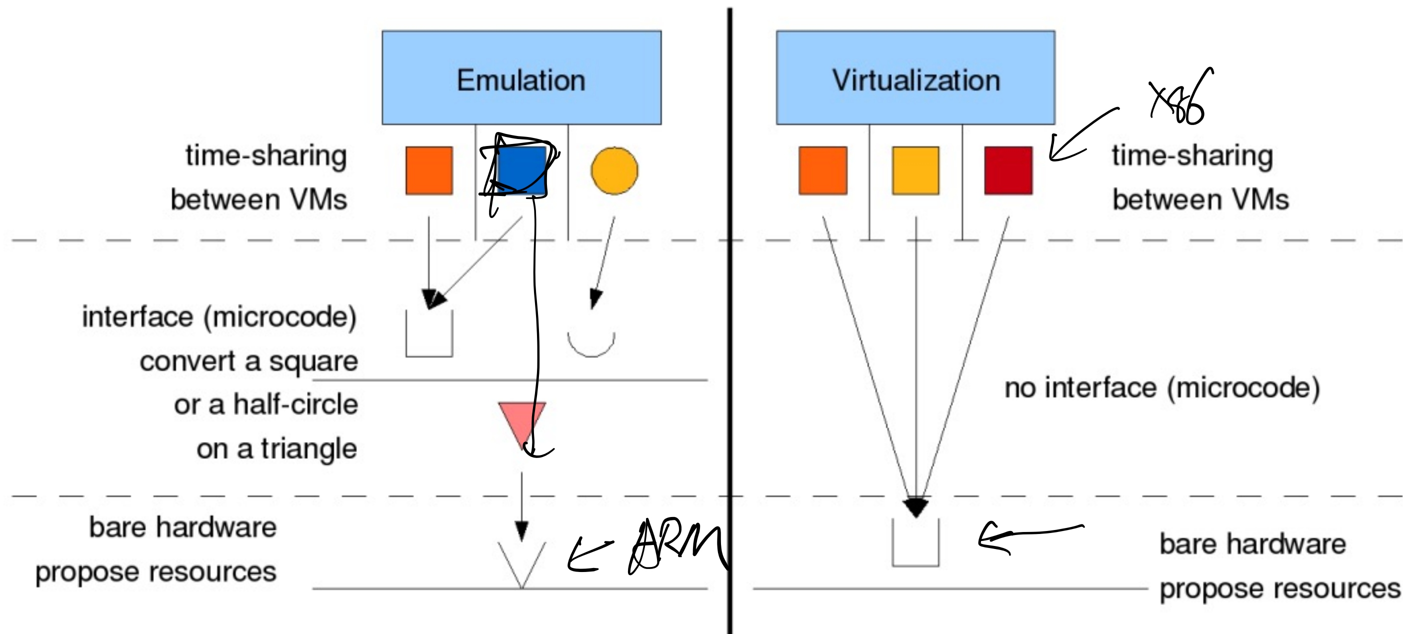


Figure 4: Virtualization vs Emulation

Gallard, Jérôme, et al. *Refinement Proposal of the Goldberg's Theory*. Diss. INRIA, 2008.

Register	Usage	Preserved across function calls
%rax	temporary register; with variable arguments passes information about the number of vector registers used; 1st return register	No
%rbx	callee-saved register	Yes
%rcx	used to pass 4th integer argument to functions	No
%rdx	used to pass 3rd argument to functions; 2nd return register	No
%rsp	stack pointer	Yes
%rbp	callee-saved register; optionally used as frame pointer	Yes
%rsi	used to pass 2nd argument to functions	No
%rdi	used to pass 1st argument to functions	No
%r8	used to pass 5th argument to functions	No
%r9	used to pass 6th argument to functions	No
%r10	temporary register, used for passing a function's static chain pointer	No
%r11	temporary register	No
%r12-r14	callee-saved registers	Yes
%r15	callee-saved register; optionally used as GOT base pointer	Yes

https://github.com/hjl-tools/x86-psABI/wiki/x86-64-psABI-1.0.pdf

Figure 3.4

master / linux / arch / x86 / entry / syscalls / syscall_64.tbl

Go to file

arndb compat: remove some compat entry points ...

