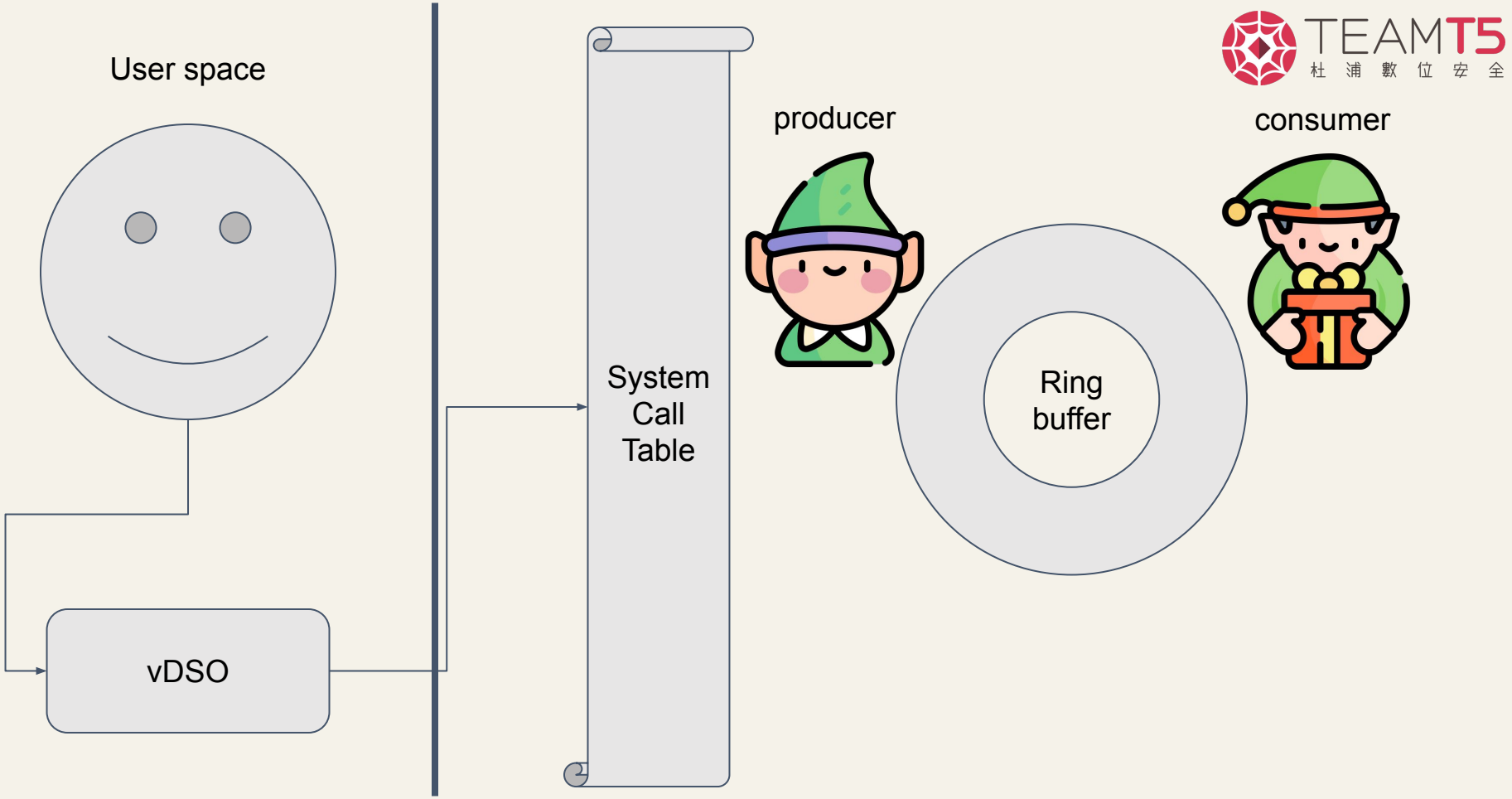
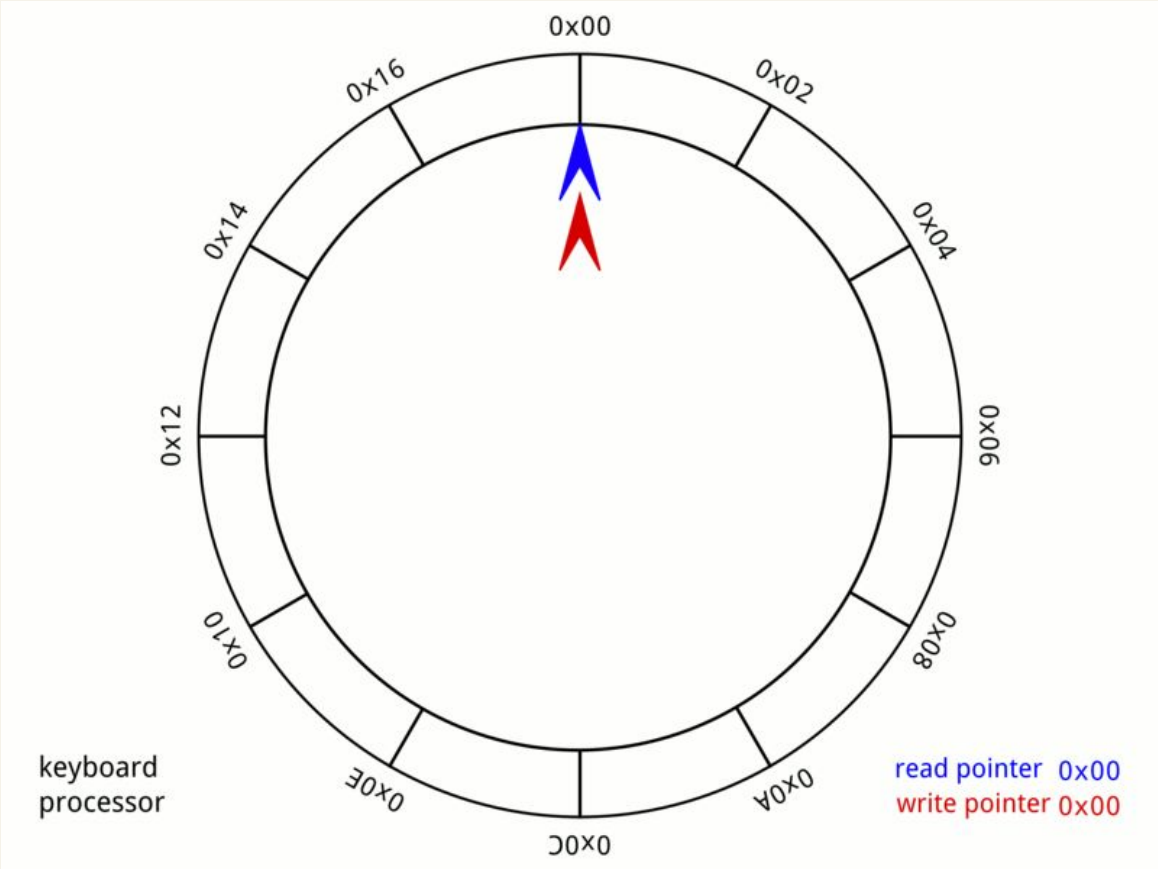
# Sonar-[C]ernel-Commander

Event tracing kernel provider on Linux

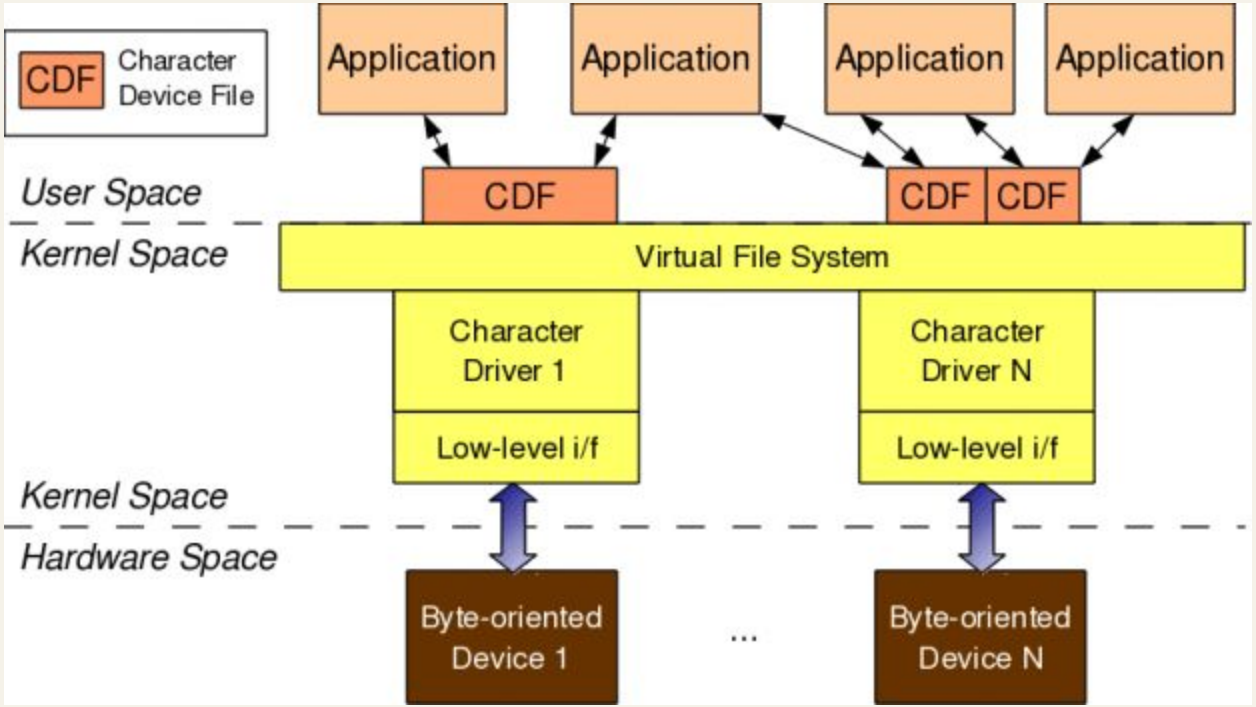
scc@teamt5.org


User space
vDSO
System
Call
Table
producer
Ring
buffer
consumer

# The ring buffer

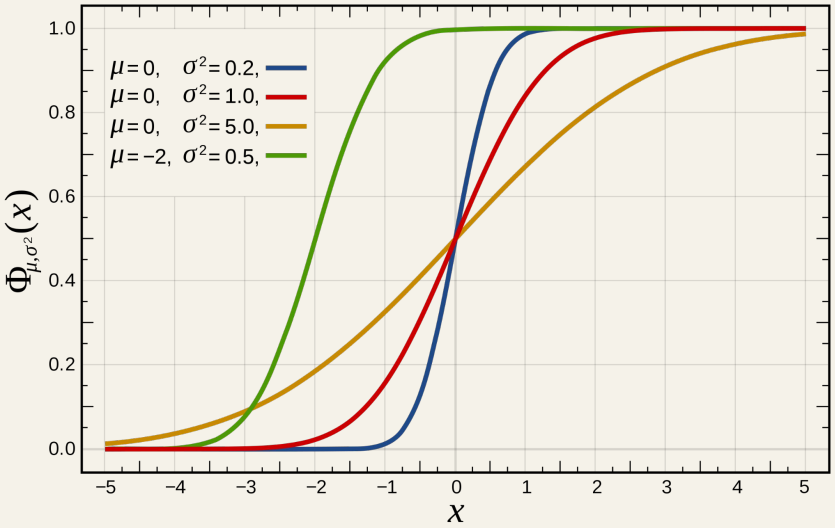


# Character device



Cumulative Distribution Function

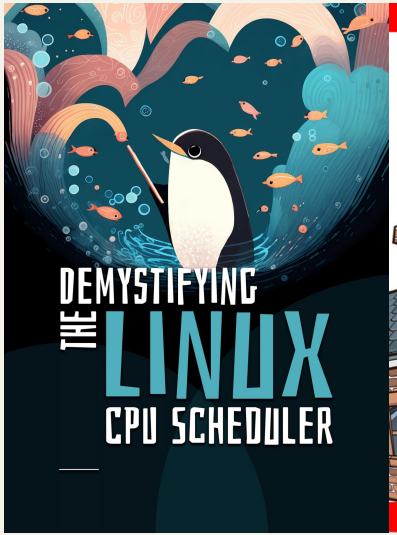
Character Device File

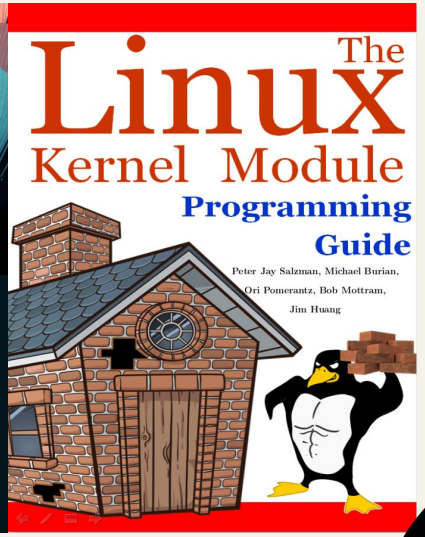


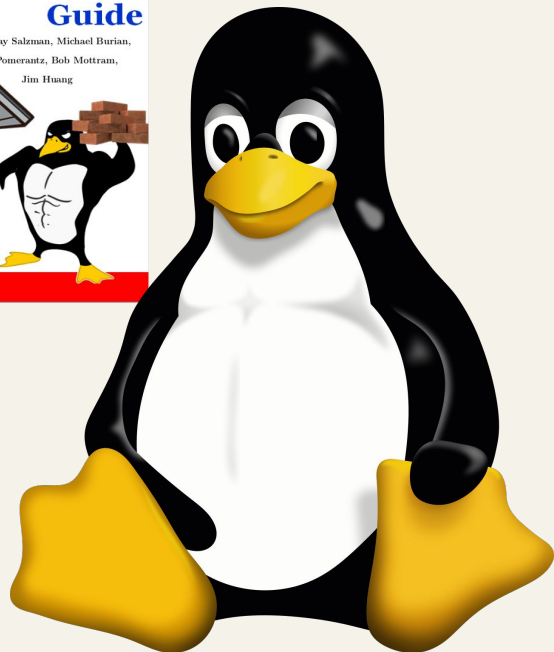
# Jserv Linux 核心設計/實作

TEAMT5
杜 浦 數 位 安 全

Ring buffer: 作業 1 (lab0-c)
cdev : 作業 2 (fibdrv)
ktcp : 作業 3



https://wiki.csie.ncku.edu.tw/linux/schedule

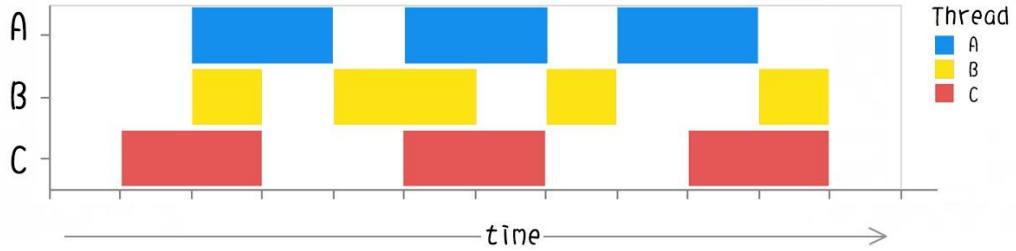
【寶寶的並行程式設計】中英雙語繪本

# Concurrent computing for Babies

# 寶寶的並行程式設計

自宅警備隊
宅色夫 著

從小培育時間管理大師




“ Redesigning your application to run multithreaded
on a multicore machine
is a little like learning to swim by jumping into the deep end. ”

