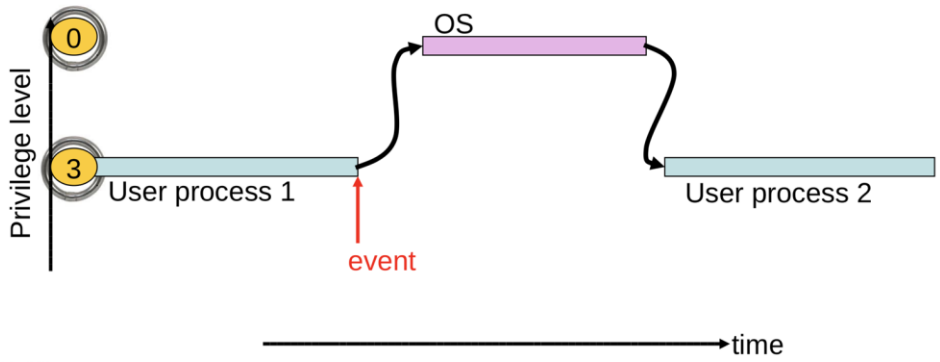
پروژه دوم آزمایشگاه سیستم‌عامل

System Calls and Processes

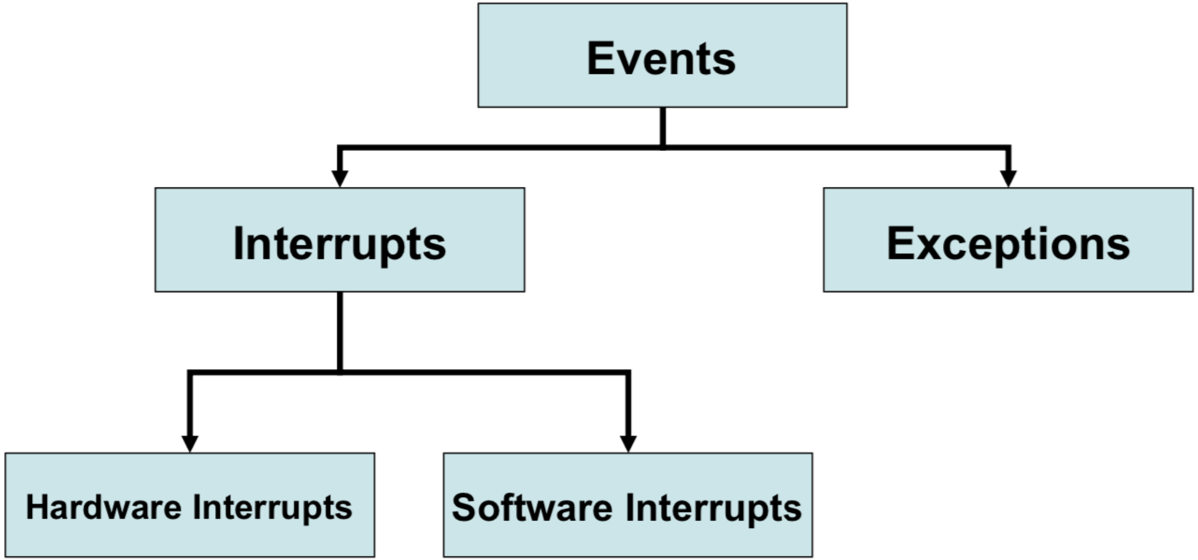
Sadaf Sadeghian - Aylin Jamali

Why event driven design?

