

Virtual Machine Homework 2

組員：

- B04703001 蔡明宏
- B04705001 陳約廷

1 - Armv8 Architecture

Exception model

We start from understanding the Armv8-A exception model, splitting into different levels of privilege. Level of privilege changes only when an exception occurs. Privilege levels are referred as exception levels.

- *EL0*: unprivileged
- *EL1*: OS kernel mode
- *EL2*: Hypervisor mode
- *EL3*: TrustZone® monitor mode

Uses exceptions to require for higher privileges. (like interrupt, page fault...)

- *EL0* → *EL1*: SVC (system call)
- *EL1* → *EL2*: HVC (hypervisor call)
- *EL2* → *EL3*: SMC (secure monitor call)

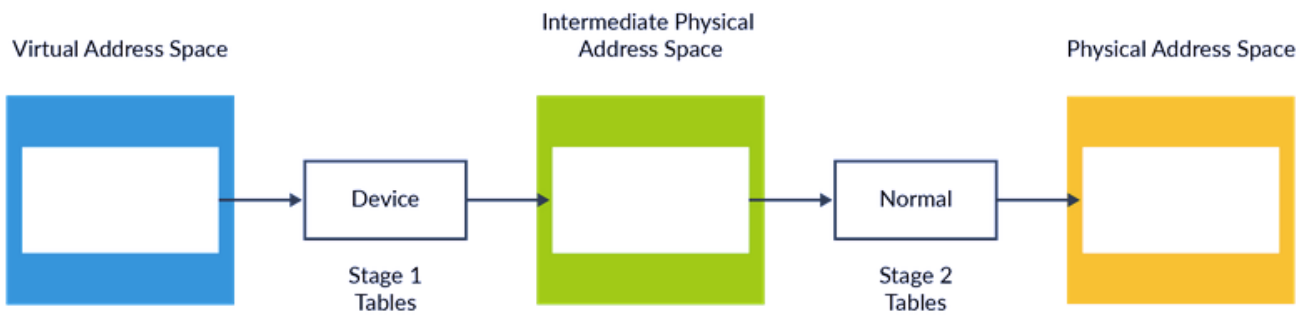
Memory Management

Armv8 manages memory in "**Memory Management Unit**" (MMU). A virtual address must be translated to a physical address before an memory access can take place.

Virtualization

This homework wants us to instrument the traps from guest to host in System VM. So we also need to understand the virtualization of Armv8-A.

Armv8 uses "**Stage 2 translation**" to allow hypervisor to control a view of memory in the VM.



Sensitive instruction

Guest needs a trap when sensitive instructions occur. Therefore we need to find the functions that calls it.

虛擬化指令(ARM)

問題指令(Problematic Instructions)

- Type I: 在user mode執行會產生未定義的指令異常
 - MCR、MRC: 需要依賴協處理器(coprocessor)
- Type II: 在user mode執行會沒有作用
 - MSR、MRS: 需要操作系統暫存器
- Type III: 在user mode執行會產生不可預測的行為
 - MOVN PC, LR: 返回指令，改變PC並跳回user mode，在user mode執行會產生不可預測的結果
- ARM 的敏感指令:
 - 存取協處理器: MRC / MCR / CDP / LDC / STC
 - 存取SIMD/VFP 系統暫存器: VMRS / VMSR
 - 進入TrustZone 安全狀態: SMC
 - 存取 Memory-Mapped I/O: Load/Store instructions from/into memory-mapped I/O locations
 - 直接存取CPSR: MRS / MSR / CPS / SRS / RFE / LDM (conditional execution) / DPSPC
 - 間接存取CPSR: LDRT / STRT – Load/Store Unprivileged ("As User")
 - 存取Banked Register: LDM / STM

In the picture we can see that the traps interacting with the processor are `MRC/MCR/CDP/LDC/STC` .

2 - Trace code

Start to trace into the call of traps.