- Herb Sutter

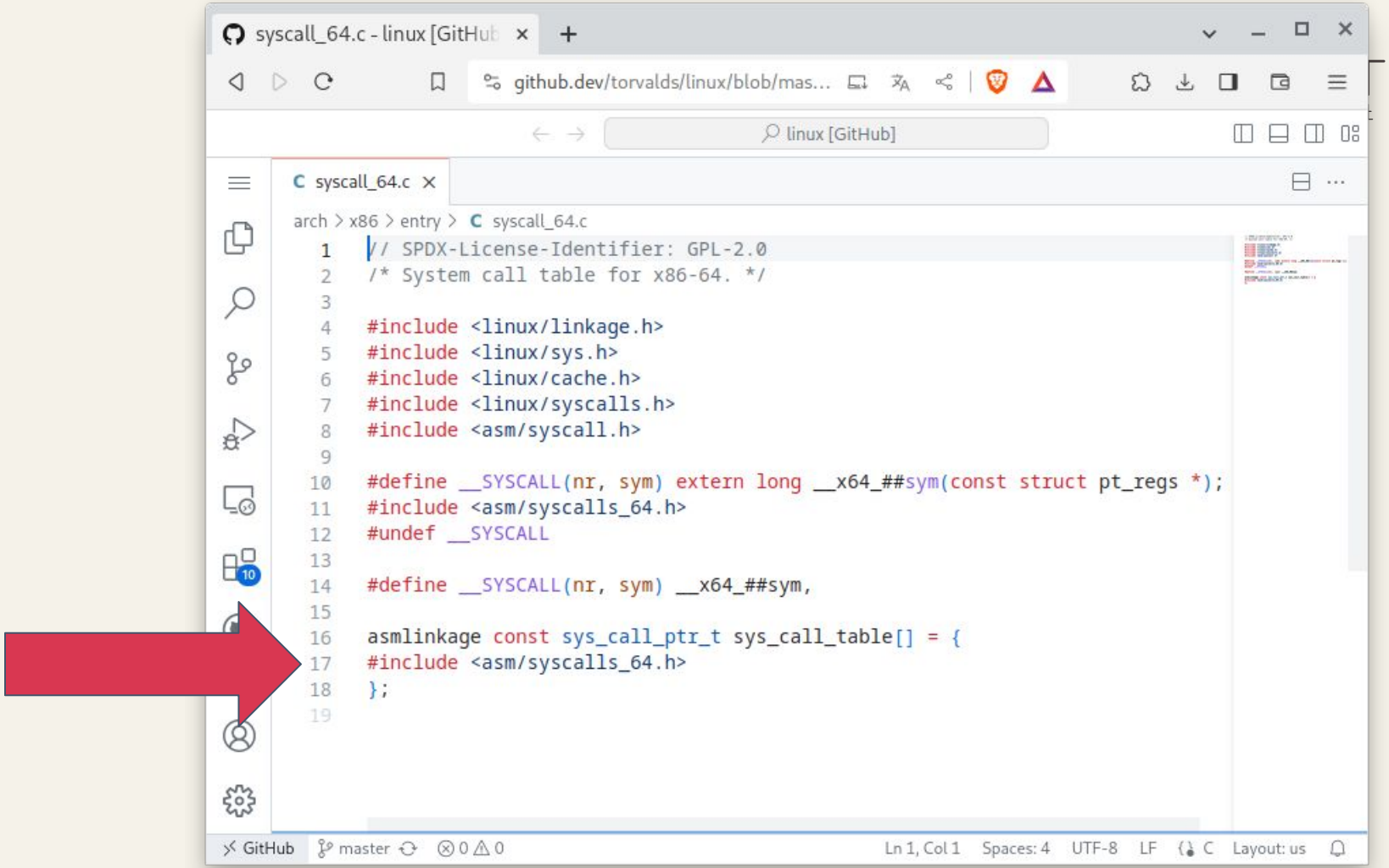
https://hackmd.io/@sysprog/concurrency

# Hook the system call

# HITCON 2019 CMT

TEAMT5
杜 浦 數 位 安 全


https://hitcon.org/2019/CMT/slide-files/d1_s1_r4.pdf


syscall_64.c - linux [GitHub]
github.dev/torvalds/linux/blob/mas...
linux [GitHub]
syscall_64.c
arch > x86 > entry > syscall_64.c
// SPDX-License-Identifier: GPL-2.0
/* System call table for x86-64. */
#include <linux/linkage.h>
#include <linux/sys.h>
#include <linux/cache.h>
#include <linux/syscalls.h>
#include <asm/syscall.h>
#define __SYSCALL(nr, sym) extern long __x64_##sym(const struct pt_regs *);
#include <asm/syscalls_64.h>
#undef __SYSCALL
#define __SYSCALL(nr, sym) __x64_##sym,
asmlinkage const sys_call_ptr_t sys_call_table[] = {
#include <asm/syscalls_64.h>
};
GitHub
master
Ln 1, Col 1
Spaces: 4
UTF-8
LF
C
Layout: us

# 腳踏實地法(苦幹實幹法)

1. 從 PAGE_OFFSET 開始找
2. 每 8 bytes 作比對
3. 檢查第 __NR_close 項是否等於 sys_close 的 address

# 偷看路標法

sys_call_table 的位置就在 code[3~6]

P.S. 要記得signed extended to 8 bytes

```
00000000000000ac <entry_SYSCALL_64_fastpath>:
      ac:   25 ff ff ff bf          and    eax,0xbfffffff
      b1:   3d 21 02 00 00          cmp    eax,0x221
      b6:   77 0f                   ja     c7 <entry_SYSCALL_64_fastpath+0x1b>
      b8:   4c 89 d1                mov    rcx,r10
      bb:   ff 14 c5 00 00 00 00    call   QWORD PTR [rax*8+0x0]
      c2:   48 89 44 24 50          mov    QWORD PTR [rsp+0x50],rax
      c7:   50                      push   rax
```

# 直接導航法

```
genesis@genesis:~$ sudo cat /proc/kallsyms | grep sys_call_table
ffffffff81a00200 R sys_call_table
```

```
genesis@genesis:~$ sudo cat /boot/System.map-4.4.0-138-generic | grep sys_call_table
ffffffff81a00200 R sys_call_table
```

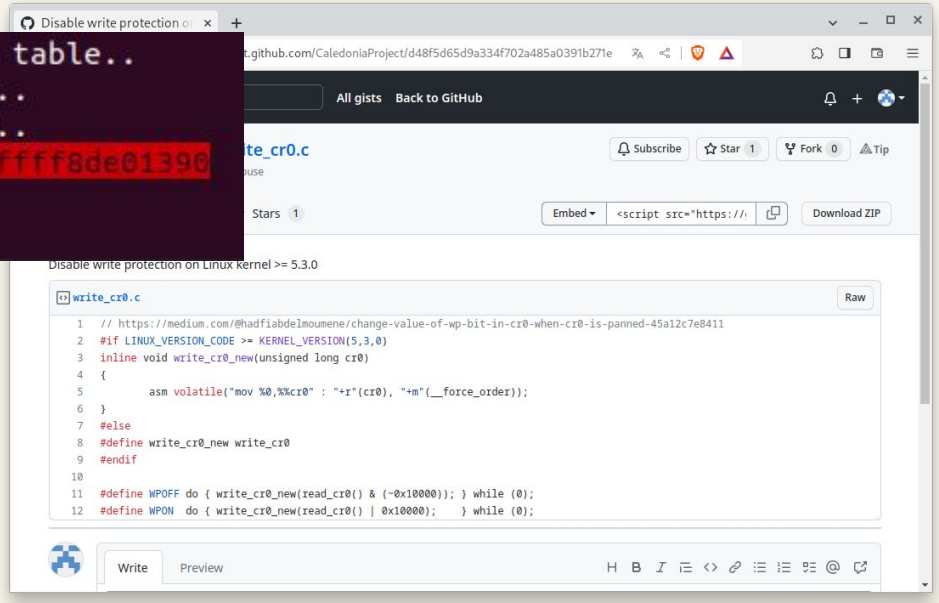
# 融合超強法

- 直接導航 X 腳踏實地
- 利用 sys_call_table 被存放在 rodata 段的特性
  - 透過 kallsyms_lookup_name 找到 _etext
  - 對齊 HPAGE_SIZE 後就是 rodata 的起始位置
- 之後比照腳踏實地法

# The WP is forced since 2019

```
$ sudo cat /proc/kallsyms | grep sys_call_table
ffffffffaf1aa0a0 t proc_sys_call_handler
ffffffffafe002a0 R sys_call_table
ffffffffafe01320 R ia32_sys_call_table
ffffffffafe020e0 R x32_sys_call_table
```

```
m4ko->m4ko_init: 42:  IAMHERE: Getting the syscall table..
m4ko->m4ko_init: 47:  IAMHERE: Saving the old call..
m4ko->m4ko_init: 49:  IAMHERE: Setting the new one..
BUG: unable to handle page fault for address: ffffffff8de01390
#PF: supervisor write access in kernel mode
#PF: error_code(0x0003) - permissions violation
```




https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=8dbec27a242cd3e2816eeb98d3237b9f57cf6232

https://outflux.net/blog/archives/2019/11/14/security-things-in-linux-v5-3/

## 9.1.1 Processor State After Reset

Following power-up, The state of control register CR0 is 60000010H (see Figure 9-1). This places the processor is in real-address mode with paging disabled.

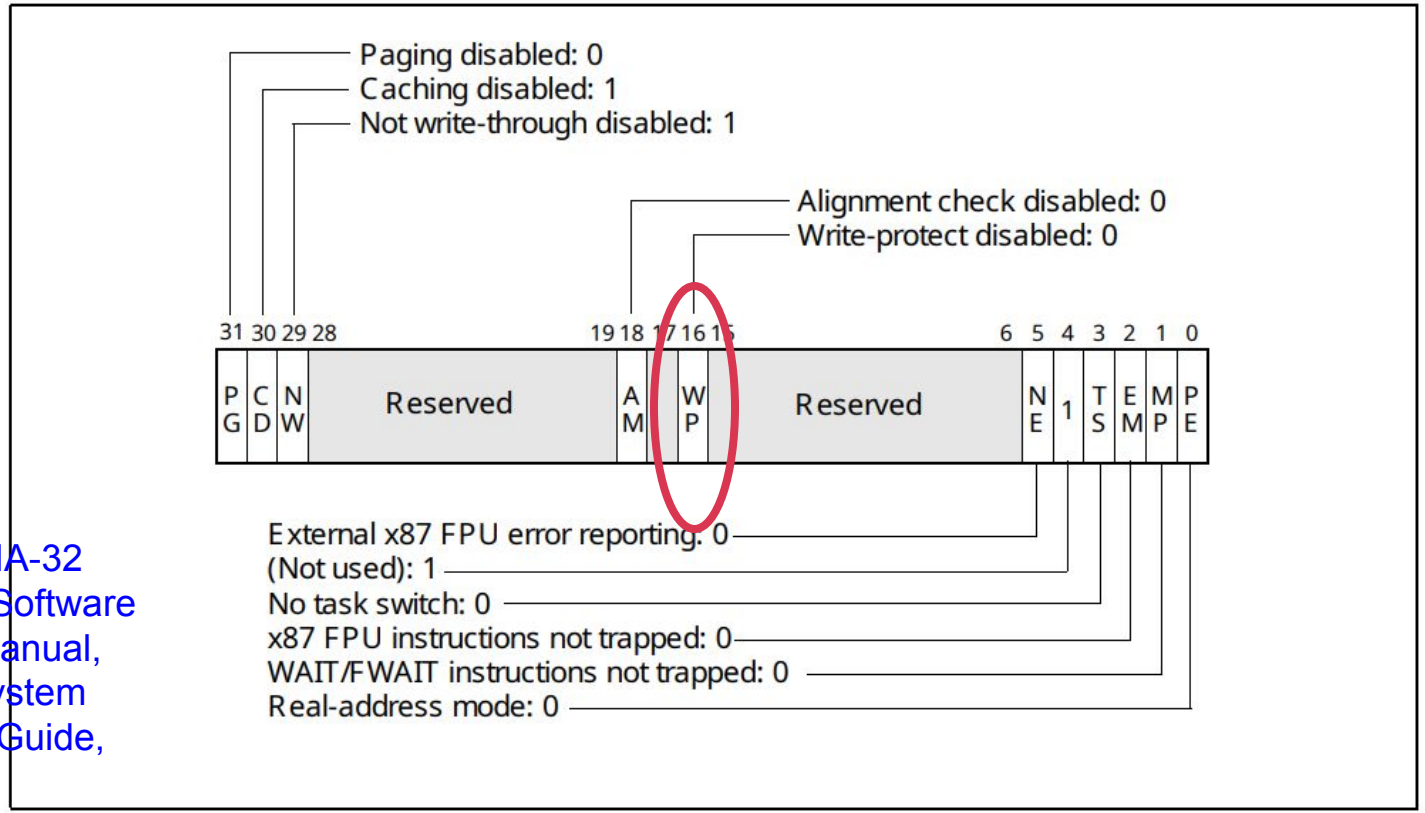



Figure 9-1. Contents of CR0 Register after Reset

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1

# The WP is forced since 2019

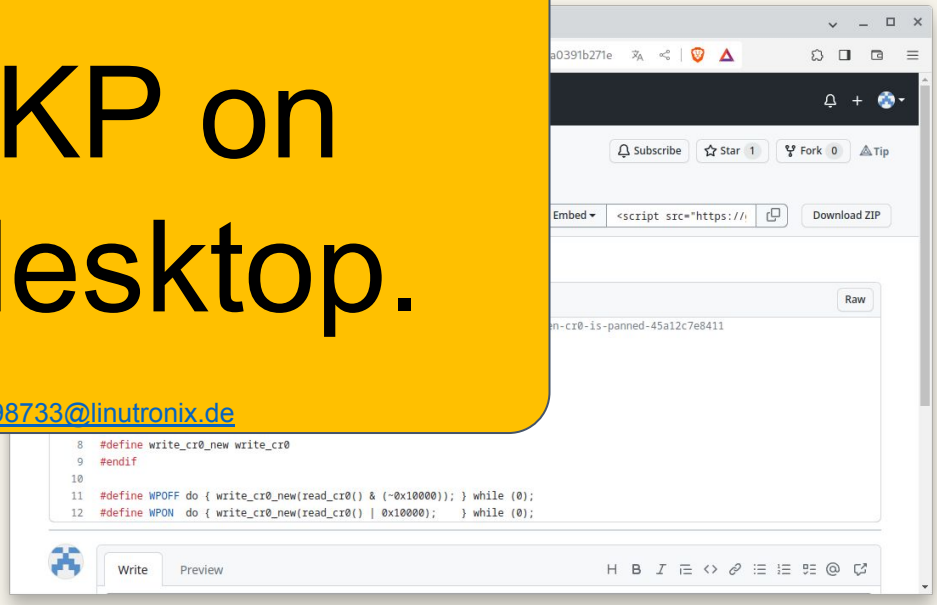
```
$ sudo cat /proc/kallsyms | grep sys_call_table
ffffffffaf1aa0a0 t proc_sys_call_handler
ffffffffafe002a0 R sys_call_table
ffffffffafe01320 R
ffffffffafe020e0
```

```
m4ko->m4ko_init: 4
m4ko->m4ko_init: 4
m4ko->m4ko_init: 4
BUG: unable to han
#PF: supervisor wr
#PF: error_code(0x
```

It still KP on DELL desktop.