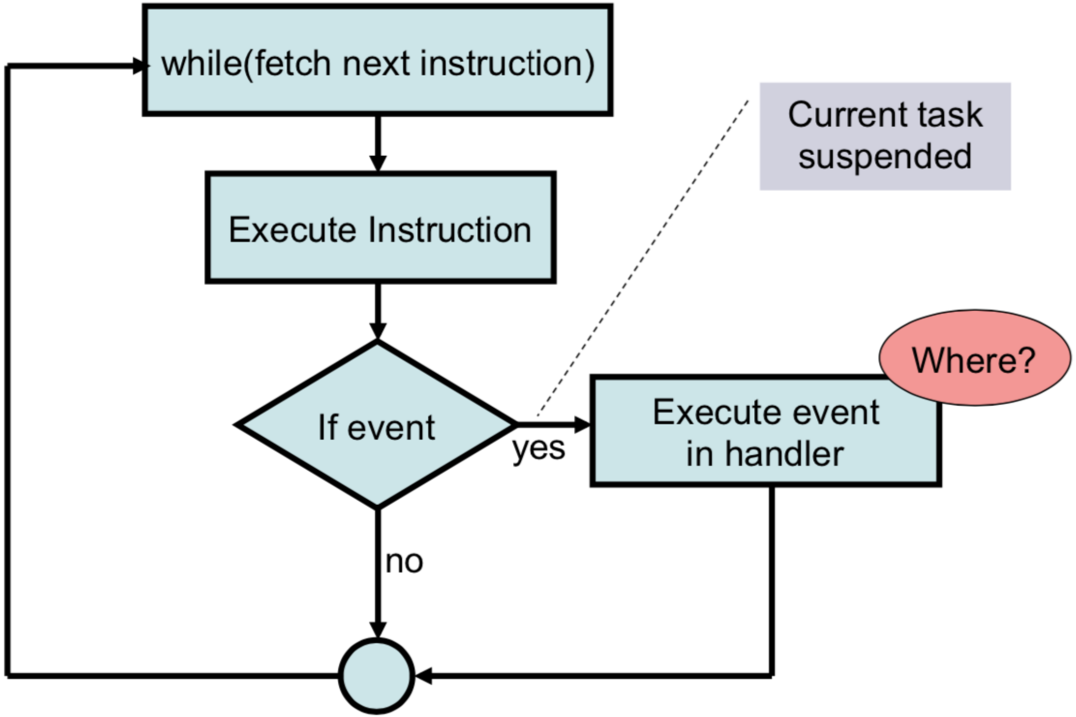
- OS cannot trust user processes
  - User processes may be buggy or malicious
  - User process crash should not affect OS
- OS needs to guarantee fairness to all user processes
  - One process cannot 'hog' CPU time
  - Timer interrupts