First we can see that TA already setted the instrument variables of trap count inside structure of `vcpu` .

```

271▼ #ifdef CONFIG_VM_HW2
272    unsigned long cpu_trap_count;
273    unsigned long mem_trap_count;
274    unsigned long io_trap_count;
275▼ #endif

```

Next on we can see that inside `kvm_main.c` it, in TAG `KVM_RUN` it calls `kvm_arch_vcpu_ioctl_run`

```

1987    case KVM_RUN:
1988        r = -EINVAL;
1989        if (arg)
1990            goto out;
1991        r = kvm_arch_vcpu_ioctl_run(vcpu, vcpu->run);
1992        trace_kvm_userspace_exit(vcpu->run->exit_reason, r);
1993        break;

```

In `arm.c`, `kvm_arch_vcpu_ioctl_run` enters the guest, exits when `kvm_call_hyper`. Using the result from `kvm_call_hyper`, the `kvm` calls `handle_exit` to handle traps.

```

573    /* Enter the guest
574    */
575    trace_kvm_entry(*vcpu_pc(vcpu));
576    kvm_guest_enter();
577    vcpu->mode = IN_GUEST_MODE;
578
579    ret = kvm_call_hyp(__kvm_vcpu_run, vcpu);
580
581    vcpu->mode = OUTSIDE_GUEST_MODE;
582    vcpu->arch.last_pcpu = smp_processor_id();
583    kvm_guest_exit();
584    trace_kvm_exit(*vcpu_pc(vcpu));
585    /*
586     * We may have taken a host interrupt in HYP mode (ie
587     * while executing the guest). This interrupt is still
588     * pending, as we haven't serviced it yet!
589     *
590     * We're now back in SVC mode, with interrupts
591     * disabled. Enabling the interrupts now will have
592     * the effect of taking the interrupt again, in SVC
593     * mode this time.
594     */
595    local_irq_enable();
596
597    /*
598     * Back from guest
599     */
600    /*
601     */
602    kvm_timer_sync_hwstate(vcpu);
603    kvm_vgic_sync_hwstate(vcpu);
604
605    ret = handle_exit(vcpu, run, ret);
606    #ifdef CONFIG_VM_HW2
607        trace_kvm_vm_hw2(vcpu->cpu_trap_count, vcpu->mem_trap_count, vcpu->io_trap_count);
608    #endif

```

In the `handle_exit` function (in `handle_exit.c`), it calls `kvm_get_exit_handler` to get corresponding calls it needs.

```

98  /*
99  * Return > 0 to return to guest, < 0 on error, 0 (and set exit_reason) on
100  * proper exit to userspace.
101  */
102  int handle_exit(struct kvm_vcpu *vcpu, struct kvm_run *run,
103                 int exception_index)
104  {
105      exit_handle_fn exit_handler;
106
107      switch (exception_index) {
108      case ARM_EXCEPTION_IRQ:
109          return 1;
110      case ARM_EXCEPTION_TRAP:
111          /*
112           * See ARM ARM B1.14.1: "Hyp traps on instructions
113           * that fail their condition code check"
114           */
115          if (!kvm_condition_valid(vcpu)) {
116              kvm_skip_instr(vcpu, kvm_vcpu_trap_il_is32bit(vcpu));
117              return 1;
118          }
119
120          exit_handler = kvm_get_exit_handler(vcpu);
121
122          return exit_handler(vcpu, run);

```

Which are the following calls:

```

68  static exit_handle_fn arm_exit_handlers[] = {
69      [ESR_EL2_EC_WFI]      = kvm_handle_wfx,
70      [ESR_EL2_EC_CP15_32]  = kvm_handle_cp15_32,           // CPU
71      [ESR_EL2_EC_CP15_64]  = kvm_handle_cp15_64,           // CPU
72      [ESR_EL2_EC_CP14_MR]  = kvm_handle_cp14_access,       // CPU
73      [ESR_EL2_EC_CP14_LS]  = kvm_handle_cp14_load_store,   // CPU
74      [ESR_EL2_EC_CP14_64]  = kvm_handle_cp14_access,       // CPU
75      [ESR_EL2_EC_HVC32]    = handle_hvc,
76      [ESR_EL2_EC_SMC32]    = handle_smc,
77      [ESR_EL2_EC_HVC64]    = handle_hvc,
78      [ESR_EL2_EC_SMC64]    = handle_smc,
79      [ESR_EL2_EC_SYS64]    = kvm_handle_sys_reg,           // CPU
80      [ESR_EL2_EC_IABT]     = kvm_handle_guest_abort,       // IO / Mem
81      [ESR_EL2_EC_DABT]     = kvm_handle_guest_abort,       // IO / Mem
82  };

