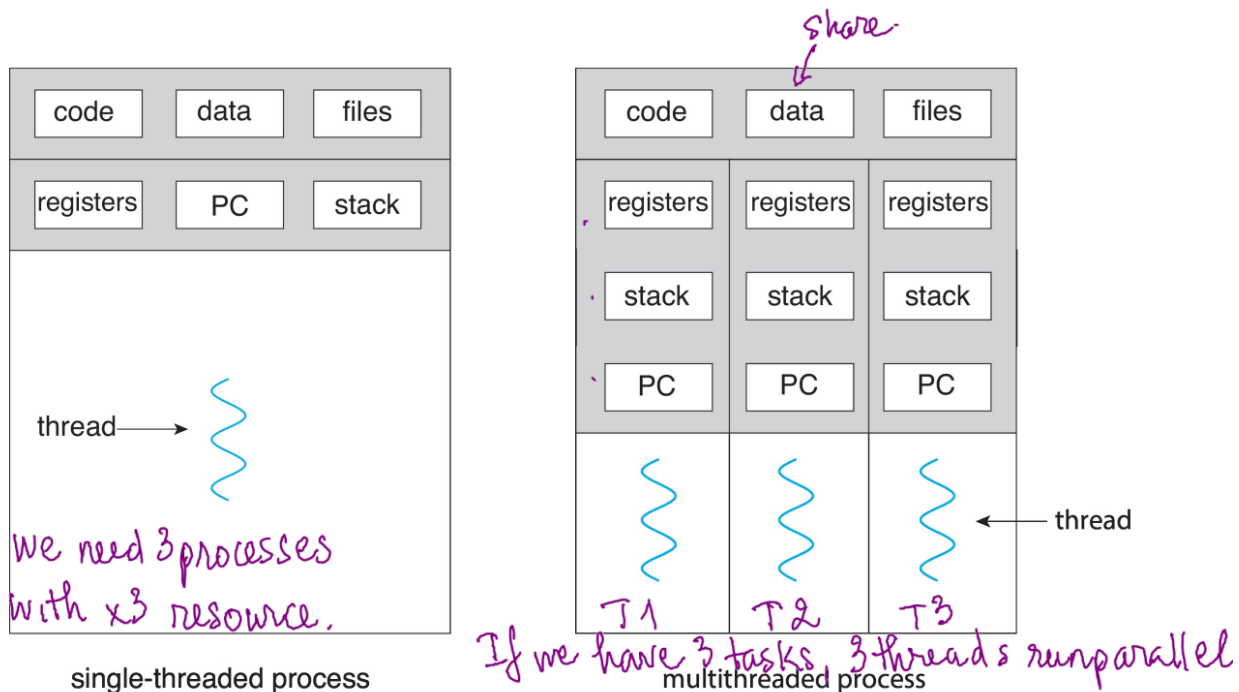


Fig 7. Single-Threaded vs Multithreaded

We can separate the process into multithread, that the resources are shared between threads with many benefits.

- Responsiveness: the execution may continue if a part of process are blocked (user interface)
- Resource sharing: threads share resources of process, then don't need share memory
- Economy: we need less resources than create a new process.

To create a thread:

```
int pthread_create(pthread_t * thread,  
                  const pthread_attr_t * attr,  
                  void * (*start_routine)(void *),  
                  void *arg);
```

- thread - returns the thread id. (unsigned long int defined in bits/pthreadtypes.h)
- attr - Set to NULL if default thread attributes are used.
- void * (*start_routine) - pointer to the function to be threaded. Function has a single argument: pointer to void.
- *arg - pointer to argument of function. To pass multiple arguments, send a pointer to a structure.

```
/*Ham thuc thi luong 1*/  
void *func_thread_1(void *ptr){  
    int dem_1 = 0;  
    while(1){  
        printf("[Luong 1] Dem = %d\n",dem_1);  
        dem_1++;  
        sleep(0.5); //delay 2 giay  
    }  
}  
  
/*Ham thuc thi luong 2*/  
void *func_thread_2(void *ptr){  
    int dem_2 = 0;  
    while(1){  
        printf("[Luong 2] Dem = %d\n",dem_2);  
        dem_2++;  
        sleep(1); //delay 2 giay  
    }  
}
```