Since v5.9

https://lkml.kernel.org/r/20180917143546.078998733@linutronix.de




https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=8dbec27a242cd3e2816eeb98d3237b9f57cf6232

https://outflux.net/blog/archives/2019/11/14/security-things-in-linux-v5-3/

# Compiler Bug?!

TEAMT5 杜浦數位安全

```
46  #define OUR_SYSCALL_IMPL(number, orig)                          \
47      noinline asmlinkage static void NEW_FUNC_##number(void)     \
48      {                                                           \
49          SAVE_REGS();                                            \
50          ASM_CALL_FP(event_logger_fp);                           \
51          RESTORE_REGS();                                         \
52          ASM_CALL_FP(orig);                                      \
53      }
```

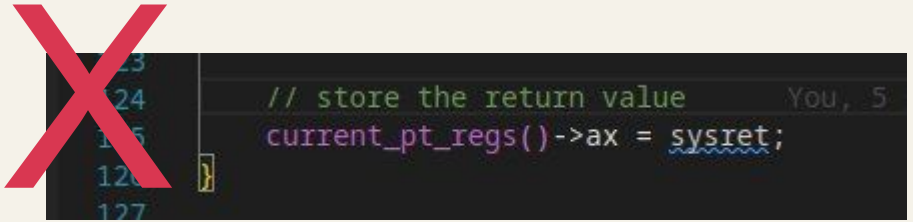
```
JNC_311>:
                endbr64
00              call   50a9 <NEW_FUNC_311+0x9>
                push   rax
                push   rbx
                push   rcx
                push   rdx
                push   rsi
                push   rdi
                push   rbp
13382       50af:   55
13383       50b0:   48 c7 c0 00 00 00 00    mov    rax,0x0
13384       50b7:   ff d0                   call   rax
13385       50b9:   5d                      pop    rbp
13386       50ba:   5f                      pop    rdi
13387       50bb:   5e                      pop    rsi
13388       50bc:   5a                      pop    rdx
13389       50bd:   59                      pop    rcx
13390       50be:   5b                      pop    rbx
13391       50bf:   58                      pop    rax
13392       50c0:   48 8b 05 00 00 00 00    mov    rax,QWORD PTR [rip+0x0]
13393       50c7:   ff d0                   call   rax
13394       50c9:   e9 00 00 00 00          jmp    50ce <NEW_FUNC_311+0x2e>
13395       50ce:   66 90                   xchg   ax,ax
13396
```

# Compiler Bug?!

TEAMT5 杜浦數位安全

```
0000000000001160 <NEW_FUNC_0>:
    1160:       50                      push   %rax
    1161:       53                      push   %rbx
    1162:       51                      push   %rcx
    1163:       52                      push   %rdx
    1164:       56                      push   %rsi
    1165:       57                      push   %rdi
    1166:       55                      push   %rbp
    1167:       48 8d 05 e2 ff ff ff    lea    -0x1e(%rip),%rax        # 1150 <myFunction>
    116e:       ff d0                   call   *%rax
    1170:       5d                      pop    %rbp
    1171:       5f                      pop    %rdi
    1172:       5e                      pop    %rsi
    1173:       5a                      pop    %rdx
    1174:       59                      pop    %rcx
    1175:       5b                      pop    %rbx
    1176:       58                      pop    %rax
    1177:       ff d0                   call   *%rax
    1179:       c3                      ret
```

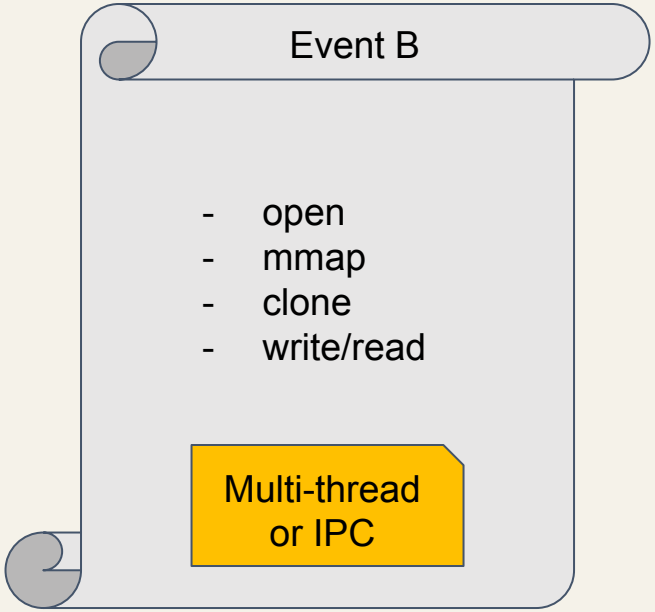
```
46  #define OUR_SYSCALL_IMPL(number, orig)                          \
47      noinline asmlinkage static void NEW_FUNC_##number(void) \
48      {                                                           \
49          SAVE_REGS();                                            \
50          ASM_CALL_FP(event_logger_fp);                           \
51          RESTORE_REGS();                                         \
52          ASM_CALL_FP(orig);                                      \
53          PUSH_REG(ax);                                           \
54          ASM_CALL_FP(post_event_logger_fp);                      \
55          POP_REG(ax);                                            \
56      }       You, 上週 • feat: imple. our syscall functor
57
58  #endif // __SCC_SYS_CALL_TABLE_H__
```



```
46  #define OUR_SYSCAL
47      noinline asmli
48      {
49          SAVE_REGS(
50          ASM_CALL_F
51          RESTORE_RE
52          ASM_CALL_F
53          PUSH_REG(a
54          ASM_CALL_F
55          POP_REG(ax
56      }       You, H
57
58  #endif // __SCC_SY
```

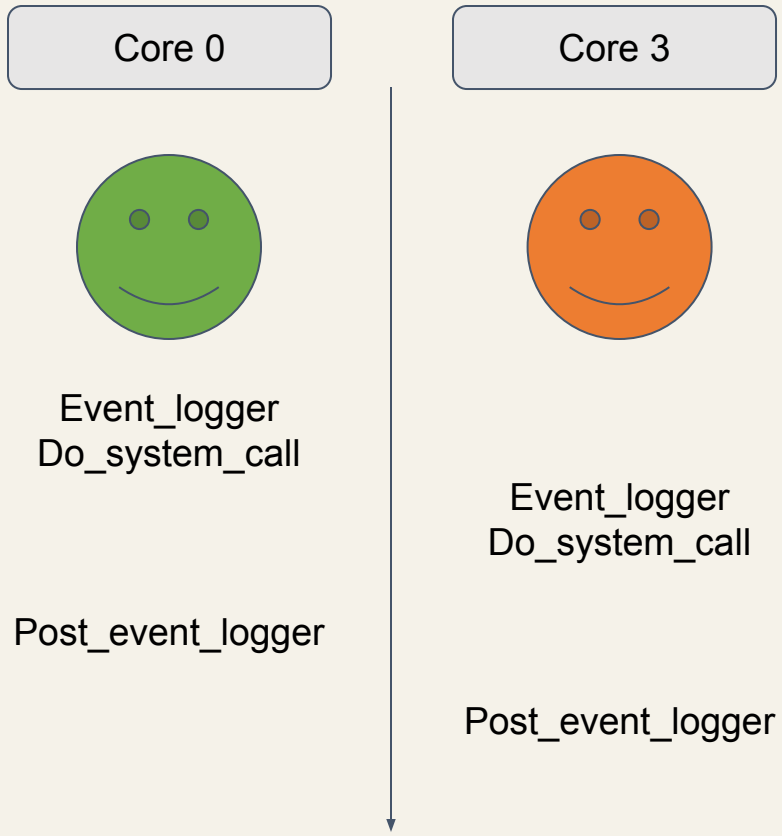
# Events creation

# Big problems

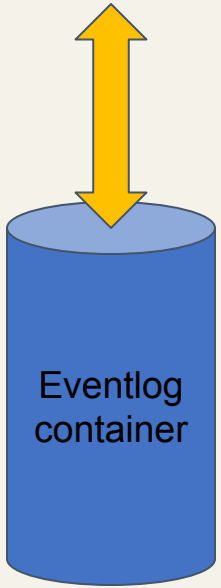
# Big problems

1. Event_logger
2. Do_system_call
3. Post_event_logger



# Big problems

1. Event_logger
2. Do_system_call
3. Post_event_logger




Core 0

Core 3

Event_logger
Do_system_call

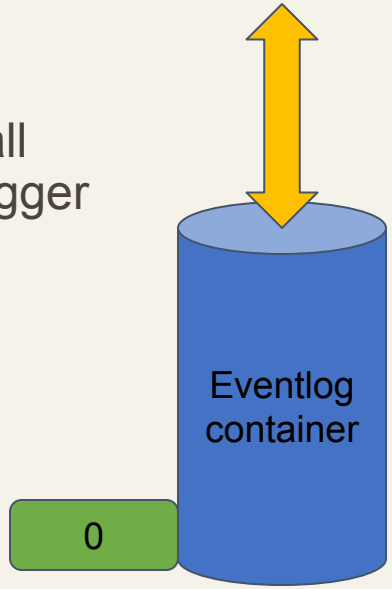
Post_event_logger

Event_logger
Do_system_call

Post_event_logger

# Big problems

1. Event_logger
2. Do_system_call
3. Post_event_logger




Core 0

Core 3

Event_logger
Do_system_call

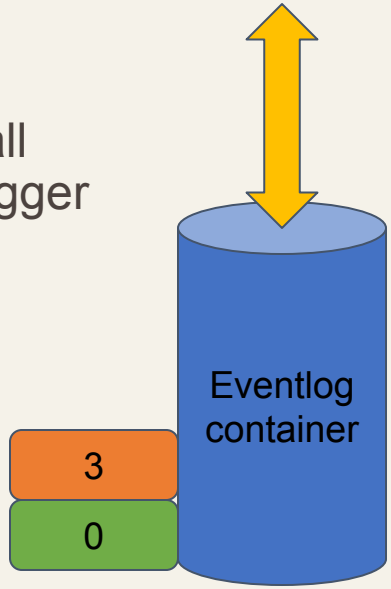
Post_event_logger

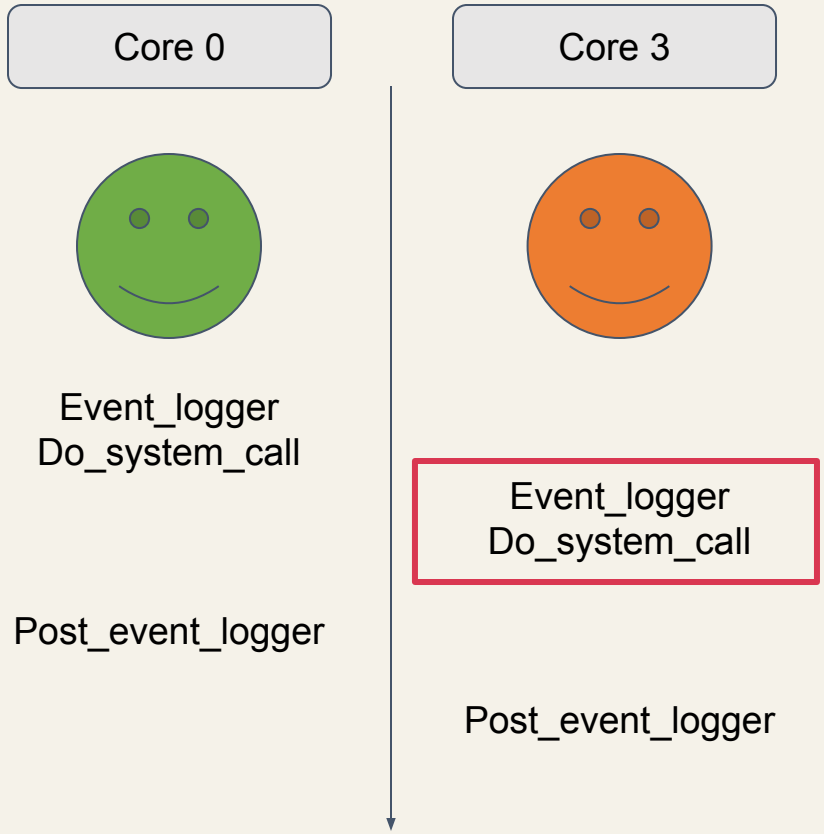
Event_logger
Do_system_call

Post_event_logger

# Big problems

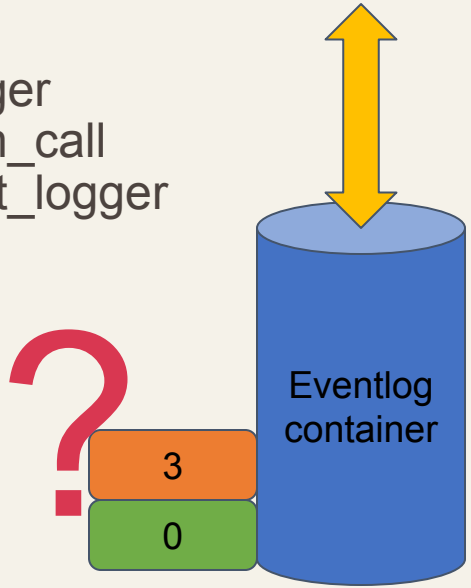
1. Event_logger
2. Do_system_call
3. Post_event_logger