```

Also we can find there codes to lookup at the Armv8 manual:

```

206  /* Hyp Prefetch Fault Address Register (HPFAR/HDFAR) */
207  #define HPFAR_MASK (~0xFUL)
208
209  #define ESR_EL2_EC_UNKNOWN (0x00)
210  #define ESR_EL2_EC_WFI (0x01) // handle_wfx
211  #define ESR_EL2_EC_CP15_32 (0x03) // handle_cp15_32
212  #define ESR_EL2_EC_CP15_64 (0x04) // handle_cp15_64
213  #define ESR_EL2_EC_CP14_MR (0x05) // handle_cp14_access
214  #define ESR_EL2_EC_CP14_LS (0x06) // handle_cp14_load_store
215  #define ESR_EL2_EC_FP_ASIMD (0x07)
216  #define ESR_EL2_EC_CP10_ID (0x08)
217  #define ESR_EL2_EC_CP14_64 (0x0C) // handle_cp14_access
218  #define ESR_EL2_EC_ILL_ISS (0x0E)
219  #define ESR_EL2_EC_SVC32 (0x11)
220  #define ESR_EL2_EC_HVC32 (0x12) // handle_hvc
221  #define ESR_EL2_EC_SMC32 (0x13) // handle_smc
222  #define ESR_EL2_EC_SVC64 (0x15)
223  #define ESR_EL2_EC_HVC64 (0x16) // handle_hvc
224  #define ESR_EL2_EC_SMC64 (0x17) // handle_smc
225  #define ESR_EL2_EC_SYS64 (0x18)
226  #define ESR_EL2_EC_IABT (0x20) // handle_guest_abort
227  #define ESR_EL2_EC_IABT_HYP (0x21)
228  #define ESR_EL2_EC_PC_ALIGN (0x22)
229  #define ESR_EL2_EC_DABT (0x24) // handle_guest_abort
230  #define ESR_EL2_EC_DABT_HYP (0x25)
231  #define ESR_EL2_EC_SP_ALIGN (0x26)
232  #define ESR_EL2_EC_FP_EXC32 (0x28)
233  #define ESR_EL2_EC_FP_EXC64 (0x2C)
234  #define ESR_EL2_EC_ERRORR (0x2F)
235  #define ESR_EL2_EC_BREAKPT (0x30)
236  #define ESR_EL2_EC_BREAKPT_HYP (0x31)
237  #define ESR_EL2_EC_SOFTSTP (0x32)
238  #define ESR_EL2_EC_SOFTSTP_HYP (0x33)
239  #define ESR_EL2_EC_WATCHPT (0x34)
240  #define ESR_EL2_EC_WATCHPT_HYP (0x35)
241  #define ESR_EL2_EC_BKPT32 (0x38)
242  #define ESR_EL2_EC_VECTOR32 (0x3A)
243  #define ESR_EL2_EC_BRK64 (0x3C)

```

3 - Modified code

Modified

- `kernel_code/mmu.c` : from `linux-kvm-arm/arch/arm/kvm/mmu.c`
- `kernel_code/sys_reg.c` : from `linux-kvm-arm/arch/arm64/kvm/sys_reg.c`

IO trap / Memory trap

We look inside the `handle_guest_abort` function (inside `mmu.c`). We can see that it deals with traps relating to the second stage translation table entry.

```

737 /**
738  * kvm_handle_guest_abort - handles all 2nd stage aborts
739  * @vcpu: the VCPU pointer
740  * @run: the kvm_run structure
741  *
742  * Any abort that gets to the host is almost guaranteed to be caused by a
743  * missing second stage translation table entry, which can mean that either the
744  * guest simply needs more memory and we must allocate an appropriate page or it
745  * can mean that the guest tried to access I/O memory, which is emulated by user
746  * space. The distinction is based on the IPA causing the fault and whether this
747  * memory region has been registered as standard RAM by user space.
748  */

```

The key function we see is `kvm_is_visible_gfn`, where GFN stands for "**Guest Frame Number**".