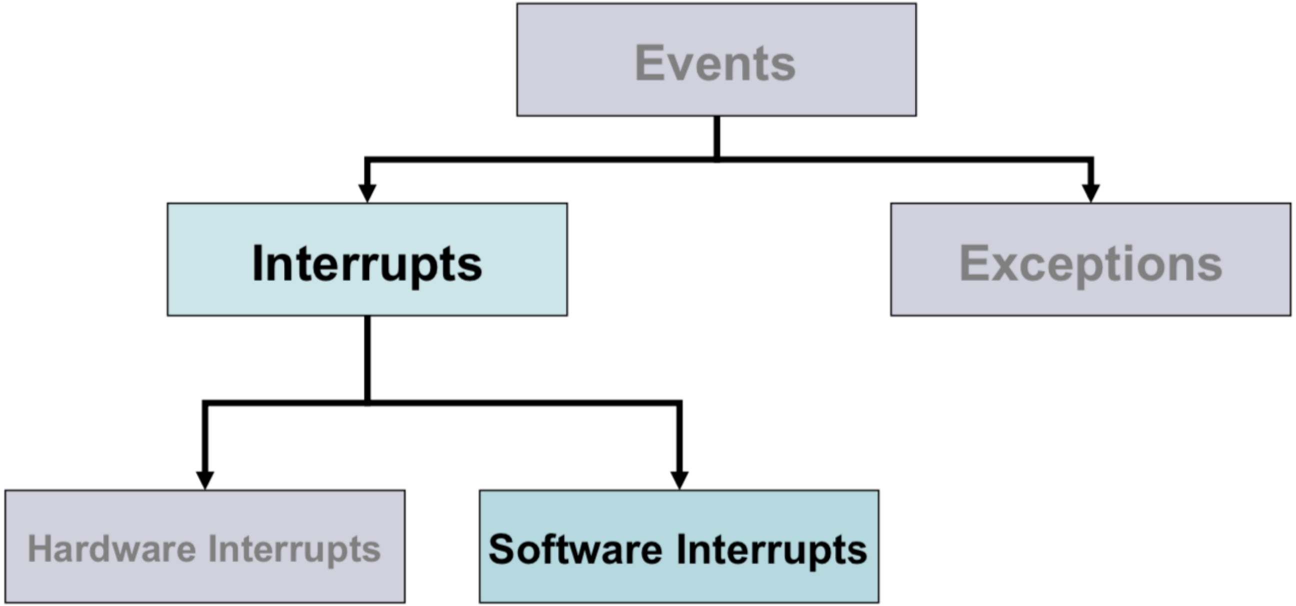
Event Types


Events
Interrupts
Exceptions
Hardware Interrupts
Software Interrupts

Event View of CPU

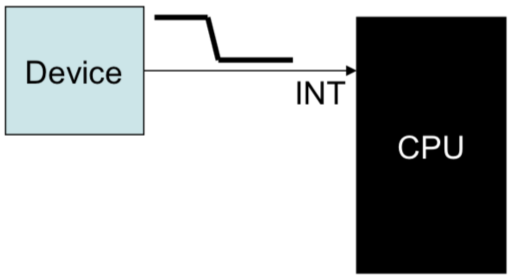


System Calls



Hardware vs Software Interrupt

Hardware Interrupt



A device (like PIC) asserts a pin in the CPU

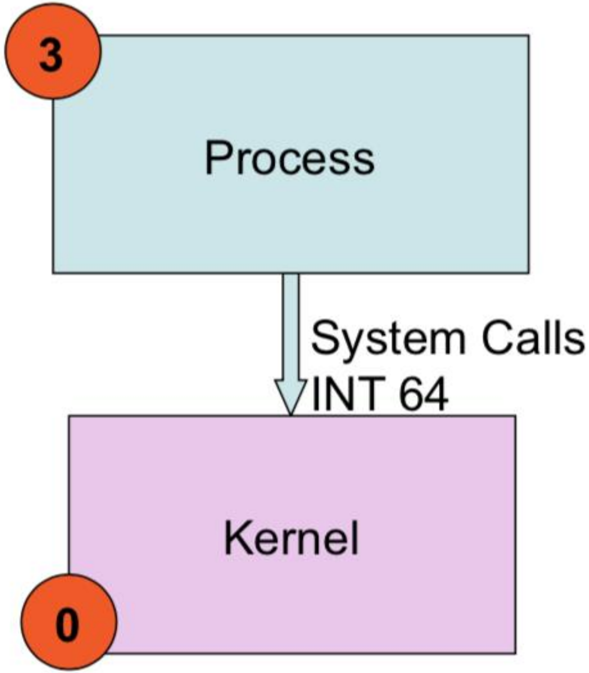
Software Interrupt

An instruction which when executed causes an interrupt

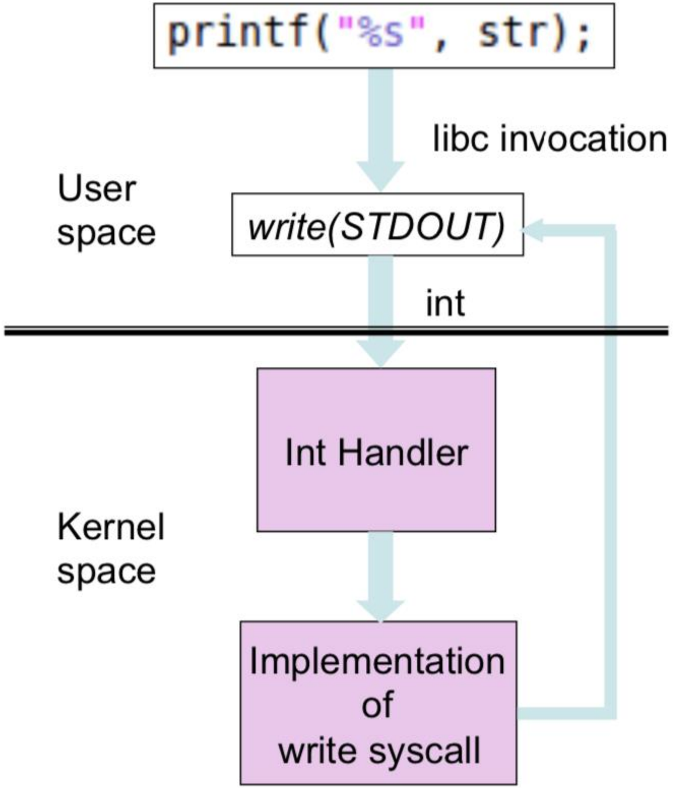
Software Interrupt

Software interrupt used for implementing system calls

- In Linux INT 128, is used for system calls
- In xv6, INT 64 is used for system calls

