

# 嵌入式作業系統 Embedded Operating Systems

## LAB2

312512005 黃名諄

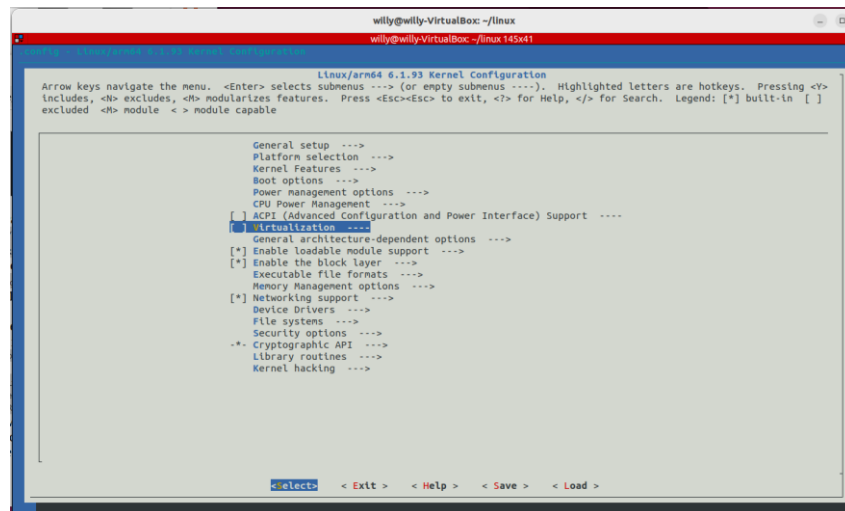
1. 移除不必要的功能, 縮小 RPi OS 的 image size (應說明: 目的、  
移除哪些功能、如何移除、前後 image size 與功能的比較)

嵌入式系統是為了執行特定功能所設計, 因此在需考量到實際在應用上所需要的**特定目的**去做為設計依據, 因此在做此 lab 縮減 image size 時, 我也是先思考了應用情境, 再根據此情境需求及目的去做縮減。我以目前實驗室研究做發想將應用設定為**自主式移動機器人平台上之嵌入式系統**, 自主式移動機器人簡單來說就是類似餐廳中送餐機器人的概念, 可自主導航並移動到目標位置, 其中整個系統需要特定裝置及算法來處理, 例如要處理定位、路徑規劃、避障、底盤控制等等演算法, 非常需要能及時且低延遲的運算, 且需要 sensors 來感知環境, 因此需求如下:

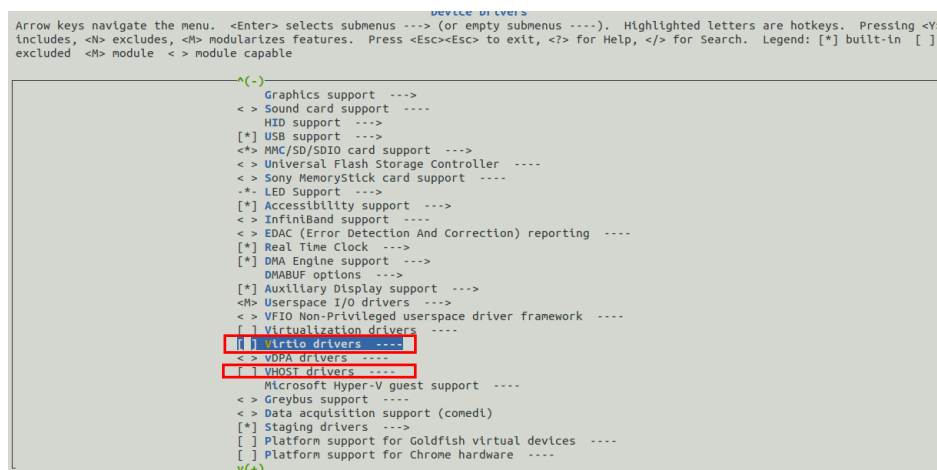
- 盡可能**提升運算速度**, 使各演算法能互相及時配合
- **降低延遲**, 使各系統能及時完成計算及互相回應
- 數據紀錄, **讀寫速度盡可能提高**
- Sensor 如光達、相機靠有線連接, 如 usb、ethernet, 其他不需要的裝置可移除
- **縮減 image size**, 使 memory 有更多空間能給 user program 使用

先透過 `make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-menuconfig` 編輯.config, 在先前 default config 之上進行刪減修改, 以下是我針對我的需求所做的改動:

- i. 關 **Virtualization** : 因為我的情境需求上沒有在 RPi 上建虛擬機需求, 可關閉虛擬化功能

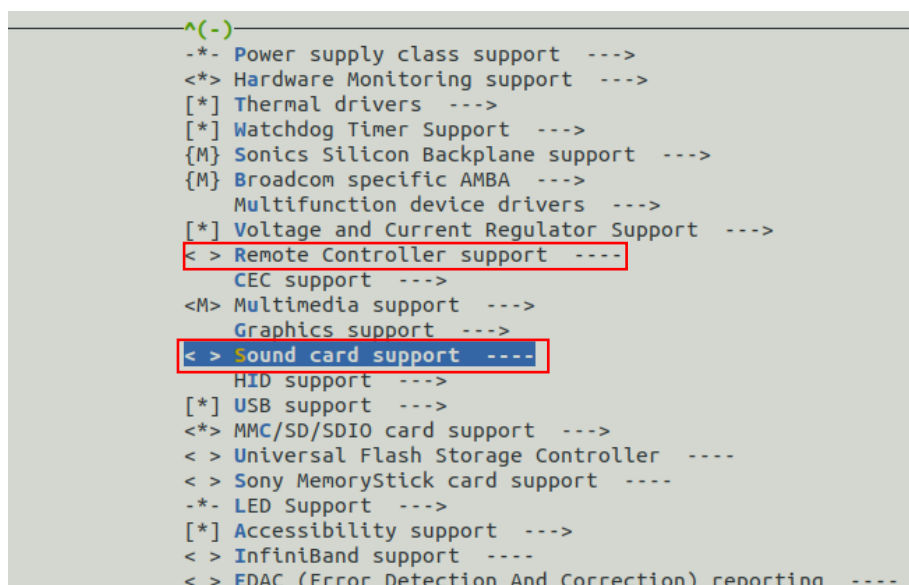


- ii. 在 device driver 中，也關閉虛擬化相關的選項：  
關 Virtio drivers 及 VHOST drivers



- iii. 在 device driver 中，關閉不必要的裝置驅動：

- 不需要音效，關閉 **Sound card support**
- 不會用到遙控設備，關閉 **Remote controller support**



iv. 在 Networking support 中

- sensor 都用有線連接，不需要藍芽，關閉 **Bluetooth subsystem support**
- 不會用到無線電設備，關閉 **Amateur Radio support**

```
--- Networking support
    Networking options --->
[*]  Amateur Radio support --->
<M> CAN bus subsystem support --->
< > Bluetooth subsystem support ----
< >  RxRPC session sockets
< >  KCM sockets
```

```
networking support
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <M> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
excluded <M> module < > module capable

--- Networking support
    Networking options --->
    [*]  Amateur Radio support --->
    <M> CAN bus subsystem support --->
    < > Bluetooth subsystem support ----
    < >  RxRPC session sockets
    < >  KCM sockets
    [ ] MCTP core protocol support ----
    -* Wireless --->
    <M> RF switch subsystem support --->
    <M> Plan 9 Resource Sharing Support (9P2000) --->
    < > CAIF support ---
    (M) Ceph core library
    [ ] Include file:line in ceph debug output
    [ ] Use in-kernel support for DNS lookup
    <M> NFC subsystem support --->
    < > Packet-sampling netlink channel ----
    < > Inter-PE based on IETF Forces InterFE LFB ----
    -* Network light weight tunnels
    [*] Execute BPF program as route nexthop action
    [ ] Page pool stats
    < > Generic fallback module
    [*] Netlink interface for ethtool
```