- When `kvm_is_visible_gfn` return `false`, the function tries to search for IO memory. (by the error log in the code) **We add `io_trap_count++` right here.**
- When `kvm_is_visible_gfn` returns `true`, it fetches for the corresponding memory slot. (fetched by `gfn_to_memslot`) **We add `mem_trap_count++` right here.**

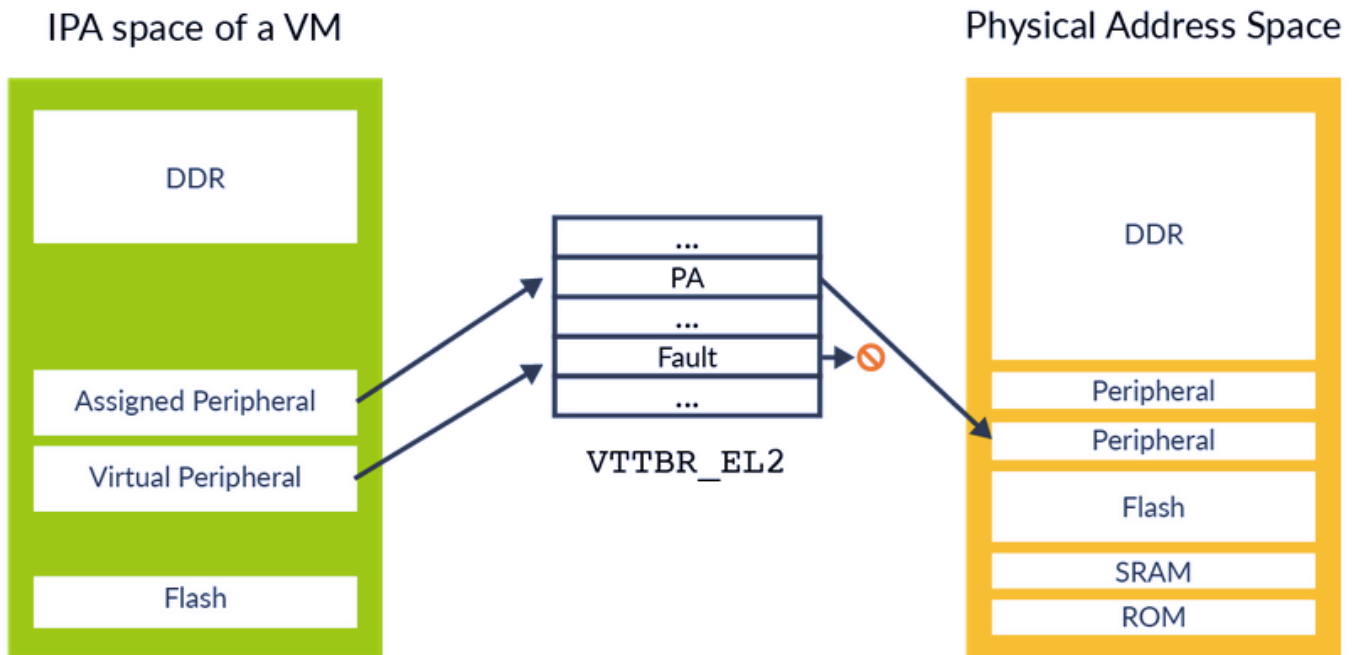
```

772     idx = srcu_read_lock(&vcpu->kvm->srcu);
773
774     gfn = fault_ipa >> PAGE_SHIFT;
775     if (!kvm_is_visible_gfn(vcpu->kvm, gfn)) {
776         if (is_iabt) {
777             /* Prefetch Abort on I/O address */
778             kvm_inject_pabt(vcpu, kvm_vcpu_get_hfar(vcpu));
779             ret = 1;
780             goto out_unlock;
781         }
782         vcpu->io_trap_count++;
783         if (fault_status != FSC_FAULT) {
784             kvm_err("Unsupported fault status on io memory: %#lx\n",
785                 fault_status);
786             ret = -EFAULT;
787             goto out_unlock;
788         }
789
790         /*
791          * The IPA is reported as [MAX:12], so we need to
792          * complement it with the bottom 12 bits from the
793          * faulting VA. This is always 12 bits, irrespective
794          * of the page size.
795          */
796         fault_ipa |= kvm_vcpu_get_hfar(vcpu) & ((1 << 12) - 1);
797         ret = io_mem_abort(vcpu, run, fault_ipa);
798         goto out_unlock;
799     }
800     vcpu->mem_trap_count++;
801     memslot = gfn_to_memslot(vcpu->kvm, gfn);