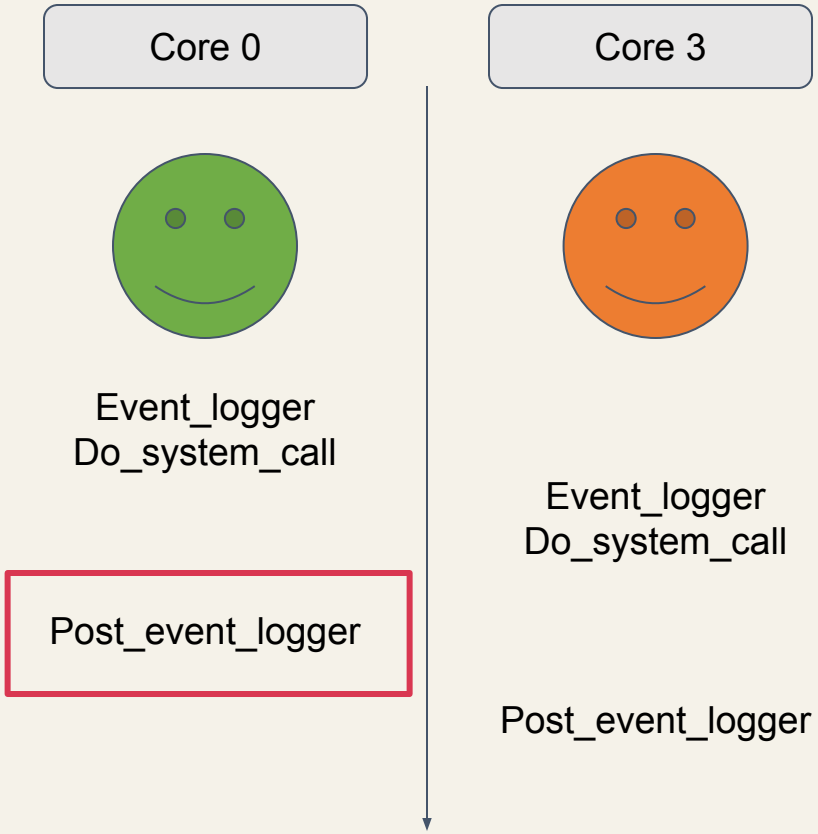


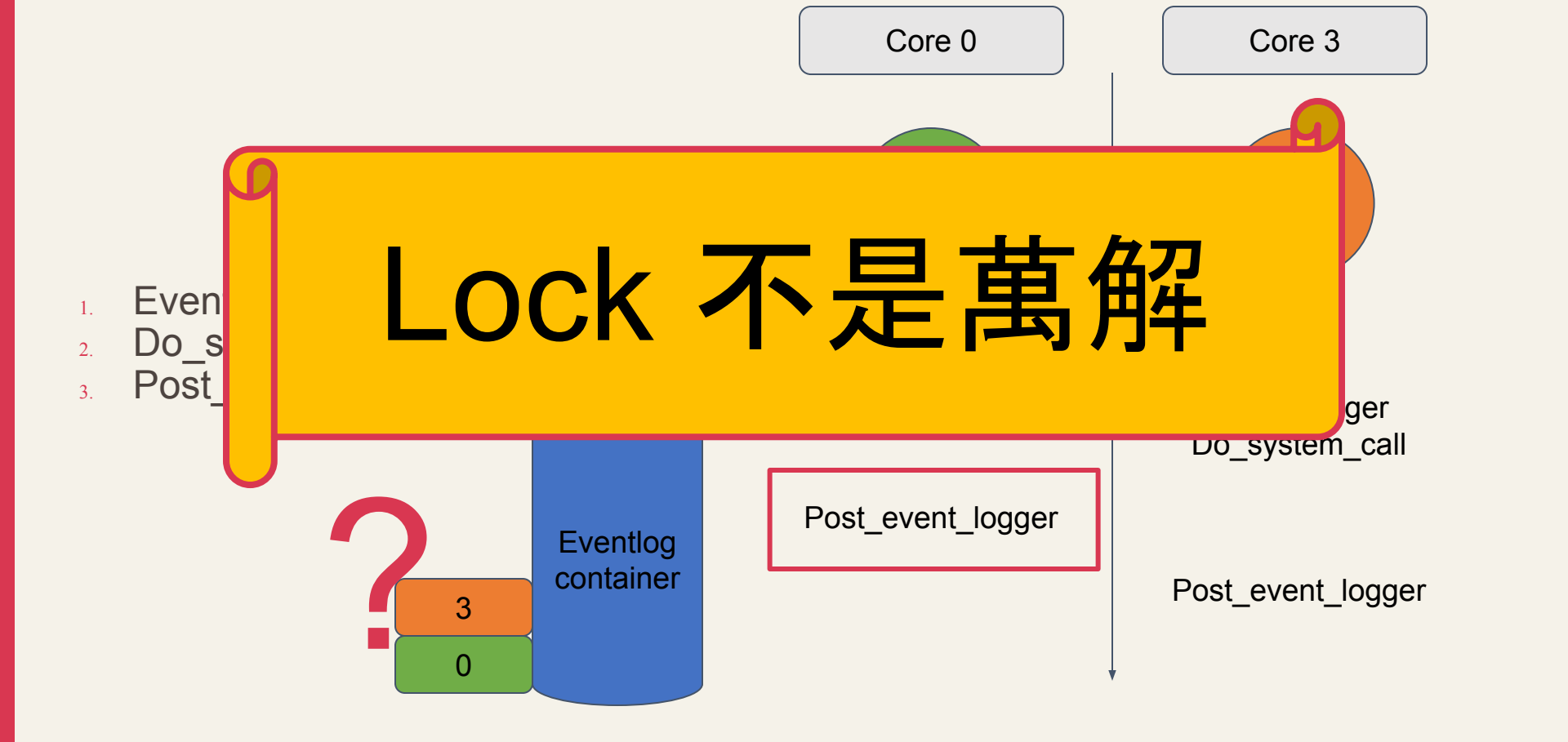
# Big problems

1. Event_logger
2. Do_system_call
3. Post_event_logger



# Big problems



# BKL

TEAMT5 杜浦數位安全

Linux 2.4 在 SMP 的效率問題在哪？

- Big Kernel Lock: 鎖定整個 Linux 核心的機制，透過 `lock_kernel()` 與 `unlock_kernel()` 函式。這是為了支援多處理器，暫時引進的鎖定機制，已在 `v2.6.39` (2011 年，在 Linux SMP 支援引入的 15 年後!) 徹底清除。
  - `v2.6.6` still had about 500 lock_kernel() calls
  - commit
  - Killing the Big Kernel Lock
  - BKL 用於保護整個核心，而 spinlock 用於保護非常特定的某一共享資源。行程 (process) 持有 BKL 時允許發生排程。實作機制：在執行 `schedule` 時，`schedule` 將檢查行程是否擁有 BKL，若有，它將被釋放，以致於其它的行程能夠獲得該 lock，而當輪到該行程執行之際，再讓它重新獲得 BKL。注意：在持有 spinlock 期間不會發生排程。
  - 整個核心僅有一個 BKL

延伸閱讀:

- 並行和多執行緒程式設計系列講座
- Linux 核心設計: 淺談同步機制
- Linux 核心設計: 多核處理器和 spinlock

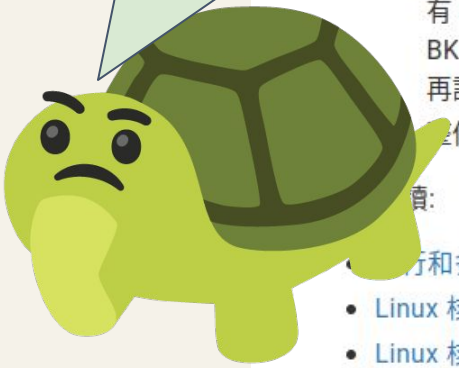
Big Kernel Lock 是個顯著的問題，但 scalability 並非單一議題，需要顧及各種議題。

不可以讓多核變成單核

# BKL

TEAMT5 杜浦數位安全

Hash table:
- Run time cost?
- Collision?
- Extension?



SMP 的效率問題在哪？

定整個 Linux 核心的機制，透過 `lock_kernel()` 與 `unlock_kernel()` 函
理器，暫時引進的鎖定機制，已在 `v2.6.39` (2011 年，在 Linux SMP 支
清除。
about 500 lock_kernel() calls

Big Kernel Lock

BKL 用於保護整個核心，而 spinlock 用於保護非常特定的某一共享資源。行程 (process) 持有 BKL 時允許發生排程。實作機制：在執行 `schedule` 時，`schedule` 將檢查行程是否擁有 BKL，若有，它將被釋放，以致於其它的行程能夠獲得該 lock，而當輪到該行程執行之際，再讓它重新獲得 BKL。注意：在持有 spinlock 期間不會發生排程。
個核心僅有一個 BKL

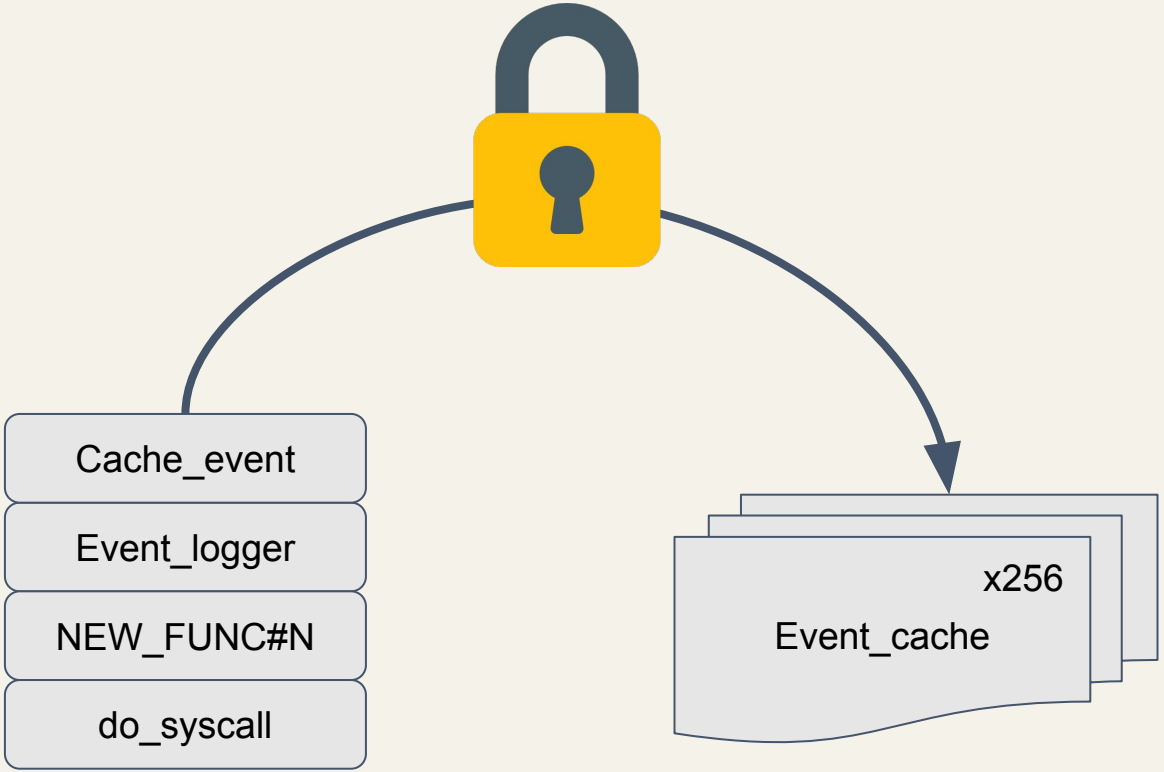
續:

- 和多執行緒程式設計系列講座
- Linux 核心設計: 淺談同步機制
- Linux 核心設計: 多核處理器和 spinlock

Big Kernel Lock 是個顯著的問題，但 scalability 並非單一議題，需要顧及各種議題。

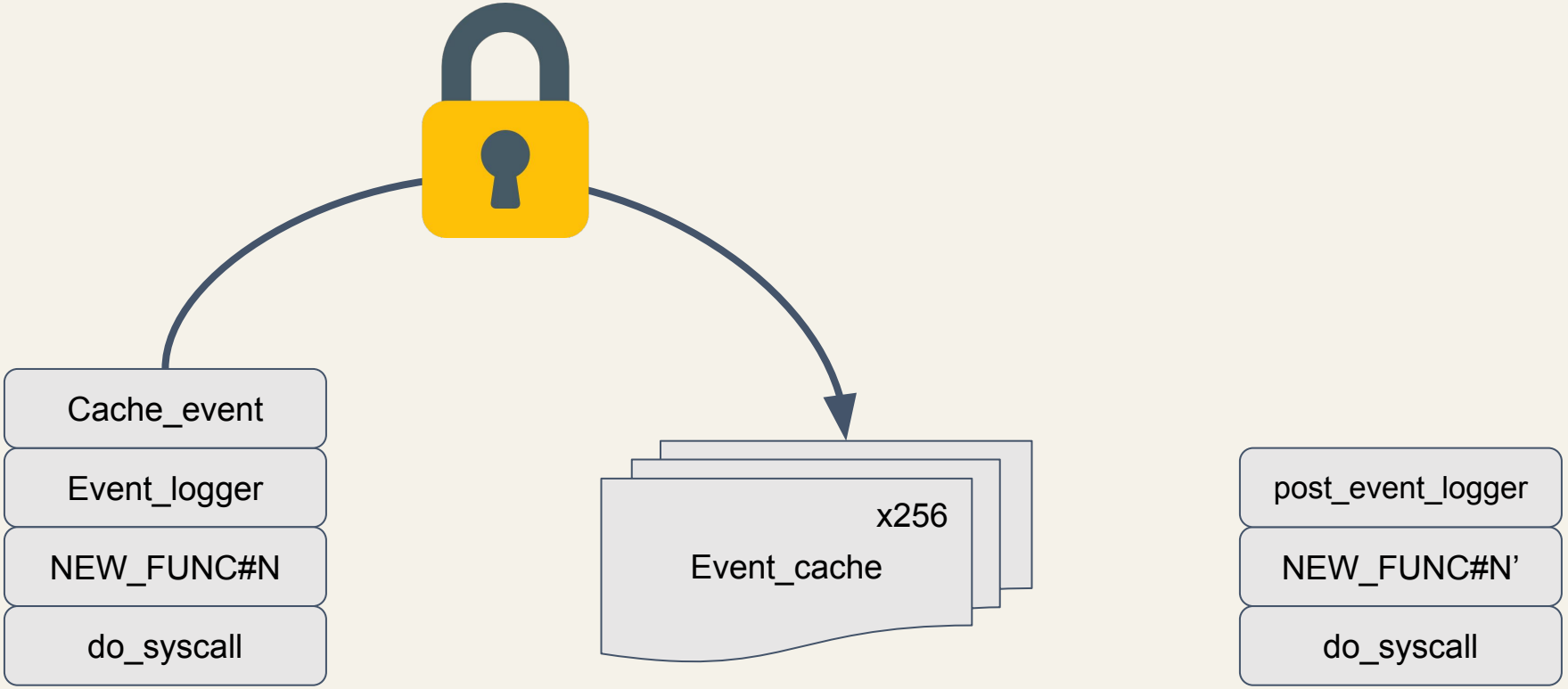
不可以讓多核變成單核

# Finally:

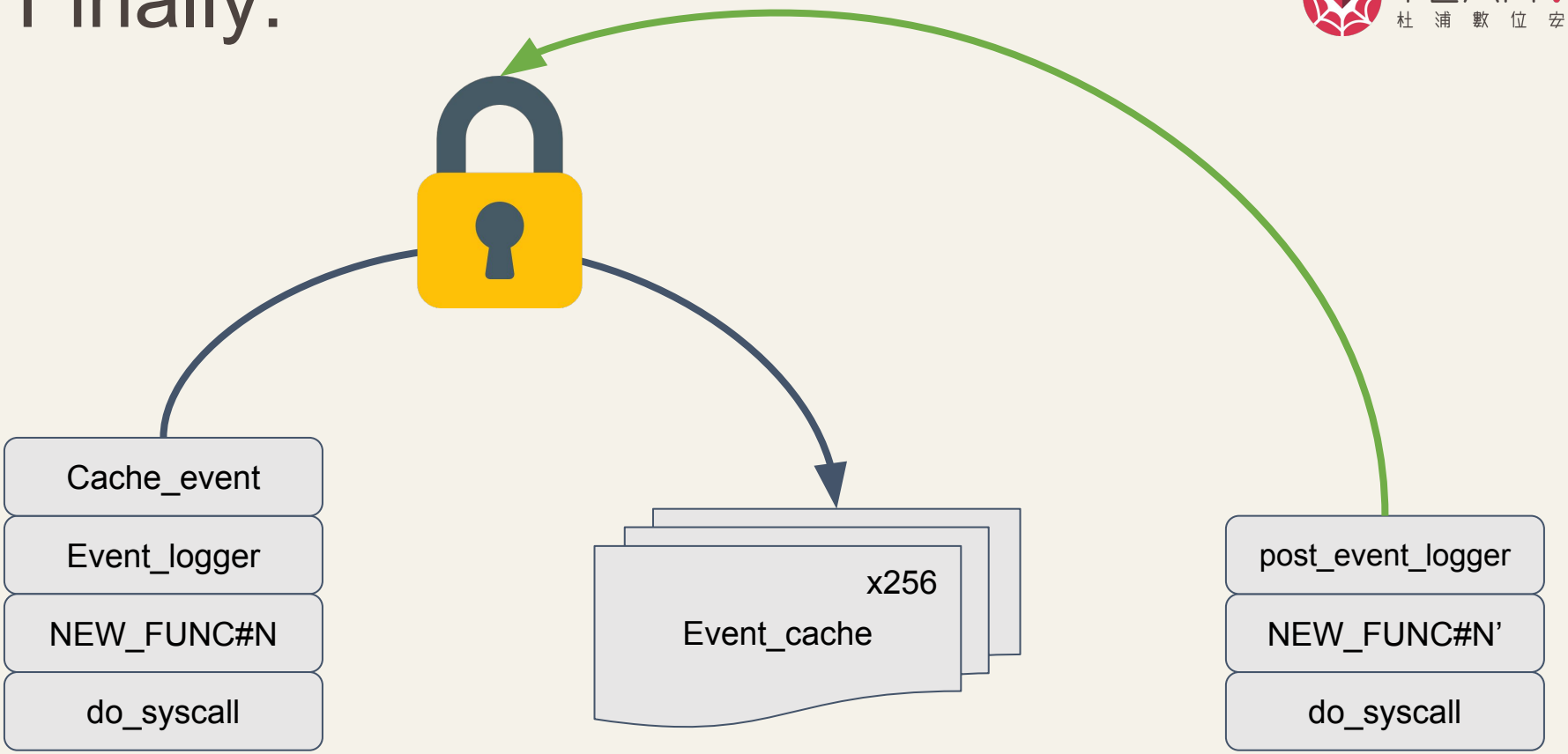

Cache_event
Event_logger
NEW_FUNC#N
do_syscall
x256
Event_cache

# Finally:

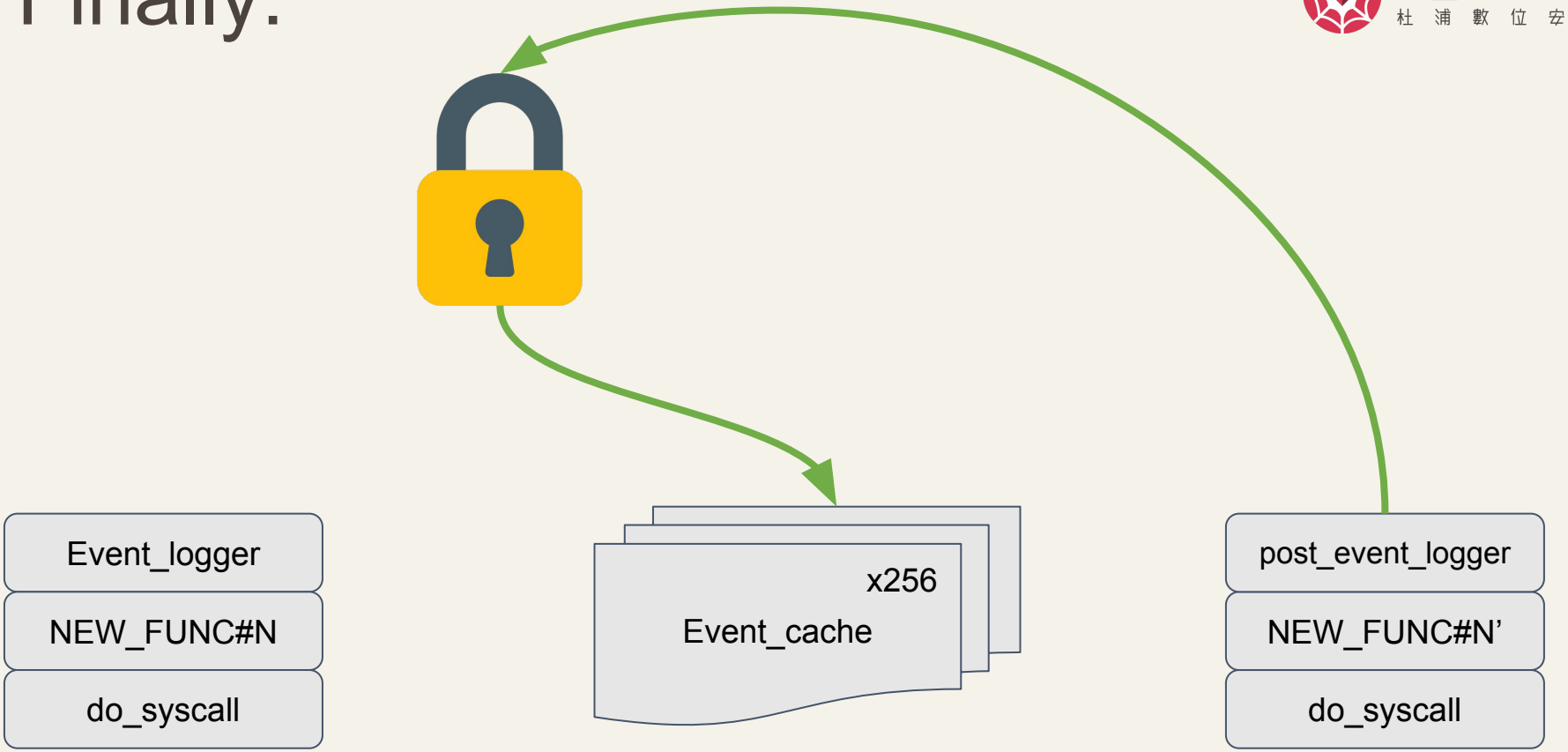
TEAMT5
杜 浦 數 位 安 全



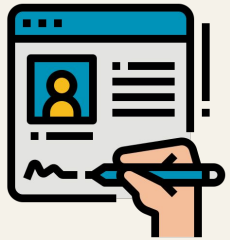
# Finally:


Cache_event
Event_logger
NEW_FUNC#N
do_syscall
x256
Event_cache
post_event_logger
NEW_FUNC#N'
do_syscall

# Finally:


Event_logger
NEW_FUNC#N
do_syscall
x256
Event_cache
post_event_logger
NEW_FUNC#N'
do_syscall

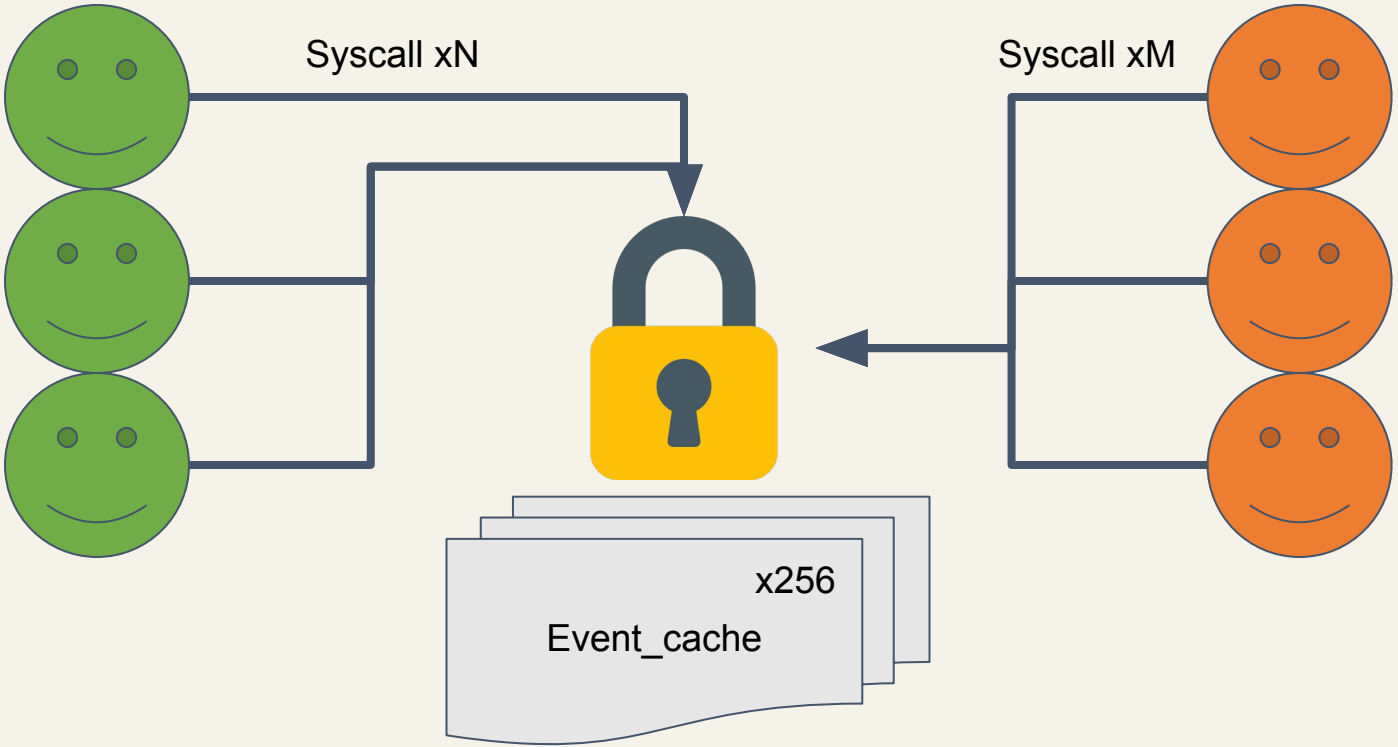
# Finally:

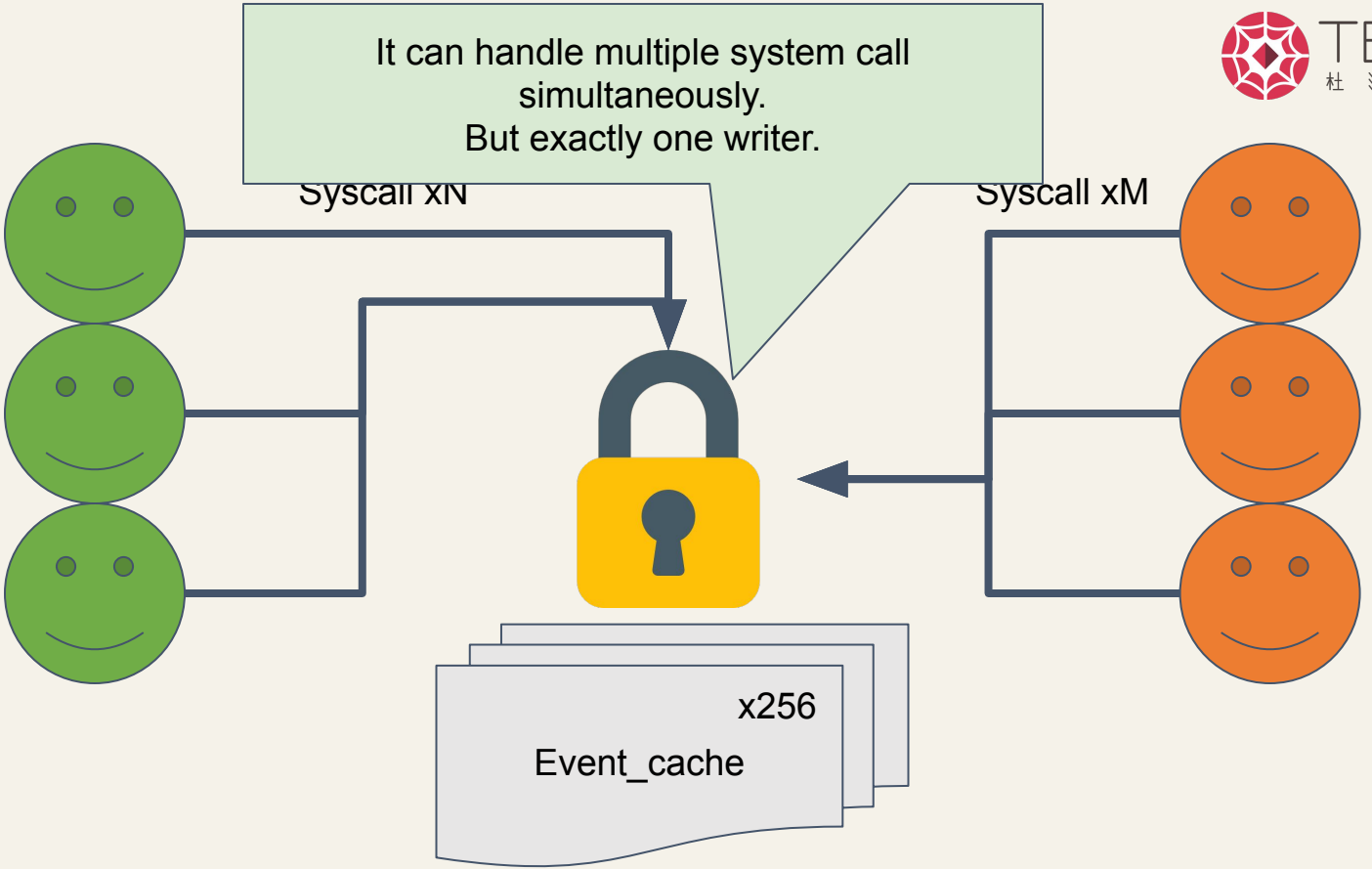


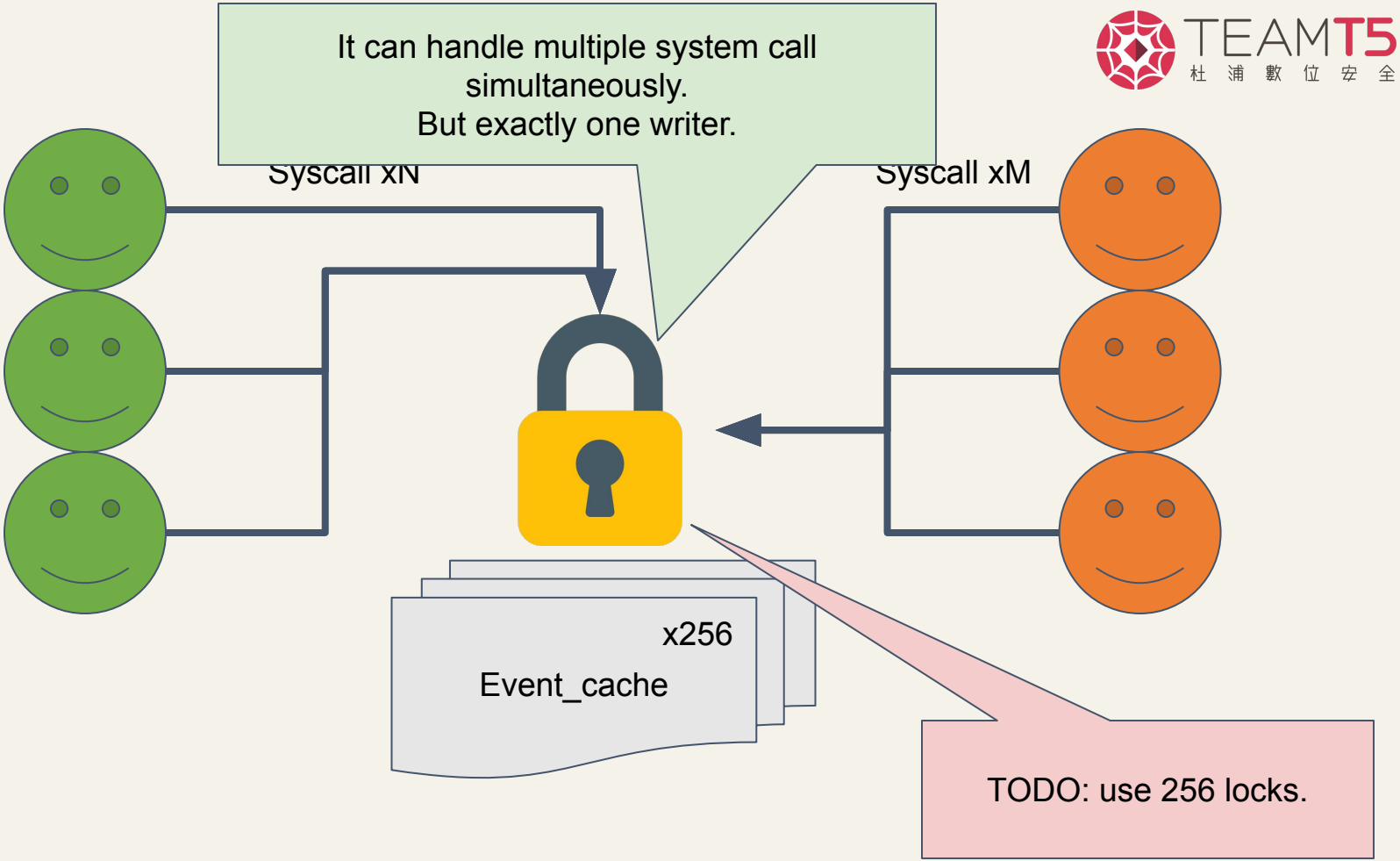
Event_logger
NEW_FUNC#N
do_syscall

x256
Event_cache

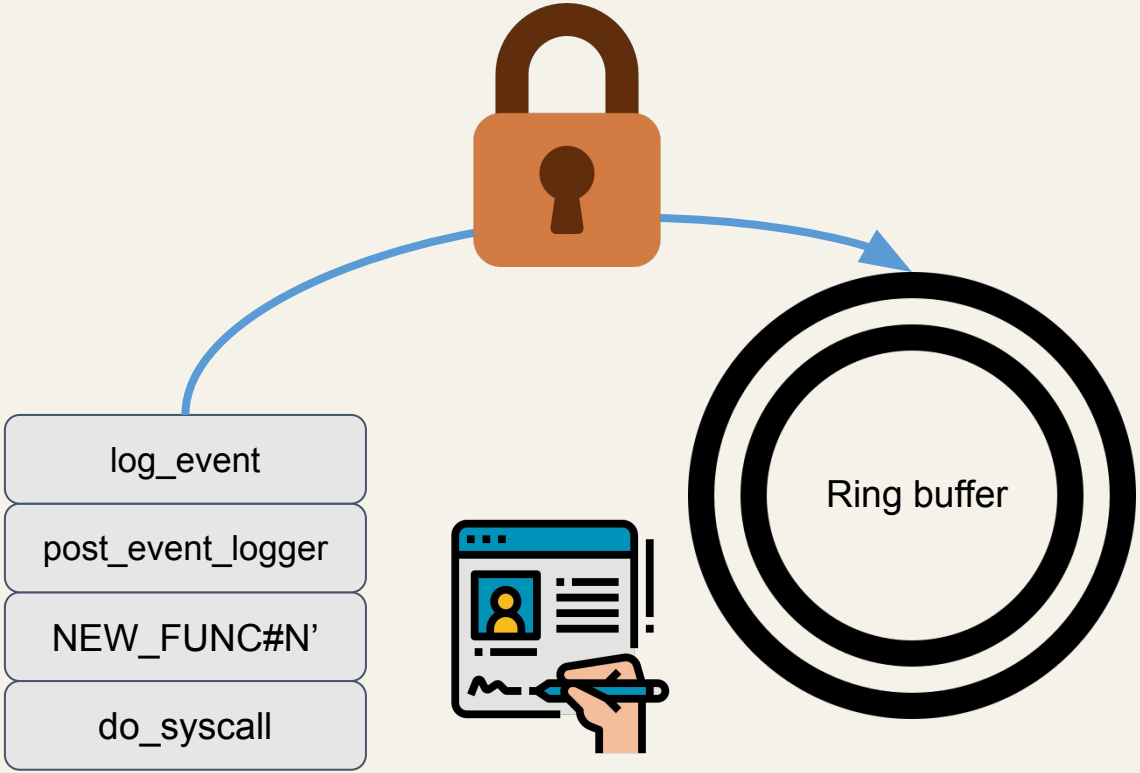
post_event_logger
NEW_FUNC#N’
do_syscall


Syscall xN
Syscall xM
x256
Event_cache

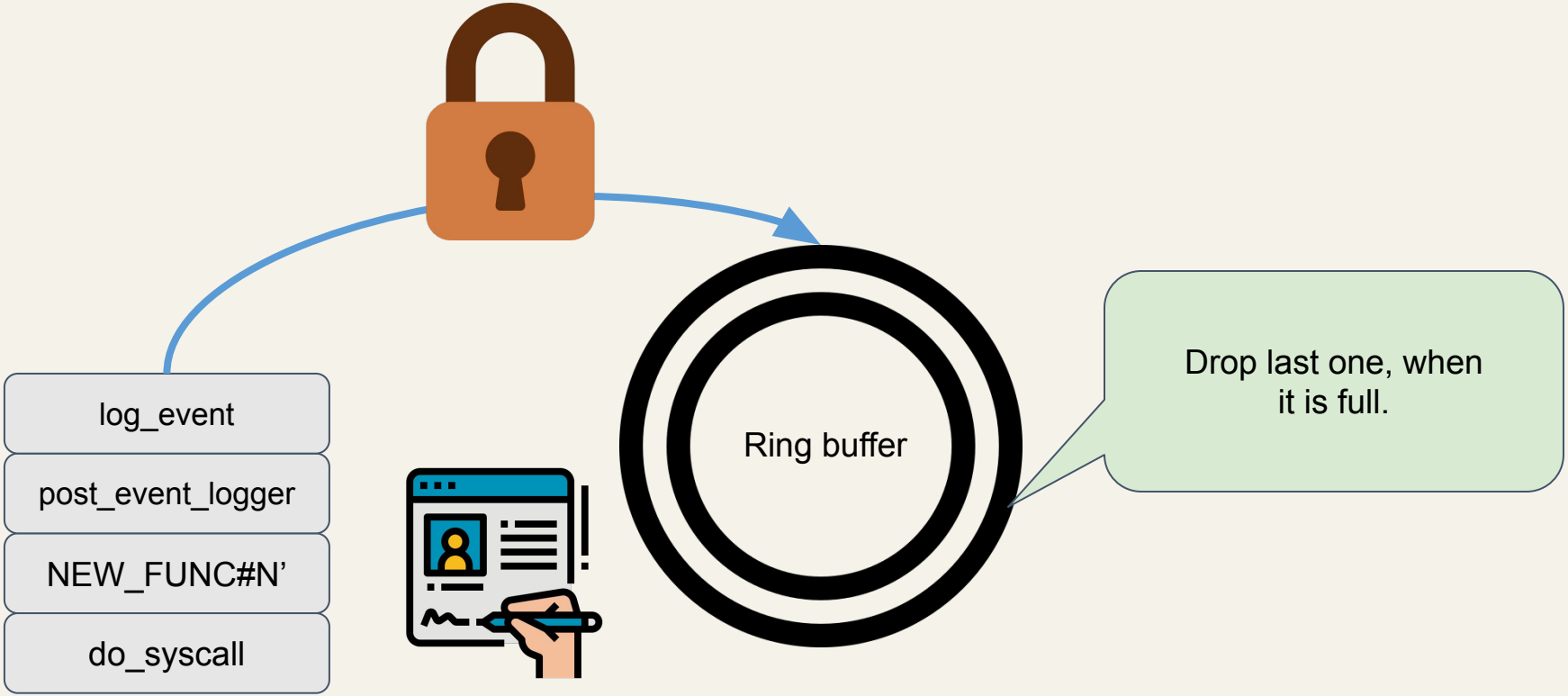

It can handle multiple system call
simultaneously.
But exactly one writer.
Syscall xN
Syscall xM
x256
Event_cache


It can handle multiple system call simultaneously.
But exactly one writer.
Syscall xN
Syscall xM
x256
Event_cache
TODO: use 256 locks.

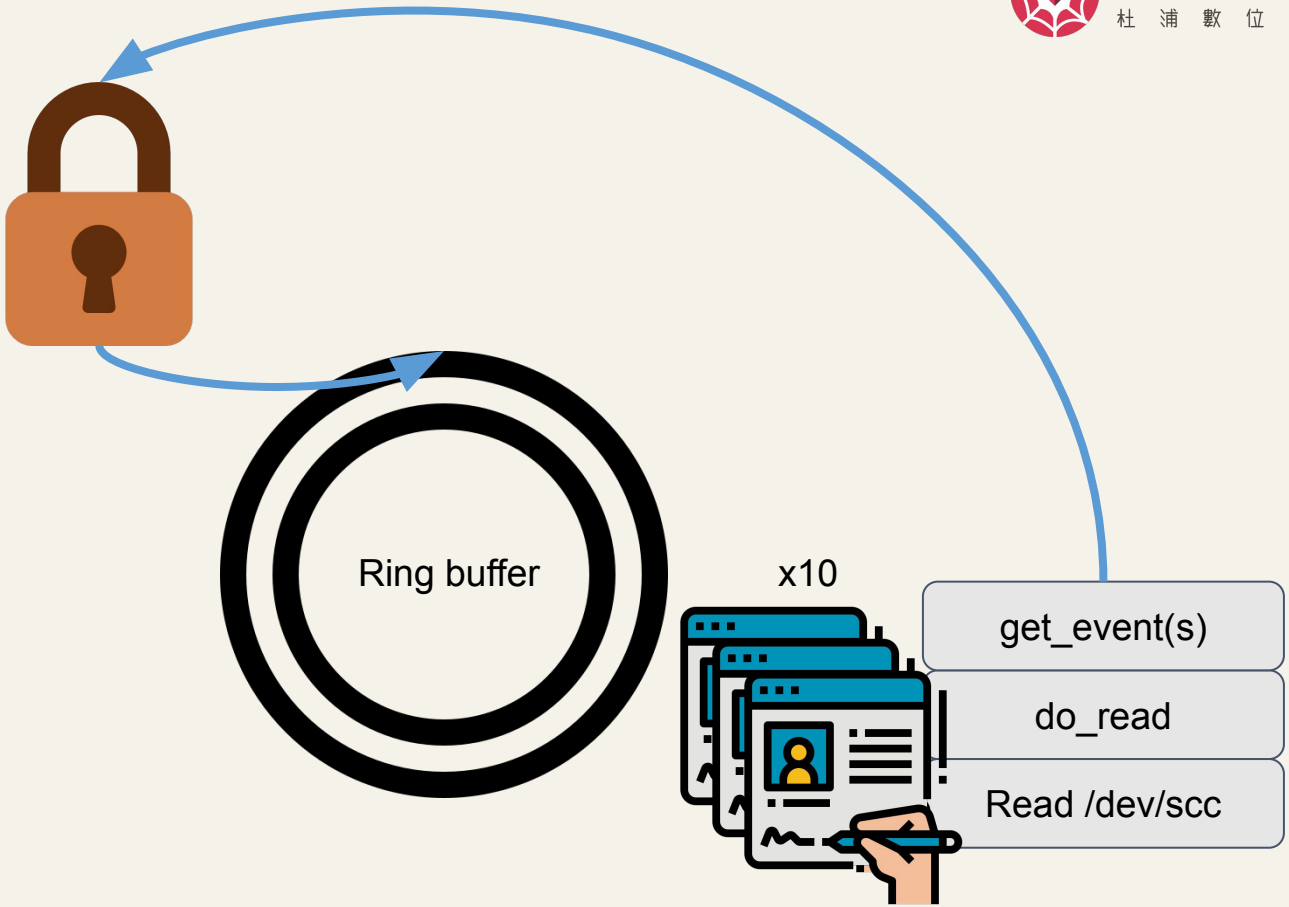
# Finally:


log_event
post_event_logger
NEW_FUNC#N’
do_syscall
Ring buffer

# Finally:


log_event
post_event_logger
NEW_FUNC#N’
do_syscall
Ring buffer
Drop last one, when
it is full.