Latest commit 59ab844 9 days ago

History

19 contributors +7

417 lines (416 sloc) | 14.4 KB

Raw Blame

```
1  #
2  # 64-bit system call numbers and entry vectors
3  #
4  # The format is:
5  # <number> <abi> <name> <entry point>
6  #
7  # The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
8  #
9  # The abi is "common", "64" or "x32" for this file.
10 #
11 0       common  read                    sys_read
12 1       common  write                   sys_write
13 2       common  open                    sys_open
14 3       common  close                   sys_close
15 4       common  stat                    sys_newstat
16 5       common  fstat                   sys_newfstat
17 6       common  lstat                   sys_newlstat
18 7       common  poll                    sys_poll
19 8       common  lseek                   sys_lseek
20 9       common  mmap                    sys_mmap
21 10      common  mprotect                sys_mprotect
22 11      common  munmap                  sys_munmap
23 12      common  brk                     sys_brk
24 13      64      rt_sigaction            sys_rt_sigaction
```

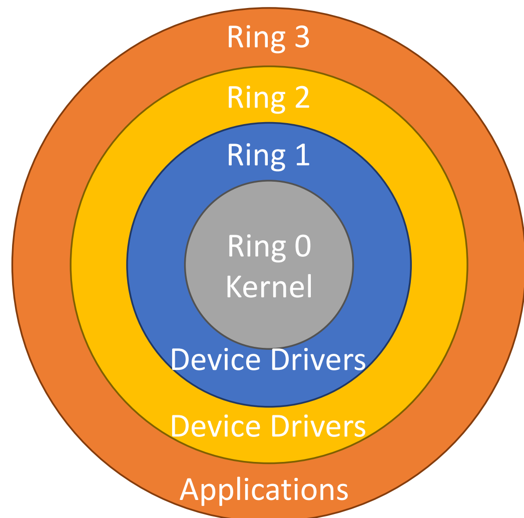
Visited 09/17/2021. There are ~400 system calls.

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

Protected Mode

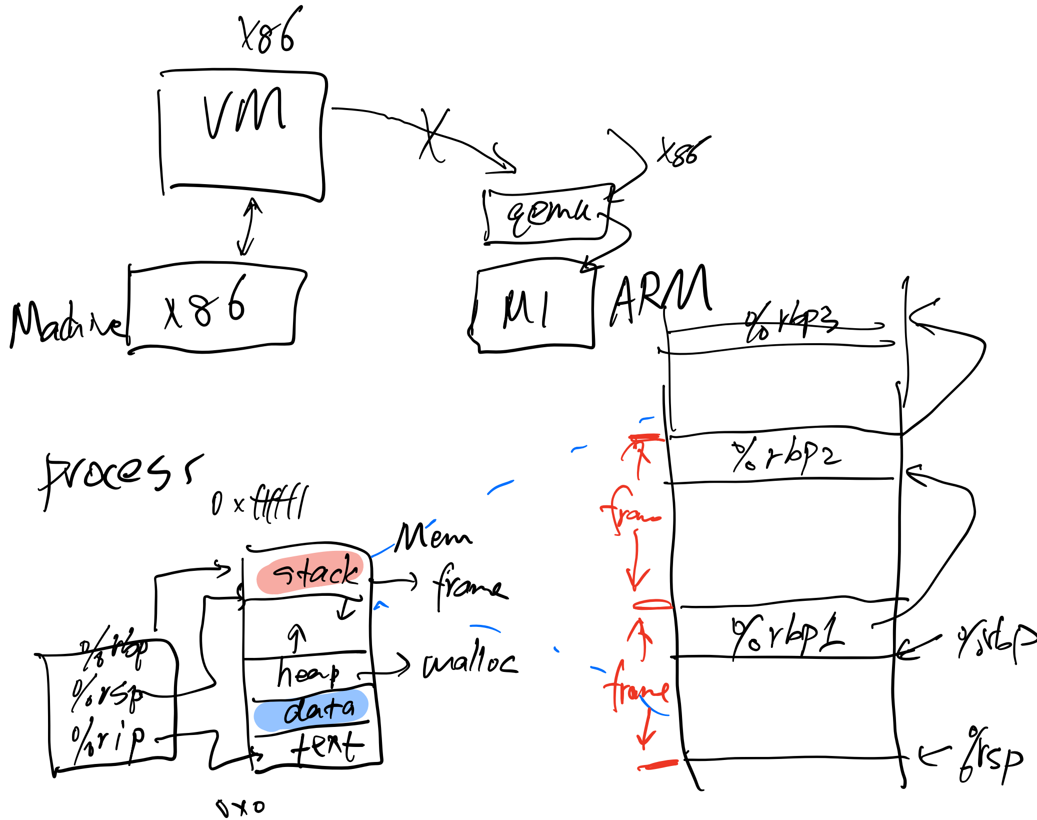
- Most modern CPUs support protected mode
- x86 CPUs support three rings with different privileges
  - Ring 0: OS kernel
  - ~~Ring 1, 2: device drivers~~
  - Ring 3: userland
- Most OSes only use rings 0 and 3