v. 在 CPU Power Management→CPU Frequency scaling 中， 將

**Default CPUFreq governor 從預設的 powersave 改為**

**performance**，讓 OS 以 performance 為優先讓 CPU 以較高

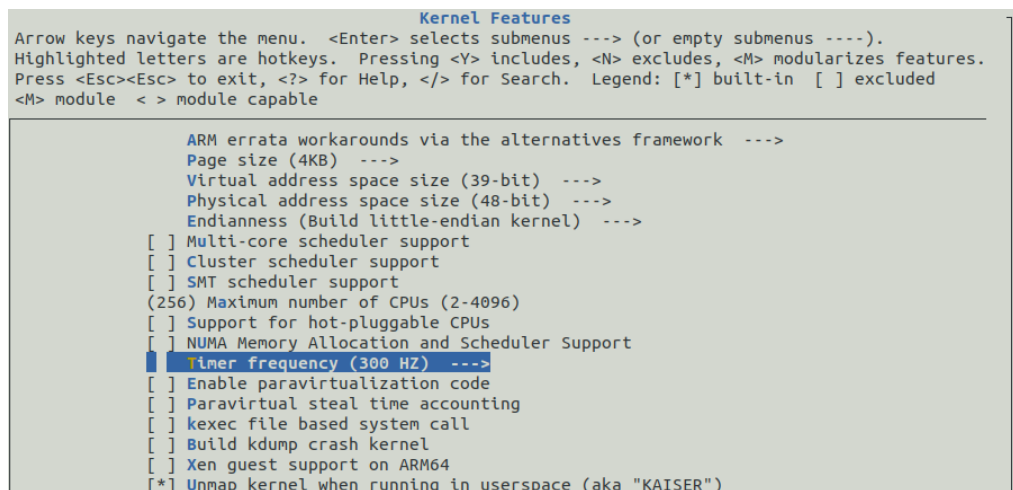
frequency 去執行以提升效能，我的應用情境相對於節省能源更要

求 CPU 能有高效能去快速計算導航演算法。

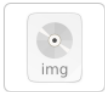

```
~ CPU Frequency scaling
CPU Frequency scaling
the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <M> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
excluded <M> module < > module capable

[*] CPU Frequency scaling
[*] CPU frequency transition statistics
Default CPUFreq governor (performance) --->
-* 'performance' governor
<*> 'powersave' governor
<*> 'userspace' governor for userspace frequency scaling
<*> 'ondemand' cpufreq policy governor
<*> 'conservative' cpufreq governor
[*] 'schedutil' cpufreq policy governor
*** CPU frequency scaling drivers ***
<*> Generic DT based cpufreq driver
<*> Broadcom STB AVS CPUFreq driver
<*> Raspberry Pi cpufreq support
```

- vi. 在 Kernel Features 中，將 **Timer frequency** 從 **250Hz** 改為 **300Hz**。此選項會影響 timer 頻率，我希望能透過適當提高 timer 頻率，提升 timer interrupt 精度使 CPU 能更有效率執行各項演算法工作，達到降低 **latency** 的目的。但其實有試過改為最大值 1000Hz，發現過高的 Timer frequency 反而使效能下降，原因我認為是過於頻繁的 timer interrupt 打斷 CPU 工作，反而會使其無法順暢工作，因此適度提高才是對系統效能提升最有效的。



- 更改完上述設置後儲存，再進行講義步驟 5~9 編譯及安裝 image 進 SD 卡  
刪減前後之 image size 大小比較：

kernel8.img Properties	kernel8-final-edit.img Properties
<div>BasicPermissionsOpen With</div> <div></div> <div>Namekernel8.img</div> <div>TypeRaw disk image (application/x-raw-disk-im...</div> <div>Size23.6 MB (2358,9376 bytes)</div> <div>Parent folder/media/willy/bootfs</div> <div>Accessed2024年九月20日 (週五) 08時00分00秒</div> <div>Modified2024年九月20日 (週五) 21時04分00秒</div> <div>Created2024年七月04日 (週四) 08時16分18秒</div>	<div>BasicPermissionsOpen With</div> <div></div> <div>Namekernel8-final-edit.img</div> <div>TypeRaw disk image (application/x-raw-disk-im...</div> <div>Size22.7 MB (2267,5968 bytes)</div> <div>Parent folder/media/willy/bootfs</div> <div>Accessed2024年十月02日 (週三) 08時00分00秒</div> <div>Modified2024年十月02日 (週三) 15時57分00秒</div> <div>Created2024年十月02日 (週三) 15時57分01秒</div>

縮減前(default config)

縮減後

可發現設置及移除部分功能後的 kernel image 大小確實變小了，使 kernel 占用更少的記憶體空間。

2. 在網路上下載可以評測 kernel 的套件，比較原本與更改後的 kernel (應說明：下載哪一套件、改善哪方面的效能與評測的結果)

- 在網路搜尋後，決定使用 **sysbench** 這套評測套件，其能簡單評測 CPU、memory、磁碟 file I/O 的效能
- 安裝 sysbench 流程：
  1. ssh 進 RPi，輸入 `sudo apt-get update` 更新資訊
  2. `sudo apt-get install -y sysbench` 安裝套件
  3. 輸入 `sysbench --version` 檢查版本及是否安裝成功

```
pi@raspberrypi:~ $ sysbench --version
sysbench 1.0.20
```