```

Translating from IPA (Intermediate Physical Address) to PA (Physical Address), kvm would need to check if the memory address shall be visible to the guest or not. This is what the function

`kvm_is_visible_gfn` is doing.



CPU trap

The Armv8-A architecture has a family of exception-generating instructions: **SVC**, **HVC**, and **SMC**. These instructions are different from a simple invalid instruction, because they target different exception levels and are treated differently when prioritizing exceptions. These instructions are used to implement system call interfaces to allow less privileged code to request services from more privileged code.

from arm.developer.com

These instructions are synchronous exceptions that does not involves in Hyp Mode.

CPU traps are those of sensitive instructions that interacts with the coprocessor (mentioned [above](#)) - `MRC/MCR/CDP/LDC/STC` .

We check it out in `sys_reg.c` , and they do deal with these sensitive instrucionts. So we add `cpu_trap_count++` in each of these functions.


```

382 int kvm_handle_cp14_load_store(struct kvm_vcpu *vcpu, struct kvm_run *run)
383 {
384     vcpu->cpu_trap_count++;
385     kvm_inject_undefined(vcpu);
386     return 1;
387 }
388
389 int kvm_handle_cp14_access(struct kvm_vcpu *vcpu, struct kvm_run *run)
390 {
391     vcpu->cpu_trap_count++;
392     kvm_inject_undefined(vcpu);
393     return 1;
394 }
395

```

```

431 /**
432  * kvm_handle_cp15_64 -- handles a mrrc/mcrr trap on a guest CP15 access
433  * @vcpu: The VCPU pointer
434  * @run: The kvm_run struct
435  */
436 int kvm_handle_cp15_64(struct kvm_vcpu *vcpu, struct kvm_run *run)

```

```

476 /**
477  * kvm_handle_cp15_32 -- handles a mrc/mcr trap on a guest CP15 access
478  * @vcpu: The VCPU pointer
479  * @run: The kvm_run struct
480  */
481 int kvm_handle_cp15_32(struct kvm_vcpu *vcpu, struct kvm_run *run)

```

```

546 /**
547  * kvm_handle_sys_reg -- handles a mrs/msr trap on a guest sys_reg access
548  * @vcpu: The VCPU pointer
549  * @run: The kvm_run struct
550  */
551 int kvm_handle_sys_reg(struct kvm_vcpu *vcpu, struct kvm_run *run)

```

4 - Experiments

This section are of experiments that the homework requires.


```
FVP terminal_0
[000] ...1 583.461870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.471870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.481870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.491870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.501870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.511870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.521870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.531870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.541870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.551870: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.561879: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636
[000] ...1 583.571910: kvm_vm_hw2: CPU trap 17414, MEM trap 23228, IO trap 636

- trace 40013/40013 100%
```

The above picture is the trace right after booting the guest.

- CPU trap comes from calls of interaction that requires the coprocessor (cache maintenance, TLB maintenance...)
- MEM trap comes from for address needed for daemons when boot
- IO trap comes from the output on boot logs on to the screen

```
FVP terminal_0
[000] ...1 713.961870: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 713.971871: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 713.981870: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 713.991870: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.001871: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.011870: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.021870: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.031871: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.041881: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.052003: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.061921: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728
[000] ...1 714.072028: kvm_vm_hw2: CPU trap 17414, MEM trap 23303, IO trap 728

- trace 39982/39982 100%
```

The above picture is the trace right after executing `home/root/mem` .