Fig 8. Example of creating multithread

```
lehung@lehung-VirtualBox:~/shared/pthread$ ./multi  
[Luong 2] Dem = 1  
[Luong 1] Dem = 1  
[Luong 2] Dem = 2  
[Luong 1] Dem = 2  
[Luong 2] Dem = 3  
[Luong 2] Dem = 4  
[Luong 1] Dem = 3  
[Luong 2] Dem = 5  
[Luong 1] Dem = 4  
[Luong 1] Dem = 5  
[Luong 1] Dem = 6  
[Luong 1] Dem = 7  
[Luong 1] Dem = 8
```

Fig 9. The result of 2 threads

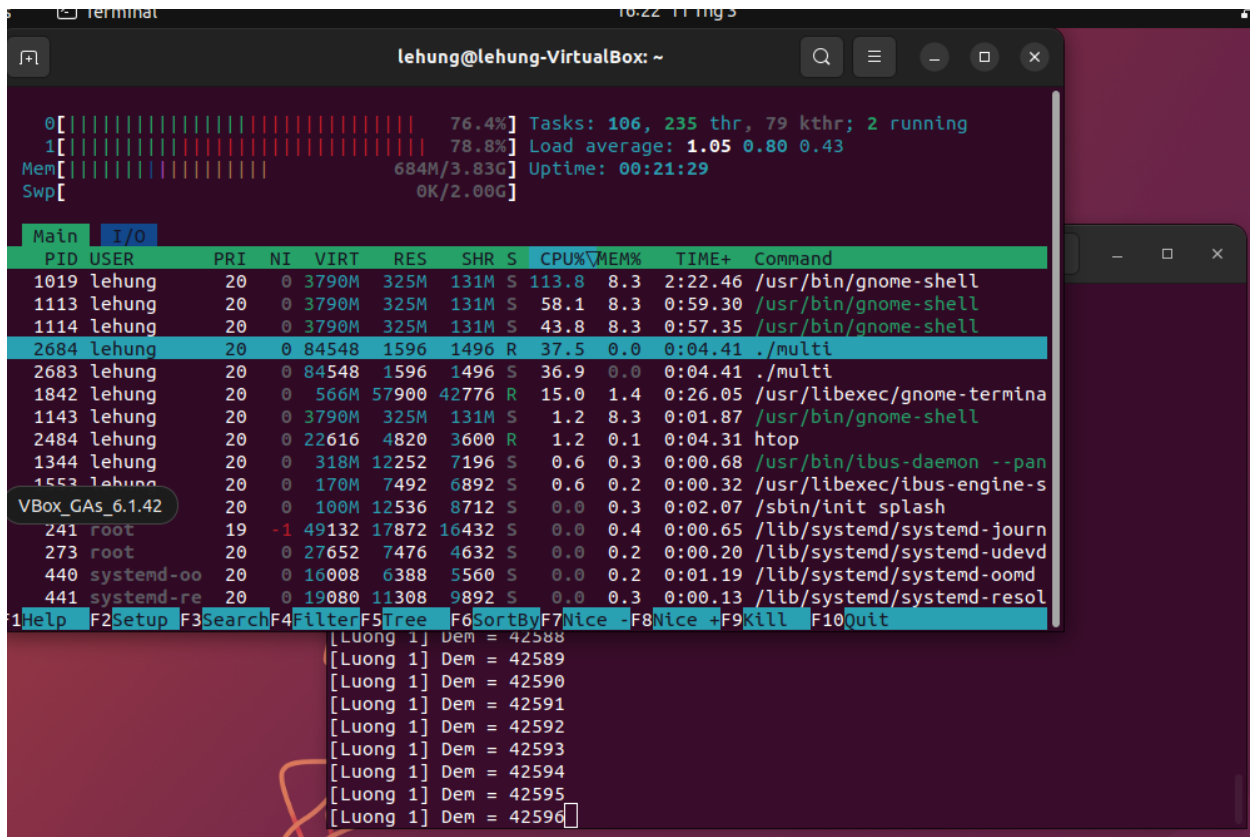


Fig 10. CPU utilization when the delay is set at high value

3. Memory sharing between threads

Data race: occurs when

- two or more tasks access in a shared memory, at least 1 access for writing and no exclusive lock.

Race condition: occurs when

- the error of execution that make unexpected output. (because the interruption).