### 1. CPU 效能

指令: `sysbench cpu --threads=2 --cpu_max_prime=20000 run`

利用質數運算測試 CPU 效能，將最大值設為 20000 模擬工作量較高情況，且因為實際情況下較常用多核執行任務，因此設置使用 2 threads，目的是評測使用多線程跑的效能

```
pi@raspberrypi:~ $ sysbench cpu --threads=2 --cpu_max_prime=20000 run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time

Prime numbers limit: 20000
Initializing worker threads...

Threads started!

CPU speed:
events per second: 228.71

General statistics:
total time: 10.0080s
total number of events: 2291

Latency (ms):
min: 8.55
avg: 8.73
max: 12.60
95th percentile: 9.22
sum: 19993.73

Threads fairness:
events (avg/stddev): 1145.5000/1.50
execution time (avg/stddev): 9.9969/0.00
```

縮減前(default config)

```
pi@raspberrypi:~ $ sysbench cpu --threads=2 --cpu_max_prime=20000 run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time

Prime numbers limit: 20000
Initializing worker threads...

Threads started!

CPU speed:
events per second: 232.36

General statistics:
total time: 10.0052s
total number of events: 2327

Latency (ms):
min: 3.65
avg: 8.59
max: 12.41
95th percentile: 9.56
sum: 19989.79

Threads fairness:
events (avg/stddev): 1163.5000/8.50
execution time (avg/stddev): 9.9949/0.00
```

縮減後

根據我前面設定的情境需求，我著重的效能改善應該是 CPU 計算速度及 latency:

	縮減前	縮減後
CPU speed Events per second	228.71	<b>232.36</b>
Latency(ms) avg.	8.73	<b>8.59</b>
Latency(ms) max.	12.60	<b>12.41</b>
Latency(ms) min.	8.55	<b>3.65</b>

從評測結果可發現，刪減修改過後的 kernel 在 CPU speed 上有些許提升，符合我需要較高運算速度的目的，且另一方面我也希望 latency 越低越好，刪減修改過後的 kernel 確實也能有較低的平均 latency，而且最大值及最小值都較低。因此在 CPU 方面，雖然提升效果沒有非常顯著，但確實有提升到我需要的效能。

## 2. Memory 效能

指令: sysbench memory run

```
pi@raspberrypi:~ $ sysbench memory run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Running memory speed test with the following options:
block size: 1KiB
total size: 102400MiB
operation: write
scope: global

Initializing worker threads...

Threads started!

Total operations: 5462453 (545502.45 per second)
5334.43 MiB transferred (532.72 MiB/sec)

General statistics:
total time:          10.0003s
total number of events: 5462453

Latency (ms):
min:                0.00
avg:                0.00
max:                0.61
95th percentile:   0.00
sum:                5055.48

Threads fairness:
events (avg/stddev): 5462453.0000/0.00
execution time (avg/stddev): 5.0555/0.00
```

縮減前(default config)

```
pi@raspberrypi:~ $ sysbench memory run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Running memory speed test with the following options:
block size: 1KiB
total size: 102400MiB
operation: write
scope: global

Initializing worker threads...

Threads started!

Total operations: 5645476 (563963.17 per second)
5513.16 MiB transferred (550.75 MiB/sec)

General statistics:
total time:          10.0003s
total number of events: 5645476

Latency (ms):
min:                0.00
avg:                0.00
max:                0.38
95th percentile:   0.00
sum:                5058.28

Threads fairness:
events (avg/stddev): 5645476.0000/0.00
execution time (avg/stddev): 5.0583/0.00
```

縮減後

根據我前面設定的情境需求，我著重的效能改善應該是**傳輸速度**及  
**latency**:

	縮減前	縮減後
Transfer speed (MiB/s)	532.72	<b>550.75</b>
Latency(ms) sum	<b>5055.48</b>	5058.28
Latency(ms) max.	0.61	<b>0.38</b>

從評測結果看來，修改過後的 kernel 能讓 memory **傳輸速度上升**，而 latency 的平均和最小值都是 0.00 太小難以比較，因此改為比較最大值及總和，可發現修改過後的 kernel 能**有效降低較大 latency 的發生**，但**總和上卻較高**，對單一較大延遲的任務有改善。或許因為我沒有修改特別的 memory 設定，對降低總體 latency 較無效果，但刪減部分無用裝置驅動對 memory 的效能還是有部分提升效果，我認為是因為整體 image 縮減能提供更多空間給 user program 使用，能使其較少發生 page fault 需要去 disk 抓 page 的需求。