- CPU traps didn't occur
- MEM traps increase because `./mem` calls for memory allocation
- IO trap slightly increases due to typing on the guest screen to execute `./mem`

```

FVP terminal_0
[000] ...1 805,851870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,861870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,871870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,881870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,891870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,901870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,911870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,921870: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,931879: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,941910: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,951921: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952
[000] ...1 805,962024: kvm_vm_hw2: CPU trap 17414, MEM trap 23306, IO trap 952

- trace 39950/39950 100%

```

The above picture is the trace right after executing `home/root/io`.

- CPU traps didn't occur
- MEM trap slightly increases because the IPA may not contain everything needed when executing and require some fetch from PA.
- IO trap increases because of the IO calls in `home/root/io`

5 - Other Experiments

divide by zero (`test_program/divide.c`)

Unlike gcc in linux, aarch64 does not call for a floating point error when divided-by-zero occurs.

```
FVP terminal_0
blk-mq: CPU -> queue map
CPU 0 -> Queue 0
vda: unknown partition table
TCP: cubic registered
NET: Registered protocol family 17
9pnet: Installing 9P2000 support
Key type dns_resolver registered
regulator-dummy: disabling
EXT3-fs (vda): mounted filesystem with writeback data mode
kjournald starting. Commit interval 5 seconds
VFS: Mounted root (ext3 filesystem) readonly on device 254:0.
devtmpfs: mounted
Freeing unused kernel memory: 240K (fffffc00063d000 - fffffc000679000)
sh: cannot set terminal process group (-1): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.2# cd /home/root/
sh-4.2# ls
access    bomb      divide    fork      io        mem
access.c  bomb.c    divide.c  fork.c    io.c      mem.c
sh-4.2# ./divide
a = 50
b = 0
c = a/b = 0
sh-4.2# ./fork
i am mama
i am child
sh-4.2#
```

access bad memory (test_program/access.c)

Accessing bad memory will create an error and MEM trap will increase.

```
FVP terminal_0
c = a/b = 0
sh-4.2# ./access
access[315]: unhandled level 2 translation fault (11) at 0x00000000, esr 0x92000
046
pgd = fffffffc01afc3000
[00000000] *pgd=000000009af22003, *pmd=0000000000000000

CPU: 0 PID: 315 Comm: access Not tainted 3.14.0+ #1
task: fffffffc01ad5bd80 ti: fffffffc01af28000 task.ti: fffffffc01af28000
PC is at 0x400580
LR is at 0x7f96fba988
pc : [<0000000000400580>] lr : [<0000007f96fba988>] pstate: 60000000
sp : 0000007fea5a1a60
x29: 0000007fea5a1a70 x28: 0000000000000000
x27: 0000000000000000 x26: 0000000000000000
x25: 0000000000000000 x24: 0000000000000000
x23: 0000000000000000 x22: 0000000000000000
x21: 0000000000400420 x20: 0000000000000000
x19: 0000000000000000 x18: 0000007fea5a1930
x17: 0000000000411000 x16: 0000007f96fba898
x15: 0000007f97109028 x14: 0000000000000040
x13: 0000000000000090 x12: 0000000000000030
x11: 000000000002b028 x10: 0000007f96f9dba8
x9 : 0000007f970f69b8 x8 : 0000000000000087
x7 : adbaabff8c8c9a9c x6 : 0000007f970d9aa8
x5 : 0000007f970d9aa8 x4 : 0000000000000000
x3 : 0000000000400570 x2 : 0000007fea5a1bc8
x1 : 0000000000000030 x0 : 0000000000000000
sh-4.2#
```

fork (test_program/fork.c)