`counter++;` could be implemented as

```
register1 = counter
register1 = register1 + 1
counter = register1
```

`counter--;` could be implemented as

```
register2 = counter
register2 = register2 - 1
counter = register2
```

Consider this execution *interleaving* with "`counter = 5`" initially:

S0: producer execute	<code>register1 = counter</code>	{register1 = 5}
S1: producer execute	<code>register1 = register1 + 1</code>	{register1 = 6}
S2: consumer execute	<code>register2 = counter</code>	{register2 = 5}
S3: consumer execute	<code>register2 = register2 - 1</code>	{register2 = 4}
S4: producer execute	<code>counter = register1</code> *	{counter = 6}
S5: consumer execute	<code>counter = register2</code>	{counter = 4}

Fig 11. Brief explanation of data race

***Critical Section:** a process may contains the changing in common variables, updating table, writing file, etc. **When one process in the critical section, no other may in the critical section.**

When I set 3 thread access into 1 global variable (shared memory), it may have the unexpected output.

```
[Luong 2] Dem = 4550
[Luong 2] Dem = 4551
[Luong 2] Dem = 4552
[Luong 2] Dem = 4553
[Luong 2] Dem = 4554
[Luong 1] Dem = 4554
[Luong main] Dem = 4554
[Luong 2] Dem = 4557
[Luong 2] Dem = 4558
[Luong 2] Dem = 4559
[Luong 2] Dem = 4560
```

We can use the mutex lock (and binary semaphore) to solve that problem.

Week 4-5-6: Build Linux:

1. Requirements

Step 1: Prepare packet for build.

```
sudo apt-get update && sudo apt-get dist-upgrade
```

#Install required package.

```
sudo apt-get install gawk wget git diffstat unzip texinfo gcc-  
multilib build-essential chrpath socat cpio python python3 python3-pip  
python3-pexpect xz-utils debianutils iputils-ping libssl1.2-dev xterm  
libyaml-dev libssl-dev python3-git zstd liblz4-tool
```

#Clone poky and Yocto SDK

```
git clone git://git.yoctoproject.org/poky  
git checkout -t origin/mickledore -b my-mickledore  
git pull  
cd poky  
git clone git://git.yoctoproject.org/poky.git poky-mickledore  
git clone git://git.openembedded.org/meta-openembedded  
git clone git://git.yoctoproject.org/meta-raspberrypi  
git clone https://github.com/meta-qt5/meta-qt5
```

#Start to pass the parameter to build

```
source oe-init-build-env
```

2. Configuration and build

Because the machine is raspberry pi 4, then we need to customize the configuration file.

```
BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
/home/lehung/var-fslc-yocto/sources/poky-mickledore/meta \
/home/lehung/var-fslc-yocto/sources/poky-mickledore/meta-poky \
/home/lehung/var-fslc-yocto/sources/poky-mickledore/meta-yocto-bsp \
/home/lehung/var-fslc-yocto/sources/poky-mickledore/meta-raspberrypi \
"
```

Fig 12. In conf/bblayer.conf

```
# This sets the default machine to be qemux86-64 if no other machine is selected
:
MACHINE ??= "qemux86-64"
MACHINE ??= "raspberrypi4-64"
# These are some of the more commonly used values. Looking at the files in the
```

Fig 13. In conf/local.conf

We also need to edit the size of rootfs to make sure it have enough size to install essential packages.

```
IMAGE_ROOTFS_SIZE = "2900000"
IMAGE_ROOTFS_EXTRA_SPACE = ""
IMAGE_OVERHEAD_FACTOR = ""
```

```
IMAGE_ROOTFS_SIZE = "2900000"
#
# Additional image features
"
```

Fig 14. Modifying the size of rootfs

*Note: by default, the overhead_factor is 1.3, then the size of rootfs is

$2,900,000 \times 1.3 = 3,770,000 \text{ KB} + x$ (the size of kernel, with about of 100-200MB).

Finally, we start to build the uboot and kernel

bitbake core-image-minimal

The process may take 5-12 hours, based on the internet connection. When we need to stop, please press CTRL+C **One time**, and wait until the task remain in queue finish, if the task is forced to stop, it may cause errors when continue.