- Privileged instructions?
  - https://sites.google.com/site/masumzh/articles/x86-architecture-basics/x86-architecture-basics



Borrowed from Ji-Yong Shin, CS5600 21spring

1. Last time
2. Stack frames, continued
3. Syscall intro
4. Process/OS control transfers
5. Git and Lab grading
6. Process birth
7. shell preliminary



mov

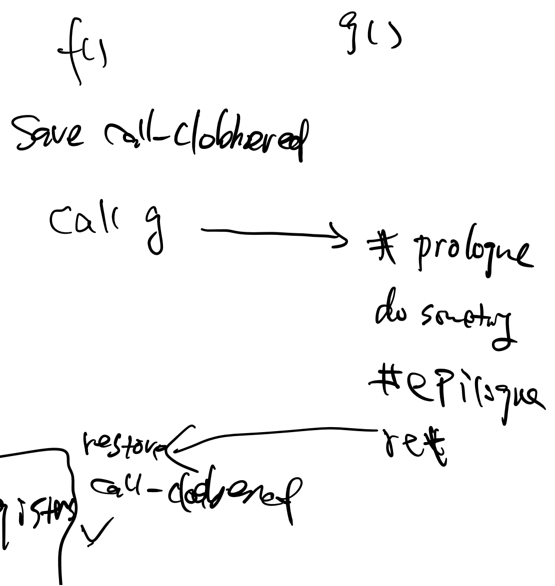
push/pop → %rsp

call/ret → %rip

Calling Conventions

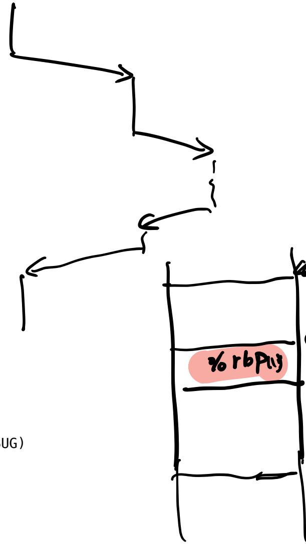
① arg: %rdi, %rsi ...
ret: %rax

② Call-clobbered/preserved Registers



```
-------------[example.c]---------------

1    /* CS5600 -- handout w01a
2     * compile and run this code with:
3     * $ gcc -g -Wall -o example example.c
4     * $ ./example
5     *
6     * examine its assembly with:
7     * $ gcc -O0 -S example.c
8     * $ [editor] example.s
9     */
10
11    #include <stdio.h>
12    #include <stdint.h>
13
14    uint64_t f(uint64_t* ptr);
15    uint64_t g(uint64_t a);
16    uint64_t* q;
17
18    int main(void)
19    {
20        uint64_t x = 0;
21        uint64_t arg = 8;
22
23        x = f(&arg);
24
25        printf("x: %lu\n", x);
26        printf("dereference q: %lu\n", *q);
27
28        return 0;
29    }
30
31    uint64_t f(uint64_t* ptr)
32    {
33        uint64_t x = 0;
34        x = g(*ptr);
35        return x + 1;
36    }
37
38    uint64_t g(uint64_t a)
39    {
40        uint64_t x = 2*a;
41        q = &x; // <-- THIS IS AN ERROR (AKA BUG)
42        return x;
43    }
```




```
-------------[as.txt]--------------

1   2. A look at the assembly...
2
3   To see the assembly code that the C compiler (gcc) produces:
4   $ gcc -O0 -S example.c
5   (then look at example.s.)
6   NOTE: what we show below is not exactly what gcc produces. We have
7   simplified, omitted, and modified certain things.
8
9   main:
10      pushq %rbp             # prologue: store caller's frame pointer
11      movq %rsp, %rbp        # prologue: set frame pointer for new frame
12
13      subq $16, %rsp         # make stack space
14