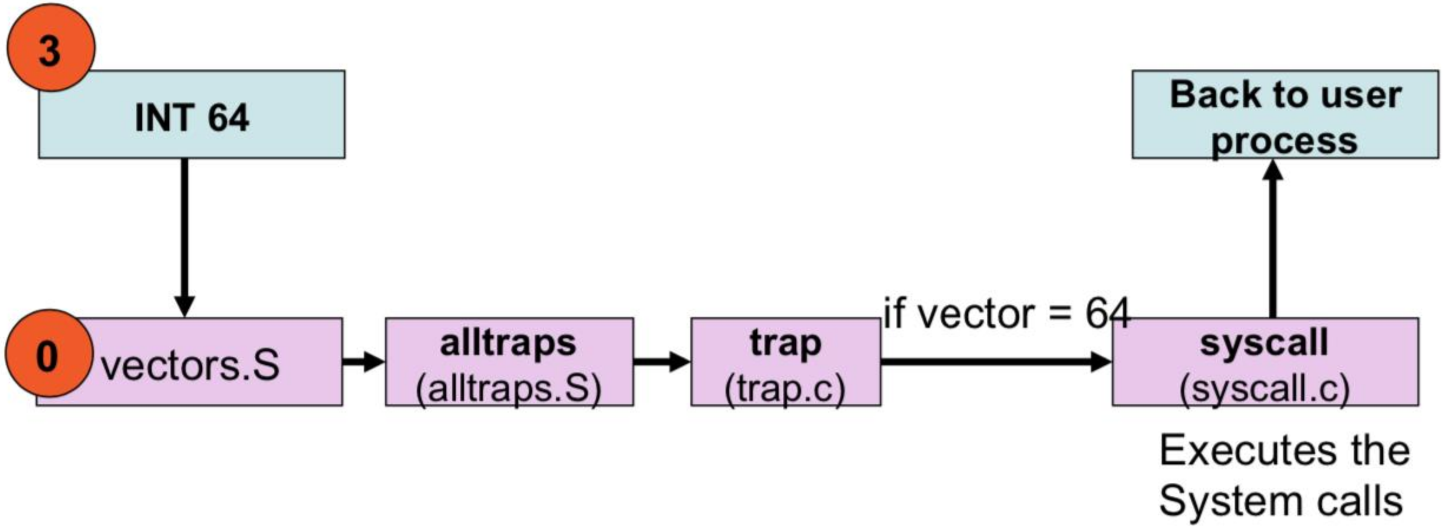


Example (write system call)