3. Install the OS

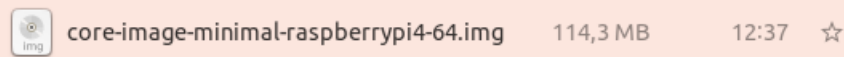
Then after finish, in `./tmp/deploy/images/raspberrypi4-64.wic.bz2`

We will unzip the `core-image-minimal-raspberrypi4-64.wic.bz2`

```
bzip2 -d -f core-image-minimal-raspberrypi4-64.wic.bz2
```

Then to write into SD card, we change the extension `.wic => .img` and write or

```
sudo dd bs="1024" if=core-image-minimal-raspberrypi4-64.wic of=/dev/sd'X'  
status=progress conv=fsync "
```



Finally, we go to `config.txt` in boot partition and comment all command.

A screenshot of a text editor window titled 'config.txt'. The editor has a menu bar with 'File', 'Edit', and 'View'. The content of the file is as follows:

```
## pwr_led_activelow  
## Set to "on" to invert the sense of the  
LED  
##  
## Not available on Model A/B boards.  
##  
## Default off.  
##  
#dtparam=pwr_led_activelow=off  
  
## pwr_led_gpio  
## Set which GPIO to use for the PWR LED  
##  
## In case you want to connect it to an  
external device  
##  
## Not available on Model A/B boards.  
##  
## Default 35.  
##  
#dtparam=pwr_led_gpio=35  
# Enable VC4 Graphics  
#dtoverlay=vc4-kms-v3d
```

The status bar at the bottom shows 'Ln 1186, Col', '100%', 'Unix (LF)', and 'UTF-8'.

4. Ch2root.sh

Ch2root.sh: The technical to sync the process with current linux of PC to install some essential packet to the new OS. In here, I use ubuntu 22.04.

```
##### UBUNTU 22.04 #####
#--1. Install QEMU
sudo apt install lib32ncurses6 qemu-user-static git pv libncurses5-dev -y

#--2. Download the Ubuntu root filesystem
mkdir ubuntu22
cd ubuntu22
wget -c https://cdimage.ubuntu.com/ubuntu-base/releases/22.04/release/ubuntu-base-22.04.1-base-armhf.tar.gz
sudo -s
mkdir rootfs
tar -xvf ubuntu-base-22.04.1-base-armhf.tar.gz -C rootfs

#--3. Copy qemu-user-static and firmwares
cp /usr/bin/qemu-arm-static rootfs/usr/bin/

#--4. Modify etc/apt/sources.list to un-comment all the repositories except the ones starting with deb-src
sed -i 's%^# deb %deb %' rootfs/etc/apt/sources.list

#--5. chroot
wget https://github.com/bigdolphins/chroot/raw/main/ch2root.sh
chmod a+x ch2root.sh
./ch2root.sh -m rootfs/

#--6. Update the repositories
chmod 1777 /tmp
# resolve the nameserver for new OS
echo "nameserver 8.8.8.8" | tee /etc/resolv.conf > /dev/null
apt update

#--7. Install minimal packages
# Extra packet for ubuntu 22.04 -----#
apt-get install linux-generic-hwe-22.04
#-----#
apt install -y language-pack-en-base sudo bash-completion dialog vim nano lsof udev lsb-base
htop psmisc locate rsyslog
apt install -y bison flex bc build-essential cmake automake autoconf cmake-curses-gui pkg-config yasm tmux git
apt install -y net-tools dkms r8168-dkms ethtool iputils-ping alsa-utils unzip net-tools
netbase ifupdown network-manager ntp usbutils ssh whois
apt install -y apt-utils subversion graphviz espeak i2c-tools evtest sox onboard device-tree-compiler alsamixer-gui

#--8. Exit chroot and unmount proc, sys, dev, dev/pts
exit
./ch2root.sh -u rootfs/

#--9. Copy rootfs to sd
# Please make sure that unmounted rootfs before rsync
rsync -aP rootfs/ img_mount/
sync

#--10. Edit password ###
cd img_mount/etc
sudo nano passwd
```

```
#--11 unmount ----  
umount img_mount/
```

```
GNU nano 4.8          passwd          Modified  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
sys:x:3:3:sys:/dev:/usr/sbin/nologin  
sync:x:4:65534:sync:/bin:/bin/sync  
games:x:5:60:games:/usr/games:/usr/sbin/nologin  
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin  
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin  
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin  
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin  
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin  
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin  
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin  
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin  
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin  
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin  
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin  
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin  
_apt:x:100:65534:./nonexistent:/usr/sbin/nologin  
syslog:x:101:102:./home/syslog:/usr/sbin/nologin  
[ Read 26 lines ]  
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  
^X Exit      ^R Read File ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line
```

Fig 15. In etc/passwd

Delete only 'x' in root => No password required in login

Week 7-8: Build OS for Rockchip RK3588

1. Export Configuration

Export OS config, build ubuntu for Rockchip RK3588.