15      movq $0, -8(%rbp)      # x = 0 (x lives at address rbp - 8)
16      movq $8, -16(%rbp)     # arg = 8 (arg lives at address rbp - 16)
17
18      leaq -16(%rbp), %rdi   # load the address of (rbp-16) into %rdi
19                             # this implements "get ready to pass (&arg)
20                             # to f"
21
22      call f                 # invoke f
23
24      movq %rax, -8(%rbp)    # x = (return value of f)
25
26      # eliding the rest of main()
27
28   f:
29       pushq %rbp            # prologue: store caller's frame pointer
30       movq %rsp, %rbp       # prologue: set frame pointer for new frame
31
32       subq $32, %rsp        # make stack space
33       movq %rdi, -24(%rbp)  # Move ptr to the stack
34                             # (ptr now lives at rbp - 24)
35       movq $0, -8(%rbp)     # x = 0 (x's address is rbp - 8)
36
37       movq -24(%rbp), %r8   # move 'ptr' to %r8
38       movq (%r8), %r9       # dereference 'ptr' and save value to %r9
39       movq %r9, %rdi        # Move the value of *ptr to rdi,
40                             # so we can call g
41
42       call g                # invoke g
43
44       movq %rax, -8(%rbp)   # x = (return value of g)
45       movq -8(%rbp), %r10   # compute x + 1, part I
46       addq $1, %r10         # compute x + 1, part II
47       movq %r10, %rax       # Get ready to return x + 1
48
49       movq %rbp, %rsp       # epilogue: undo stack frame
50       popq %rbp             # epilogue: restore frame pointer from caller
51       ret                   # return
52
53   g:
54       pushq %rbp            # prologue: store caller's frame pointer
55       movq %rsp, %rbp       # prologue: set frame pointer for new frame
56
57       ....
58
59       movq %rbp, %rsp       # epilogue: undo stack frame
60       popq %rbp             # epilogue: restore frame pointer from caller
61       ret                   # return
```

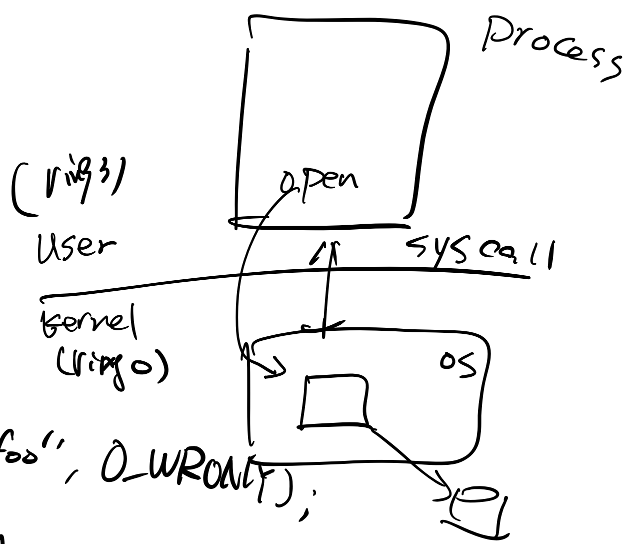
```c
int a = 1;
int* ptr = &a;
```

@ → 0x10086 [1]