A single fork can be successfully executed.

```
FVP terminal_0

CPU: 0 PID: 315 Comm: access Not tainted 3.14.0+ #1
task: ffffffff01ad5bd80 ti: ffffffff01af28000 task.ti: ffffffff01af28000
PC is at 0x400580
LR is at 0x7f96fba988
pc : [<00000000000400580>] lr : [<00000007f96fba988>] pstate: 60000000
sp : 00000007fea5a1a60
x29: 00000007fea5a1a70 x28: 0000000000000000
x27: 0000000000000000 x26: 0000000000000000
x25: 0000000000000000 x24: 0000000000000000
x23: 0000000000000000 x22: 0000000000000000
x21: 00000000000400420 x20: 0000000000000000
x19: 0000000000000000 x18: 00000007fea5a1930
x17: 00000000000411000 x16: 00000007f96fba898
x15: 00000007f97109028 x14: 0000000000000040
x13: 0000000000000090 x12: 0000000000000030
x11: 0000000000002b028 x10: 00000007f96f9dba8
x9 : 00000007f970f69b8 x8 : 0000000000000087
x7 : adbaabff8c8c9a9c x6 : 00000007f970d9aa8
x5 : 00000007f970d9aa8 x4 : 0000000000000000
x3 : 00000000000400570 x2 : 00000007fea5a1bc8
x1 : 0000000000000030 x0 : 0000000000000000

sh-4.2# ./fork
i am mama
i am child
sh-4.2#
```

fork bomb (`test_program/bomb.c`)

The hypervisor didn't add limit the numbers of process the guest can create. Simple fork bomb program gradually paralyzes the host too ~ QAQ Making it really slow and have to be reboot.

stack overflow (`test_program/stack.c`)

Stack overflow also creates memory trap.

```

FVP terminal_0
blk-mq: CPU -> queue map
CPU 0 -> Queue 0
vda: unknown partition table
TCP: cubic registered
NET: Registered protocol family 17
9pnet: Installing 9P2000 support
Key type dns_resolver registered
regulator-dummy: disabling
EXT3-fs (vda): mounted filesystem with writeback data mode
kjournald starting. Commit interval 5 seconds
VFS: Mounted root (ext3 filesystem) readonly on device 254:0.
devtmpfs: mounted
Freeing unused kernel memory: 240K (fffffc00063d000 - ffffffc000679000)
sh: cannot set terminal process group (-1): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.2# cd /home/root/
sh-4.2# ls
access      bomb        divide      fork        io          mem         stack
access.c    bomb.c      divide.c    fork.c      io.c        mem.c       stack.c
sh-4.2# ./stack
stack[314]: unhandled level 3 translation fault (11) at 0x7fd4242ff0, esr 0x9200
0047
pgd = ffffffc01afd1000
[7fd4242ff0] *pgd=000000009afd7003, *pmd=000000009afd4003, *pte=0000000000000000

CPU: 0 PID: 314 Comm: stack Not tainted 3.14.0+ #1
task: ffffffc01ad5bd80 ti: ffffffc01ae84000 task.ti: ffffffc01ae84000
PC is at 0x400570
LR is at 0x40057c
pc : [<0000000000400570>] lr : [<000000000040057c>] pstate: 60000000
sp : 0000007fd4243000
x29: 0000007fd4243000 x28: 0000000000000000
x27: 0000000000000000 x26: 0000000000000000
x25: 0000000000000000 x24: 0000000000000000
x23: 0000000000000000 x22: 0000000000000000
x21: 0000000000400420 x20: 0000000000000000
x19: 0000000000000000 x18: 0000007fd4a424a0
x17: 0000000000411000 x16: 0000007f7ab9e898
x15: 0000007f7aced028 x14: 0000000000000040
x13: 0000000000000090 x12: 0000000000000030
x11: 00000000002b028 x10: 0000007f7ab81ba8
x9 : 0000007f7acda9b8 x8 : 0000000000000087
x7 : abff949c9e8b8cd0 x6 : 0000007f7acbd8a8
x5 : 0000007f7acbd8a8 x4 : 0000000000000000
x3 : 0000000000400588 x2 : 0000007fd4a42738
x1 : 0000007fd4a42728 x0 : 0000000000000001

sh-4.2#

```

6 - Reference

- developer.arm.com - Exception Model
- developer.arm.com - Memory Management
- developer.arm.com - Armv8-A Virtualization
- 成大資工wiki - XVisor