To export the config of the kernel

```
cat /proc/config.gz | gunzip > running.config
```

Here is 2 reference for building an OS for rockchip RK3588, we use [Firefly_Linux_SDK](#)

```
mkdir ~/proj/rk3588_sdk/  
cd ~/proj/rk3588_sdk/
```

Method 1:

```
repo init --no-clone-bundle --repo-url https://gitlab.com/firefly-  
linux/git-repo.git -u https://gitlab.com/firefly-linux/manifests.git -b  
master -m rk3588_linux_release.xml
```

(in case of this doesn't work, the can change to method 2)

Download Firefly_Linux_SDK sub-volume compressed package: [Linux SDK](#)

```
md5sum rk3588_linux_release_20230114_v1.0.6c_0*
```

After confirming that it is correct, you can unzip:

```
mkdir -p ~/proj/rk3588_sdk  
cd ~/proj/rk3588_sdk  
cat path/to/rk3588_linux_release_20230114_v1.0.6c_0* | tar -xv  
# export data  
.repo/repo/repo sync -l  
.repo/repo/repo start firefly --all
```

In the `device/rockchip/rk3588/` directory, there are configuration files (xxxx.mk) for different board types, which are used to manage the compilation configuration of each project of the SDK. The relevant configuration introduction:

```
# Target arch
export RK_ARCH=arm64
# Uboot defconfig
export RK_UBOOT_DEFCONFIG=xxxx_defconfig
# Kernel defconfig
export RK_KERNEL_DEFCONFIG=xxxx_defconfig
# Kernel defconfig fragment
export RK_KERNEL_DEFCONFIG_FRAGMENT=xxxx.config
# Kernel dts
export RK_KERNEL_DTS=roc-rk3588s-pc.dts
# parameter for GPT table
export RK_PARAMETER=parameter-xxxx.txt
# rootfs image path
export RK_ROOTFS_IMG=ubuntu_rootfs/rootfs.img
```

The parameter.txt file contains the partition information of the firmware. Take parameter-ubuntu-fit.txt as an example:

path: `device/rockchip/rk3588/parameter-ubuntu-fit.txt`

```
FIRMWARE_VER: 1.0
MACHINE_MODEL: RK3588
MACHINE_ID: 007
MANUFACTURER: RK3588
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 0xffffffff
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
CMDLINE:
mtdparts=rk29xxnand:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc),0x00020000@0x0000
8000(boot:bootable),0x00040000@0x00028000(recovery),0x00010000@0x00068000(backup),0x00c00000@0
x00078000(rootfs),0x00040000@0x00c78000(oem),-@0x00cb8000(userdata:grow)
uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9
```

To set up the environment:

```
sudo apt-get install repo git ssh make gcc libssl-dev liblz4-tool expect g++ patchelf chrpath
gawk texinfo chrpath diffstat binfmt-support qemu-user-static live-build bison flex fakeroot
cmake gcc-multilib g++-multilib unzip device-tree-compiler ncurses-dev
```

Then type `./build.sh`

2. Yocto SDK

Inside the **Firefly_Linux_SDK**, Yocto is also included in

```
# Go to yocto SDK
cd build/conf

# Select the configuration file and find the configuration file corresponding to the board
ln -fs rk3588/(device_target).conf local.conf
source oe-init-build-env
bitbake core-image-minimal
```

Reference

[1. Overview — Firefly Wiki \(t-firefly.com\)](#)

[GNU make](#)

[Cùng nhau học Linux kernel | # **U-boot/Kernel Boot Process** | Facebook](#)

[u-boot – Trung Kien's Blog \(wordpress.com\)](#)

[Concept of Booting: What is Booting Process? Type of Booting & Examples \(toppr.com\)](#)

[7.3 Image Options | Building a Custom Linux Distribution | InformIT](#)

[lsmod](#)

[Ubuntu 20.04 e1000e - Reset adapter unexpectedly FIX \(github.com\)](#)

[How to install Ethernet driver on Ubuntu Server 20.04.4 LTS - Ask Ubuntu](#)

[1. Compile Linux firmware — Firefly Wiki \(t-firefly.com\)](#)