# Finally:



# Lock?!

TEAMT5
杜 浦 數 位 安 全

```
#define lock_completion(comp, lock) \
    do                                \
    {                                 \
        if (mutex_trylock(lock))      \
            break;                    \
        wait_for_completion(comp);    \
    } while (1)
```

# Lock?!

TEAMT5
杜 浦 數 位 安 全

```
#define lock_completion(comp, lock) \
    do                              \
    {                               \
        if (mutex_trylock(lock))    \
            break;                  \
        wait_for_completion(comp);  \
    } while (1)
```

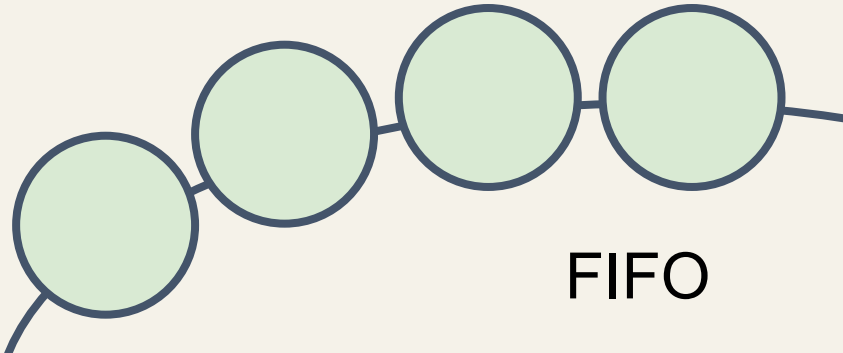
```
struct completion {
        unsigned int done;
        struct swait_queue_head wait;
    };
```

# Lock?!

TEAMT5
杜 浦 數 位 安 全

```
#define lock_completion(comp, lock) \
    do                              \
    {                               \
        if (mutex_trylock(lock))    \
            break;                  \
        wait_for_completion(comp);  \
    } while (1)
```

```
struct completion {
        unsigned int done;
        struct swait_queue_head wait;
    };
```

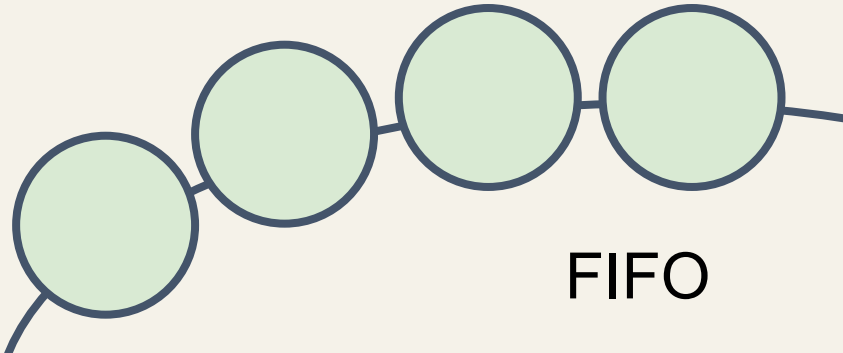


# Lock?!


TEAMT5
杜 浦 數 位 安 全

```
#define lock_completion(comp, lock) \
    do                              \
    {                               \
        if (mutex_trylock(lock))    \
            break;                  \
        wait_for_completion(comp);  \
    } while (1)
```

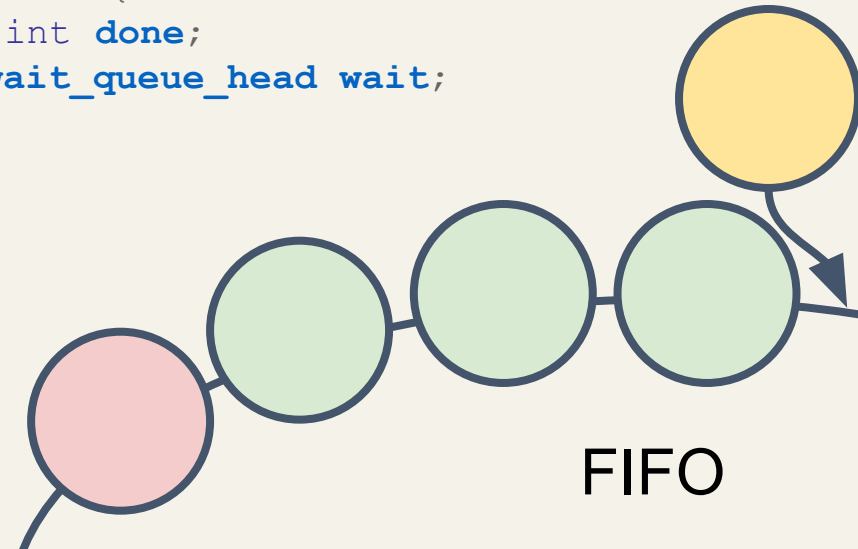
```
struct completion {
        unsigned int done;
        struct swait_queue_head wait;
    };
```



# Lock?! But…

```
#define lock_completion(comp, lock) \
    do                              \
    {                               \
        if (mutex_trylock(lock))    \
            break;                  \
        wait_for_completion(comp);  \
    } while (1)
```

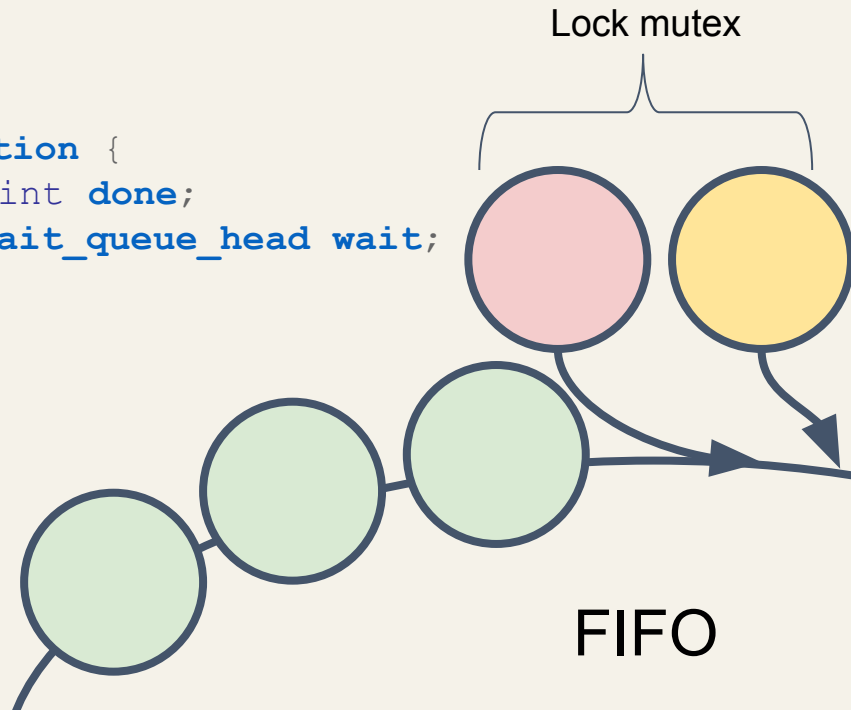
```
struct completion {
    unsigned int done;
    struct swait_queue_head wait;
};
```



# Lock?! But…

```
#define lock_completion(comp, lock) \
    do                              \
    {                               \
        if (mutex_trylock(lock))    \
            break;                  \
        wait_for_completion(comp);  \
    } while (1)
```

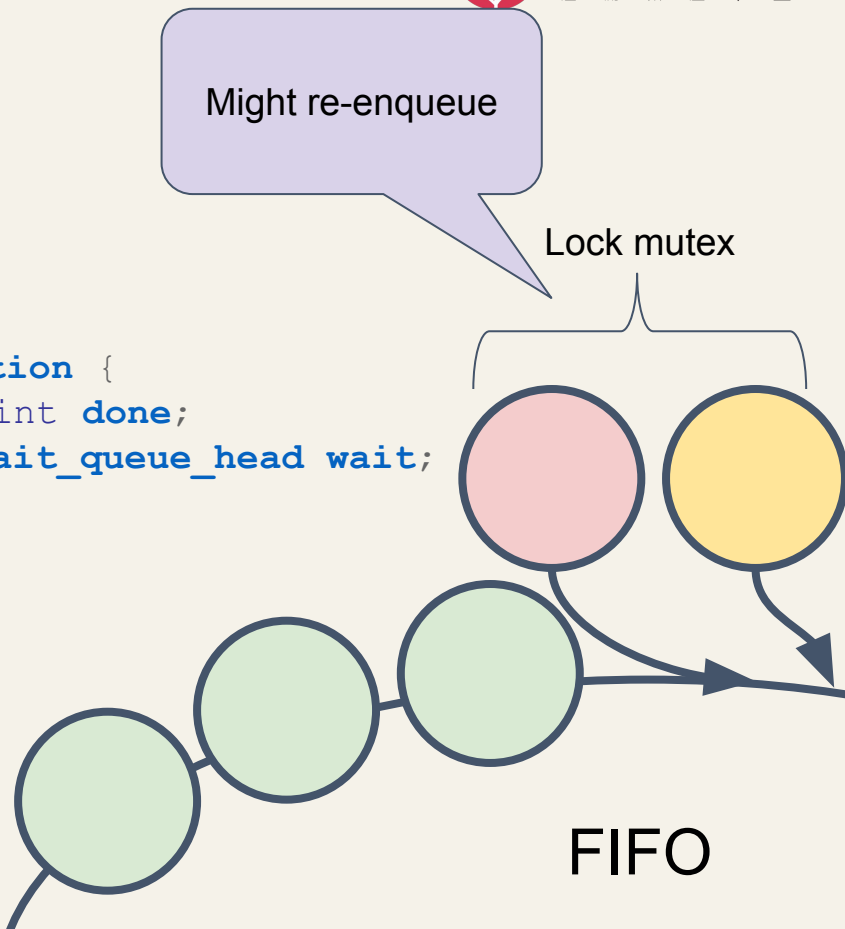
```
struct completion {
    unsigned int done;
    struct swait_queue_head wait;
};
```

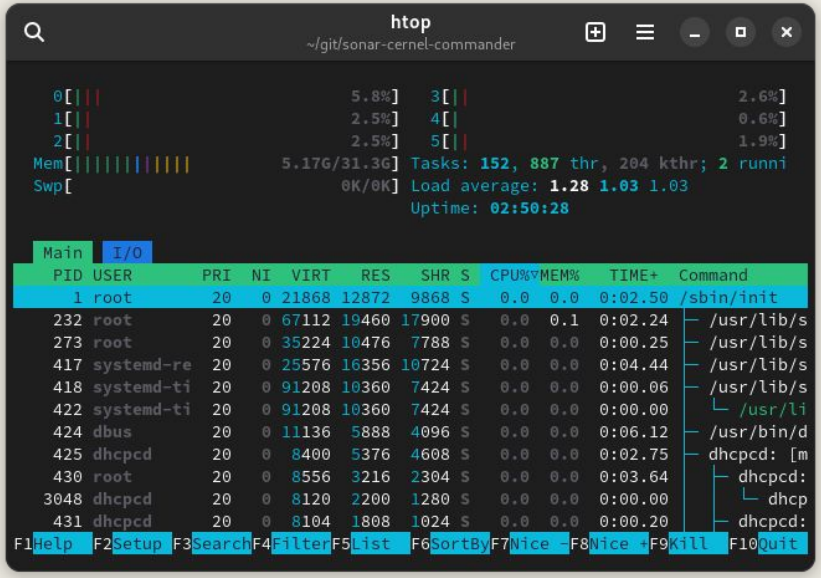


# Lock?! But…

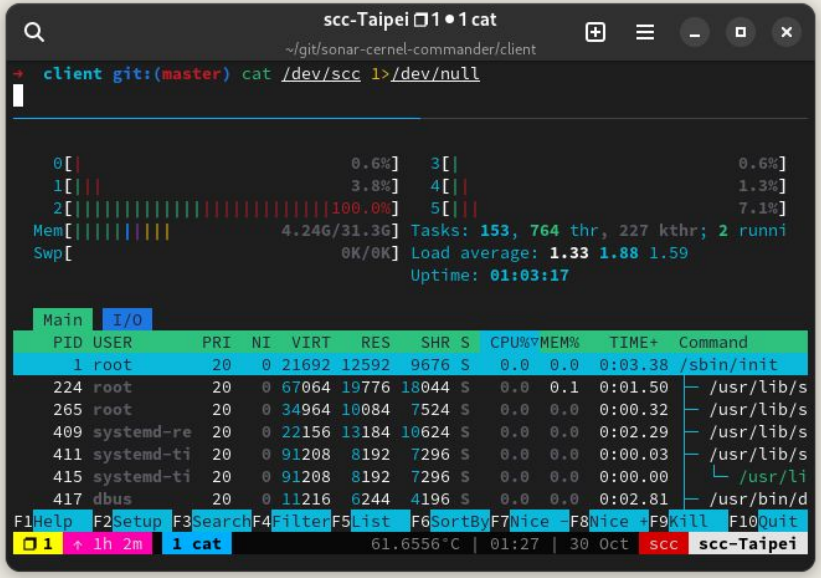
```
#define lock_completion(comp, lock) \
    do                              \
    {                               \
        if (mutex_trylock(lock))    \
            break;                  \
        wait_for_completion(comp);  \
    } while (1)
```

```
struct completion {
    unsigned int done;
    struct swait_queue_head wait;
};
```