### 3. 磁碟 file I/O 的效能

指令:`sysbench fileio --file-total-size=5G prepare`:準備好 5G 的檔案做為評測使用

`sysbench fileio --file-total-size=5G --file-test-mode=rndrw run`

```
pi@raspberrypi:~$ sysbench fileio --file-total-size=5G --file-test-mode=rndrw run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time

Extra file open flags: (none)
128 files, 40MiB each
5GiB total file size
Block size 10KiB
Number of IO requests: 0
Read/write ratio for combined random IO test: 1.50
Periodic fsync enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Initializing worker threads...

Threads started!

File operations:
  reads/s:          285.65
  writes/s:         137.10
  fsyncs/s:         445.09

Throughput:
  read, MiB/s:      3.21
  written, MiB/s:    2.14

General statistics:
  total time:        10.0562s
  total number of events: 7802

Latency (ms):
  min:              0.01
  avg:              1.28
  max:              305.75
  95th percentile: 3.30
  sum:              9961.22

Threads fairness:
  events (avg/stddev): 7802.0000/0.00
  execution time (avg/stddev): 9.9612/0.00
```

縮減前(default config)

```
pi@raspberrypi:~$ sysbench fileio --file-total-size=5G --file-test-mode=rndrw run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time

Extra file open flags: (none)
128 files, 40MiB each
5GiB total file size
Block size 10KiB
Number of IO requests: 0
Read/write ratio for combined random IO test: 1.50
Periodic fsync enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Initializing worker threads...

Threads started!

File operations:
  reads/s:          239.77
  writes/s:         159.85
  fsyncs/s:         523.65

Throughput:
  read, MiB/s:      3.75
  written, MiB/s:    2.50

General statistics:
  total time:        10.0123s
  total number of events: 9125

Latency (ms):
  min:              0.00
  avg:              1.09
  max:              257.92
  95th percentile: 2.81
  sum:              9959.65

Threads fairness:
  events (avg/stddev): 9125.0000/0.00
  execution time (avg/stddev): 9.9596/0.00
```

縮減後



類似於 memory，根據我前面設定的情境需求，我著重的效能改善應該是讀寫速度及 latency:

	縮減前	縮減後
Throughput read(MiB/s)	3.21	<b>3.75</b>
Throughput written(MiB/s)	2.14	<b>2.50</b>
Latency(ms) avg.	1.28	<b>1.09</b>
Latency(ms) max.	305.75	<b>257.92</b>
Latency(ms) min.	0.01	<b>0.00</b>

從評測結果可看出，修改過後的 kernel 在磁碟 file I/O 方面效能都有所提升，不論是讀寫速度或是 latency 都較未修改的好，對我的機器人系統在做紀錄數據或讀取地圖等操作時都會有所幫助。

### 3. 使 kernel 支援 real-time 功能，可參照講義的作法，依照自己

kernel 的版本來 patch (應說明: 欲下載哪一版本的補丁、如何下載與下載的結果)

- 原本是依照講義方式，查看 kernel 版本並下載對應版本補丁

```
pi@raspberrypi:~ $ uname -r
6.1.93-v8+
```

```
willy@willy-VirtualBox:~/linux$ head Makefile -n 4
# SPDX-License-Identifier: GPL-2.0
VERSION = 6
PATCHLEVEL = 1
SUBLEVEL = 93
```

但安裝後發現，此 6.1 的補丁在我的 RPi 3 B+ 上似乎不相容，無法開機，因此便根據助教的 hint，從步驟 3: Get the Kernel Sources 中更改 branch 的版本，clone 較舊版本的 kernel，查資料後有人用版本 4.19 在 RPi 3 B+ 成功安裝，因此我也使用 4.19 的 kernel 做 RT-patch，patch 過程如下:



1. Clone 4.19 的 kernel sources 至另外的一個資料夾 linux\_rt

```
git clone --depth=1 --branch rpi-4.19.y
```

```
https://github.com/raspberrypi/linux linux_rt
```

2. 在 clone 下來的資料夾中查看 kernel 版本

```
cd linux_rt
```

```
head Makefile -n 4
```

```
willy@willy-VirtualBox:~/linux_rt$ head Makefile -n 4
# SPDX-License-Identifier: GPL-2.0
VERSION = 4
PATCHLEVEL = 19
SUBLEVEL = 127
```