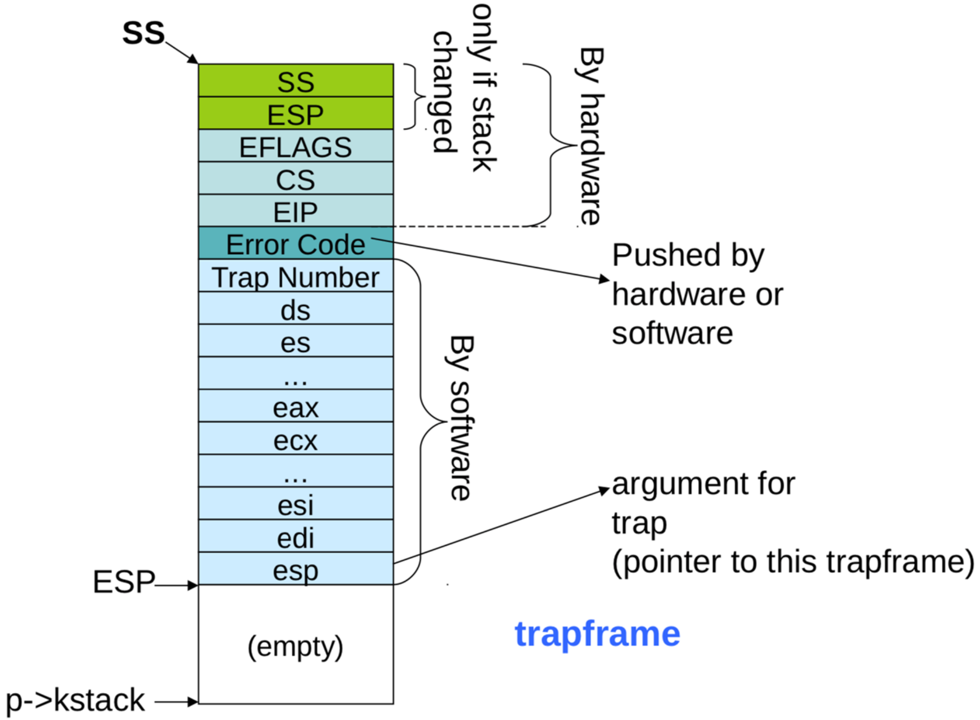


System Call Processing in kernel

Almost similar to hardware interrupts



Trapframe



ref : struct trapframe in x86.h (0602 [06])

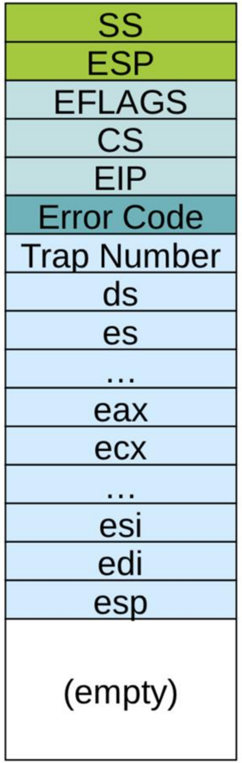
Trapframe Struct

```
struct trapframe {
  // registers as pushed by pusha
  uint edi;
  uint esi;
  uint ebp;
  uint oesp;      // useless & ignored
  uint ebx;
  uint edx;
  uint ecx;
  uint eax;

  // rest of trap frame
  ushort gs;
  ushort padding1;
  ushort fs;
  ushort padding2;
  ushort es;
  ushort padding3;
  ushort ds;
  ushort padding4;
  uint trapno;

  // below here defined by x86 hardware
  uint err;
  uint eip;
  ushort cs;
  ushort padding5;
  uint eflags;

  // below here only when crossing rings, such as from user to kernel
  uint esp;
  ushort ss;
  ushort padding6;
};
```



System Calls in xv6

System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n byes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

How does the OS distinguish between the system calls?

System Call Number

System
call number

```
mov x, %eax
INT 64
```

Based on the system call number function syscall invokes the corresponding syscall handler

System call numbers

```
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
```

System call handlers

```
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
```

ref : syscall.h, syscall() in syscall.c

xv6 System Call Naming Convention

- Usually a library function foo() will do some work and then call a system call sys_foo()
  - sys_foo() implemented in sys*.c (sysfile.c, sysproc.c)
  - All system calls begin with sys_
- System call number for foo() is SYS_foo
  - syscalls.h

Syscall(void)

All system calls are handled in this function.
The sys num which is saved in eax register is retrieved and system call is read from table.

```
void
syscall(void)
{
  int num;

  num = proc->tf->eax;
  if(num >= 0 && num < NELEM(syscalls) && syscalls[num])
    proc->tf->eax = syscalls[num]();
  else {
    cprintf("%d %s: unknown sys call %d\n",
            proc->pid, proc->name, num);
    proc->tf->eax = -1;
  }
}
```

ref : syscall.h, syscall() in syscall.c

Prototype of a Typical System Call

int system_call(resource_descriptor, parameters)

return is generally
'int' (or equivalent)
sometimes 'void'

int used to denote completion status of system call sometimes also has additional information like number of bytes written to file

What OS resource is the target here?
For example a file, device, etc.

If not specified, generally means the current process

System call specific parameters passed.
How are they passed?