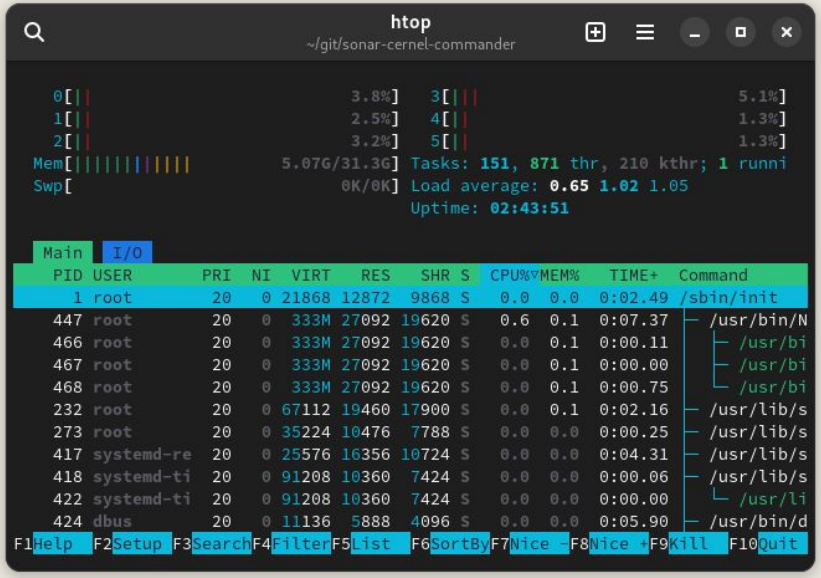




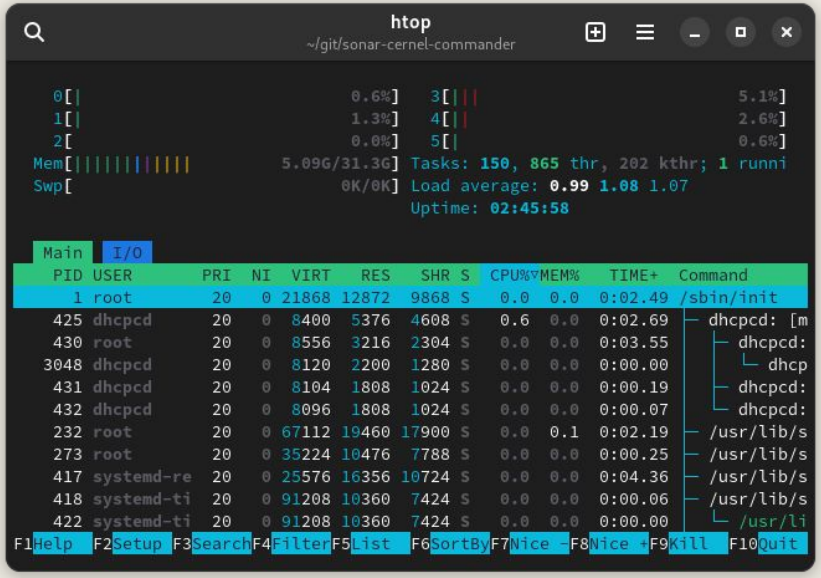

Hooked/enabled
**Not** read




Hooked/enabled
**Do** read



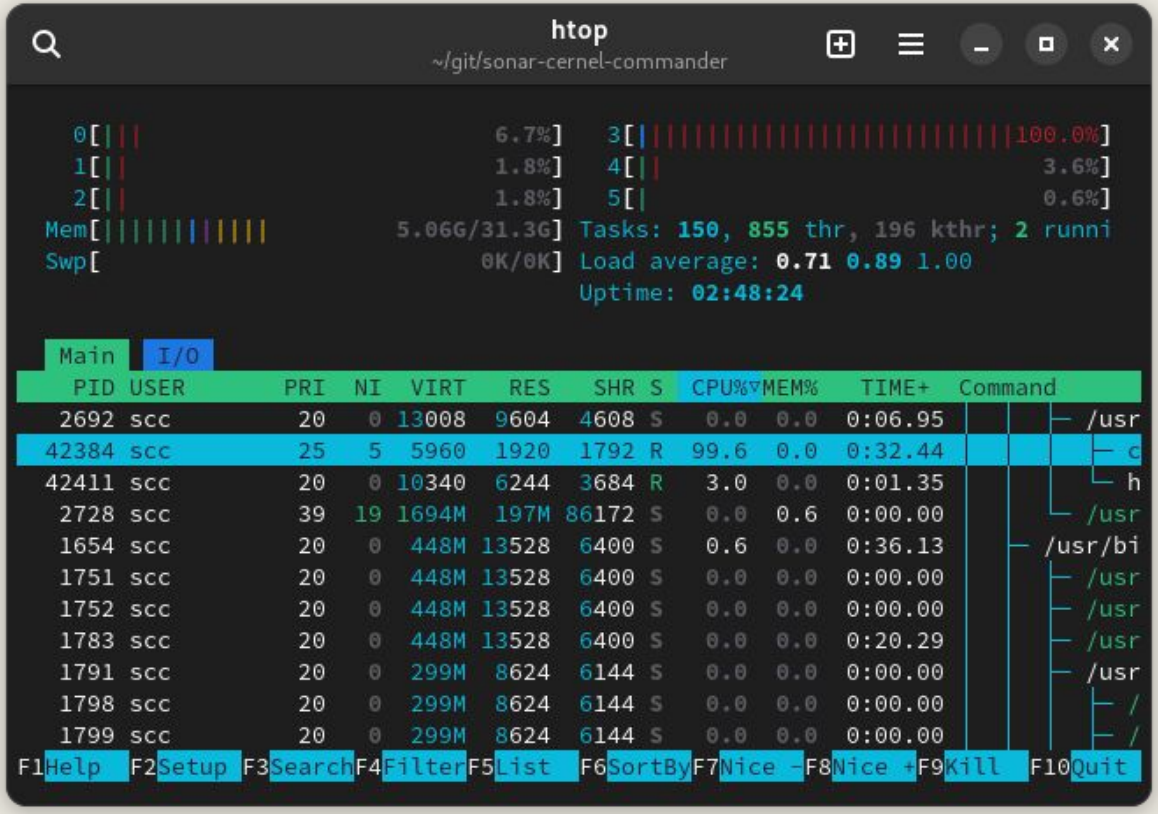

Hooked/**Not** enabled




**Not** Hooked

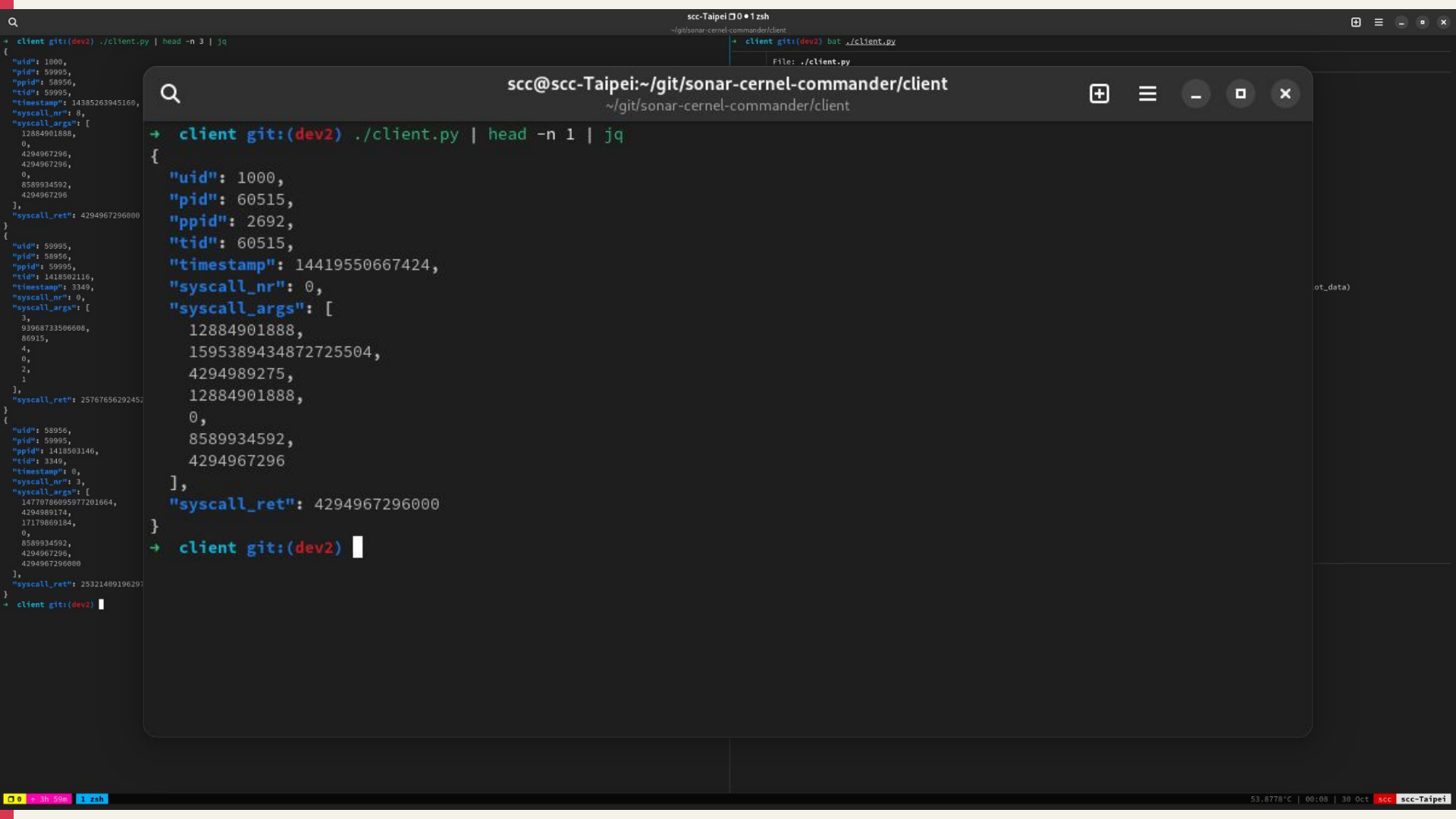
# cat /dev/urandom > /dev/null &

TEAMT5 杜浦數位安全



```
➜  client git:(dev2) ./client.py | head -n 3 | jq
{
  "uid": 1000,
  "pid": 59995,
  "ppid": 58956,
  "tid": 59995,
  "timestamp": 14385263945160,
  "syscall_nr": 8,
  "syscall_args": [
    12884901888,
    0,
    4294967296,
    4294967296,
    0,
    8589934592,
    4294967296
  ],
  "syscall_ret": 4294967296000
}
{
  "uid": 59995,
  "pid": 58956,
  "ppid": 59995,
  "tid": 1418502116,
  "timestamp": 3349,
  "syscall_nr": 0,
  "syscall_args": [
    3,
    93968733506608,
    86915,
    4,
    0,
    2,
    1
  ],
  "syscall_ret": 257676562924520
}
{
  "uid": 58956,
  "pid": 59995,
  "ppid": 1418503146,
  "tid": 3349,
  "timestamp": 0,
  "syscall_nr": 3,
  "syscall_args": [
    14770786095977201664,
    4294989174,
    17179869184,
    0,
    8589934592,
    4294967296,
    4294967296000
  ],
  "syscall_ret": 253214091962971
}
➜  client git:(dev2)
```

```
➜  client git:(dev2) bat ./client.py
File: ./client.py
#!/usr/bin/python

import struct
import json

def main():
    # Define the format string for unpacking the binary data
    event_format = "<IIIIQIQ6QQ"  # Use little-endian format for struct fields

    # Open the device file for reading
    with open('/dev/scc', 'rb') as file:
        while True:
            try:
                # Read a slot of data
                slot_data = file.read(struct.calcsize(event_format))

                if not slot_data:
                    break  # Exit loop if there's no more data

                # Unpack the binary data into variables
                uid, pid, ppid, tid, timestamp, syscall_nr, *syscall_args, syscall_ret = struct.unpack(event_format, slot_data)

                # Create a dictionary to store the event data
                event_data = {
                    "uid": uid,
                    "pid": pid,
                    "ppid": ppid,
                    "tid": tid,
                    "timestamp": timestamp,
                    "syscall_nr": syscall_nr,
                    "syscall_args": syscall_args,
                    "syscall_ret": syscall_ret
                }

                # Print the event data in JSON format
                print(json.dumps(event_data))

            except struct.error as e:
                print("Error parsing data:", e)
            except IOError: # Ignore IOError exceptions, e.g. broken pipe
                break

    # Close the file when done
    file.close()

if __name__ == '__main__':
    main()
➜  client git:(dev2)
```


scc-Taipei ❐ 0 ● 1 zsh
~/git/sonar-cernel-commander/client
scc@scc-Taipei:~/git/sonar-cernel-commander/client
~/git/sonar-cernel-commander/client
→ client git:(dev2) ./client.py | head -n 1 | jq
{
"uid": 1000,
"pid": 60515,
"ppid": 2692,
"tid": 60515,
"timestamp": 14419550667424,
"syscall_nr": 0,
"syscall_args": [
12884901888,
1595389434872725504,
4294989275,
12884901888,
0,
8589934592,
4294967296
],
"syscall_ret": 4294967296000
}
→ client git:(dev2)

# Demo

# Thank you!

https://github.com/25077667/sonar-cernel-commander

```
➜  tvm make; sudo insmod ./tvm.ko; sudo dmesg | tail -n 1; cat main.c| tail -n 16
make -C /lib/modules/6.5.9-arch2-1/build M=/home/scc/git/tvm modules
make[1]: Entering directory '/usr/lib/modules/6.5.9-arch2-1/build'
  CC [M]  /home/scc/git/tvm/main.o
  LD [M]  /home/scc/git/tvm/tvm.o
  MODPOST /home/scc/git/tvm/Module.symvers
  CC [M]  /home/scc/git/tvm/tvm.mod.o
  LD [M]  /home/scc/git/tvm/tvm.ko
  BTF [M] /home/scc/git/tvm/tvm.ko
make[1]: Leaving directory '/usr/lib/modules/6.5.9-arch2-1/build'
[ 1698.037478] Result: 4
{
    const char *code[] = {"PUSH", "5", "PUSH", "3", "SUB", "PUSH", "2", "MUL"};
    int instruction_count = sizeof(code) / sizeof(code[0]);
    execute(code, instruction_count);
    int result = pop();
    printk(KERN_INFO "Result: %d\n", result);
    return 0;
}

static void simple_module_exit(void)
{
    // Clean up if needed
}

module_init(simple_module_init);
module_exit(simple_module_exit);
➜  tvm
```

# License

TEAMT5
杜 浦 數 位 安 全

Source Code: Dual BSD/GPL

(If it needs to be a production, we need to rewrite the GPL parts.)

Slide: Follow pictures origin author