版本為 4.19.127

3. 接講義 step4，用 default config 來 build kernel sources and device tree files

```
KERNEL=kernel8
```

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
```

```
bcm2711_defconfig
```

4. 到 <https://www.kernel.org/pub/linux/kernel/projects/rt/> 搜尋對應版本的 real time patch

<a href="#">4.11/</a>	17-Oct-2017 13:42	-
<a href="#">4.13/</a>	17-Nov-2017 17:03	-
<a href="#">4.14/</a>	19-Mar-2024 17:39	-
<a href="#">4.16/</a>	03-Aug-2018 07:39	-
<a href="#">4.18/</a>	29-Oct-2018 11:51	-
<a href="#">4.19/</a>	23-Sep-2024 07:30	-
<a href="#">4.4/</a>	04-Feb-2022 09:35	-
<a href="#">4.6/</a>	30-Sep-2016 21:37	-
<a href="#">4.8/</a>	23-Dec-2016 15:26	-

找到 4.19

## Index of /pub/linux/kernel/projects/rt/4.19/

<a href="#">../</a>		
<a href="#">incr/</a>	25-Nov-2020 22:03	-
<a href="#">older/</a>	23-Sep-2024 07:30	-
<a href="#">patch-4.19.322-rt138.patch.gz</a>	23-Sep-2024 07:30	199K
<a href="#">patch-4.19.322-rt138.patch.sign</a>	23-Sep-2024 07:30	228
<a href="#">patch-4.19.322-rt138.patch.xz</a>	23-Sep-2024 07:30	163K
<a href="#">patches-4.19.322-rt138.tar.gz</a>	23-Sep-2024 07:30	414K
<a href="#">patches-4.19.322-rt138.tar.sign</a>	23-Sep-2024 07:30	228
<a href="#">patches-4.19.322-rt138.tar.xz</a>	23-Sep-2024 07:30	292K
<a href="#">sha256sums.asc</a>	23-Sep-2024 07:35	1269

點 older 資料夾進去找 4.19.127 的最新 patch

<a href="#">patch-4.19.124-rt53.patch.xz</a>	21-May-2020 20:46	163K
<a href="#">patch-4.19.127-rt54.patch.gz</a>	08-Jun-2020 20:17	199K
<a href="#">patch-4.19.127-rt54.patch.sign</a>	08-Jun-2020 20:17	833
<a href="#">patch-4.19.127-rt54.patch.xz</a>	08-Jun-2020 20:17	163K
<a href="#">patch-4.19.127-rt55.patch.gz</a>	22-Jun-2020 17:50	199K
<a href="#">patch-4.19.127-rt55.patch.sign</a>	22-Jun-2020 17:50	833
<a href="#">patch-4.19.127-rt55.patch.xz</a>	22-Jun-2020 17:50	163K

5. 接著根據講義，下載補丁並解壓縮傳入 patch utility

```
Wget https://www.kernel.org/pub/linux/kernel/projects/rt/4.19/older/patch-4.19.127-rt55.patch.gz
```

```
gunzip patch-4.19.127-rt55.patch.gz
```

```
cat patch-4.19.127-rt55.patch | patch -p1
```

6. 接著重做講義 step5~9 編譯有補丁的 kernel

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- Image
```

```
modules dtbs
```

此時應該會跳出選項，要選擇最下面 fully preemption kernel(RT)

選好後繼續編譯

```
Preemption Model
> 1. No Forced Preemption (Server) (PREEMPT_NONE)
  2. Voluntary Kernel Preemption (Desktop) (PREEMPT_VOLUNTARY)
  3. Preemptible Kernel (Low-Latency Desktop) (PREEMPT__LL) (NEW)
  4. Preemptible Kernel (Basic RT) (PREEMPT_RT) (NEW)
  5. Fully Preemptible Kernel (RT) (PREEMPT_RT_FULL) (NEW)
choice[1-5?]: 5
```

之後就接續將編好的 kernel image 安裝入 SD 卡

7. 將 SD 卡放入 RPi，可順利開機了，用指令確認是否有安裝 real time patch 成功

```
uname -r
```

```
uname -a
```

```
pi@raspberrypi:~$ uname -r
4.19.127-rt55-v8+
pi@raspberrypi:~$ uname -a
Linux raspberrypi 4.19.127-rt55-v8+ #1 SMP PREEMPT RT Wed Oct 2 20:29:48 CST 202
4 aarch64 GNU/Linux
pi@raspberrypi:~$
```

可看出是 rt 版本，patch 成功!