Adding New System Call

- A system call body is defined in sysproc.c or sysfile.c
- Multiple files are needed to be altered to add a new syscall.
- Desired new system calls in this project:
  - void set_path ()
  - void set_sleep (int sec)
  - int get_parent_id ()
  - int get_children (int pid)

Processes

A process is the running of a program, including the program's state and data. The state includes such things as:

- Memory the program occupied
- Memory contents
- Register values
- Files
- Kernel structures

Managing Processes

The kernel has a simple data structure for each process, organized in some list. The kernel juggles between the processes, using a context switch, which:

- Saves state of old process to memory.
- Loads state of new process from memory

The processes are held in a struct named ptable, who has a vector of processes named proc.

First Process

Processes are created by the kernel, after another process asks it to. Therefore, the kernel needs to run the first process itself, in order to create someone who will ask for new processes to be created.

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  volatile int pid;            // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

Proc structure Ref: proc.h

Working with Date

- The cmostime() function defined in lapic.c file can be used to get the real time clock.
- The cmostime() resolution is one second.
- The date.h file contains the definition of the rtcdate data structure, a pointer to which you will provide as an argument to cmostime().

Finding a Process with Given Pid

- You need to iterate on the ptable to find the matching pid and return the process.
- For iterating on ptable a lock should first be acquired.
- Code looks something like this:

```
Proc* get_process(int pid) {
        struct proc *p;
        acquire(&ptable.lock);
        …
        release(&ptable.lock);
}
```

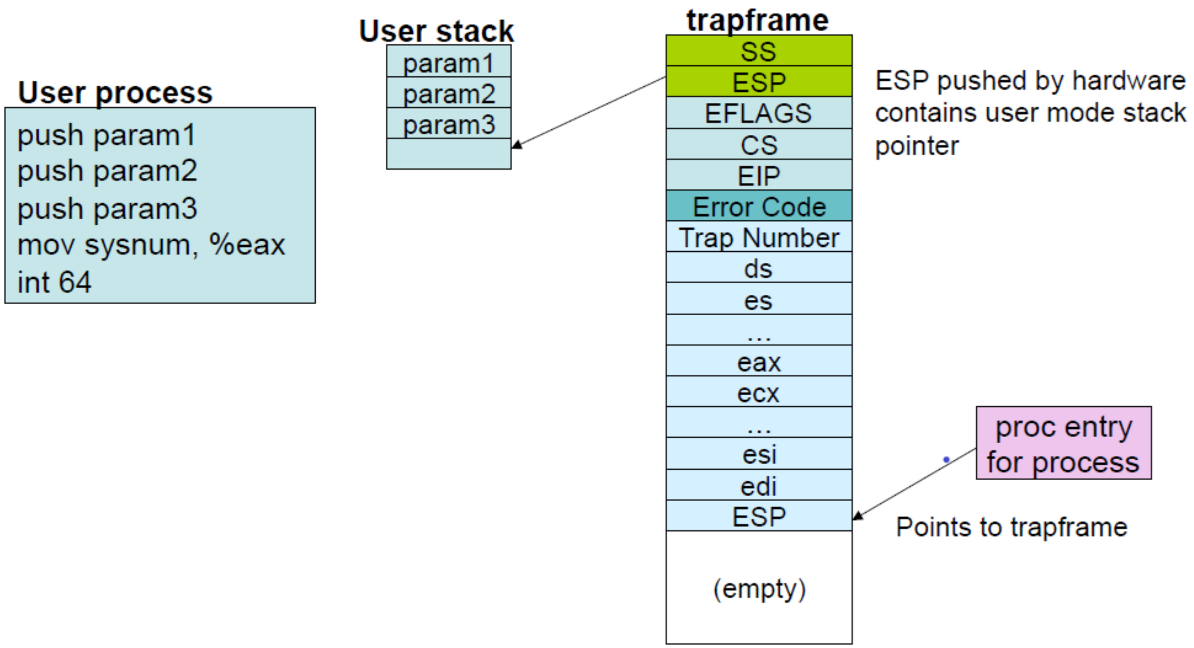
Passing Parameters in System Calls

- Passing parameters to system calls not similar to passing parameters in function calls.
    - Recall stack changes from user mode stack to kernelstack.
- Typical Methods
    - Pass by Registers (eg. Linux)
    - Pass via user mode stack (eg. xv6)
        - Complex
    - Pass via a designated memory region
        - Address passed through registers