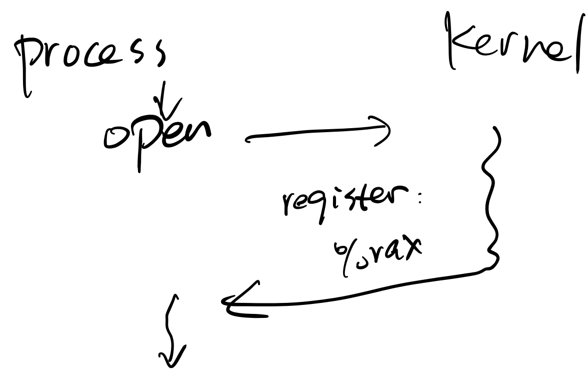
ptr → [0x10086]

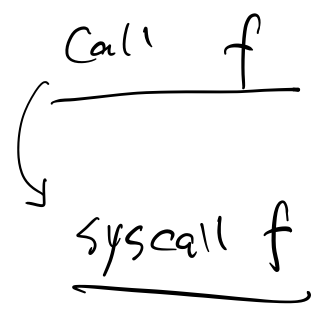
a → local
→ global
→ func

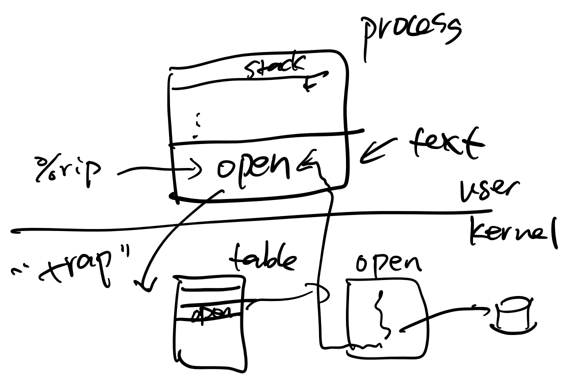
Syscall



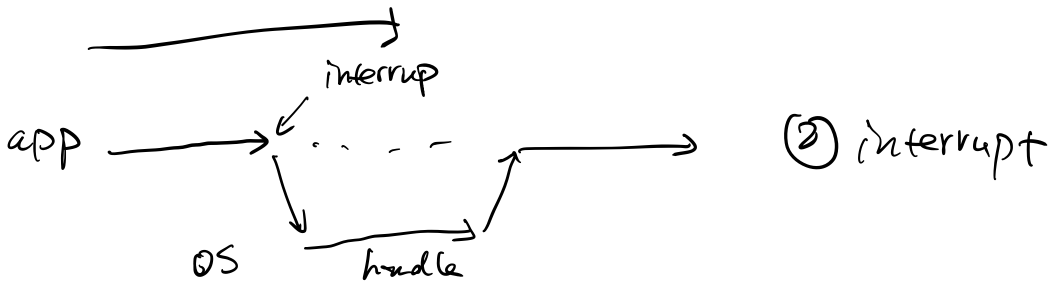

```c
int fd = open("/tmp/foo", O_WRONLY);
    read(fd, ...)
    write(fd, ...)
```