Pass By Registers(Linux)

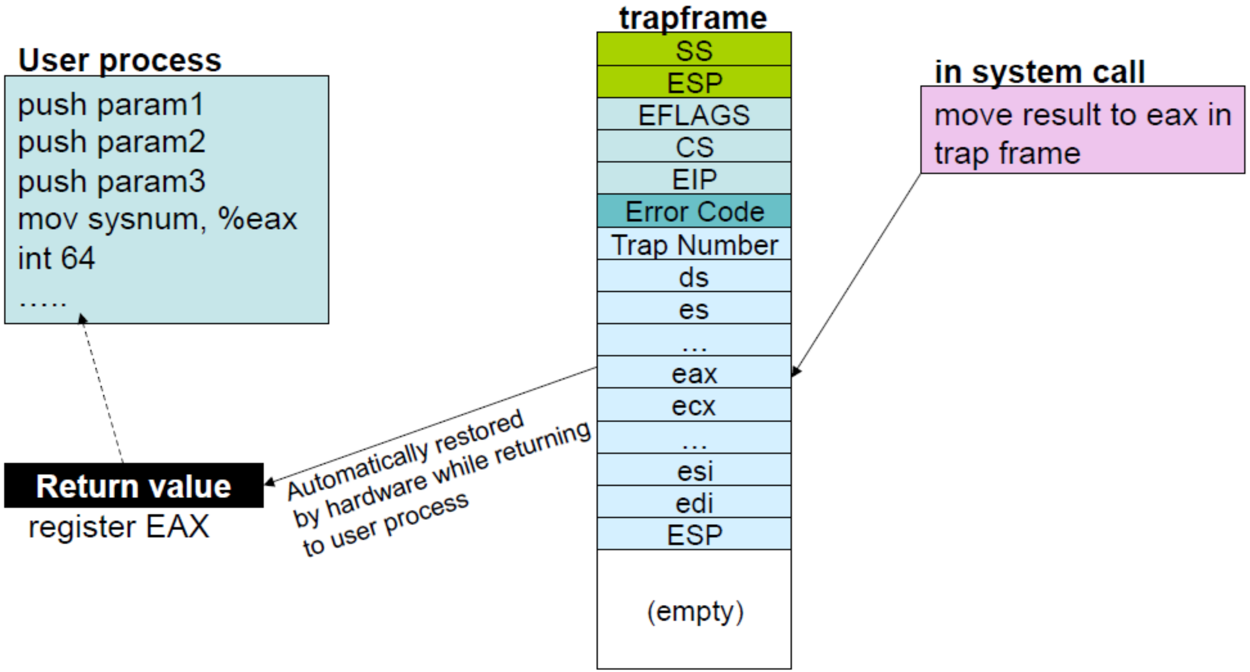
- System calls with fewer than 6 parameters passed in registers
  - – %eax (sys call number), %ebx, %ecx,, %esi, %edi, %ebp
- If 6 or more arguments
  - – Pass pointer to block structure containing argument list
- Max size of argument is the register size (eg. 32 bit)
  - – Larger pointers passed through pointers

Pass via User Mode Stack(xv6)



ref : sys_open (sysfile.c), argint, fetchint (syscall.c)

Returns from System Calls



SYS_count_num_of_digits (int num)

- Get a number as argument
- Return number of its digits
- Use a register to store the argument

Sleep System Call

- set_sleep

Adding PATH Environment Variable

- define a global variable
- set PATH /:bin:
  - system call
  - delimiter ---> :
- exec() in exec.c
  - some modification

Accessing Children Processes of a Process

- get_parent_id
- get_children(pid)
- BONUS

Deadline

- 19th Aban
- Provide Report
- Resources:
  - https://www.cs.columbia.edu/~junfeng/11spw4118/lectures/l07-proc-xv6.pdf
  - http://www.cse.iitm.ac.in/~chester/courses/16o_os/slides/6_Interrupts.pdf