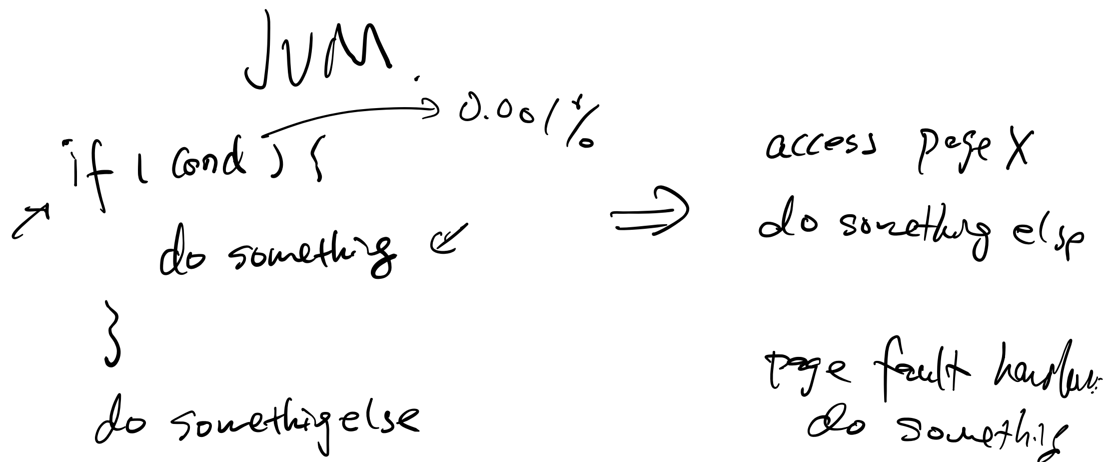


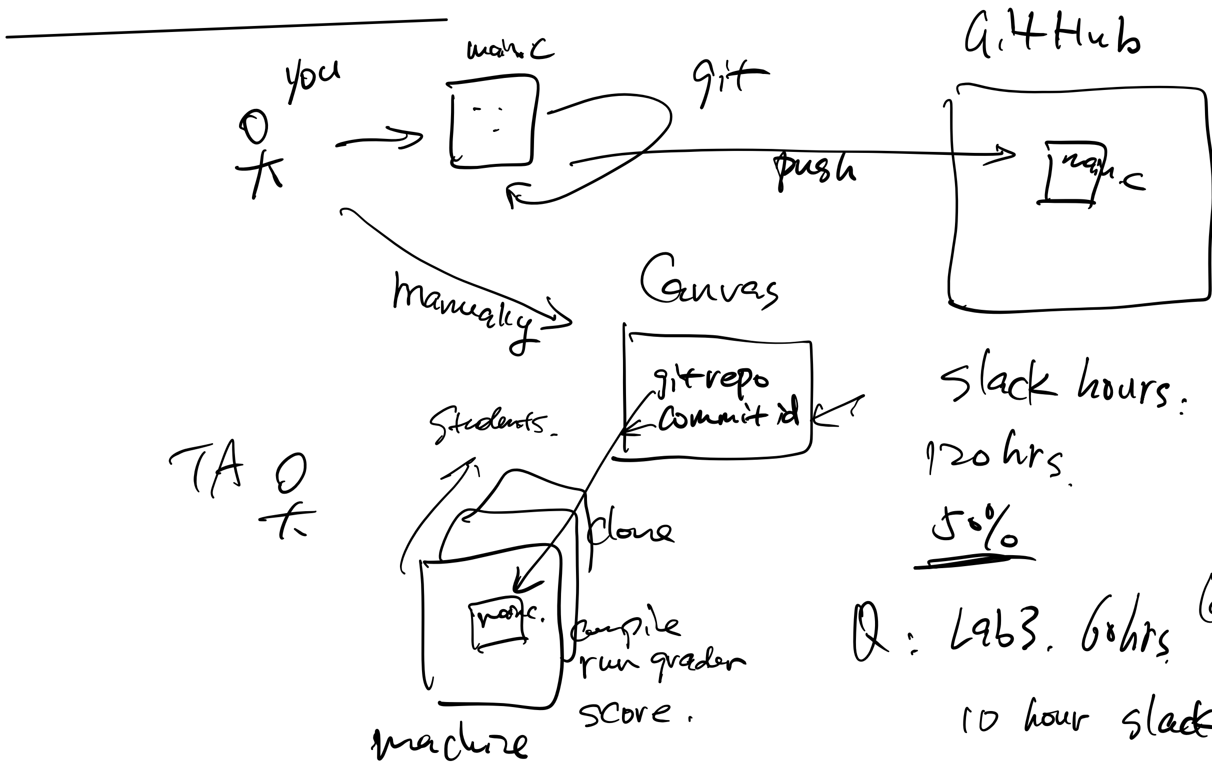




① Syscall


interrup
app
OS
handle
② interrupt


③ exception
JVM.
0.001%
if (cond) {
do something
}
do somethingelse
access page X
do something else
page fault handler
do something


you
main.c
git
push
GitHub
main.c
manually
Canvas
git repo
commit id
students
TA
clone
main.c
compile
run grader
score.
machine
slack hours:
120 hrs.
50%
Q: Lab3. 68hrs late.
10 hour slack?