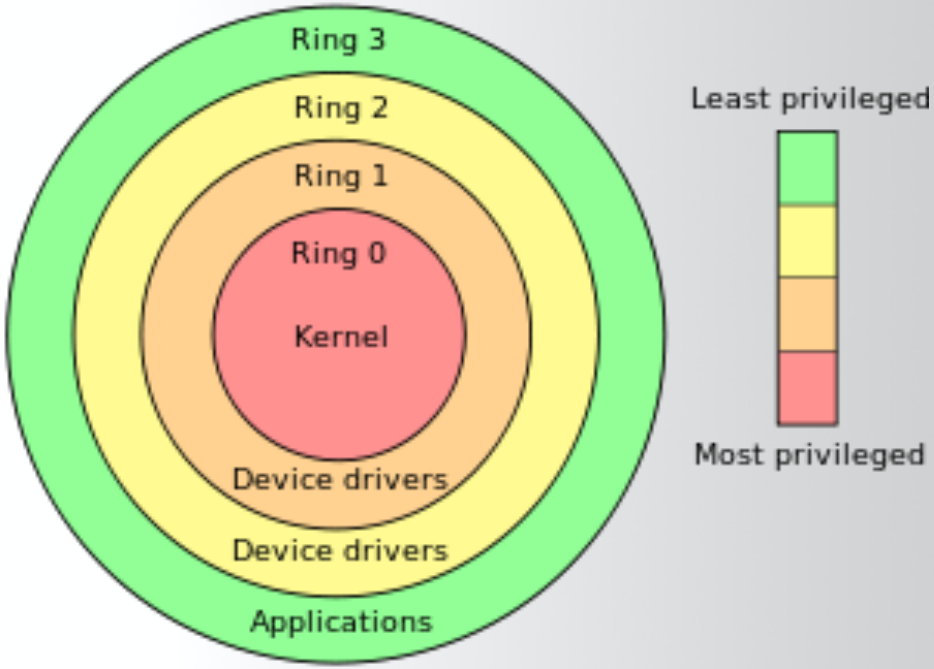
Class 4

CSC 472/583 Software Security
System Call, Shellcode

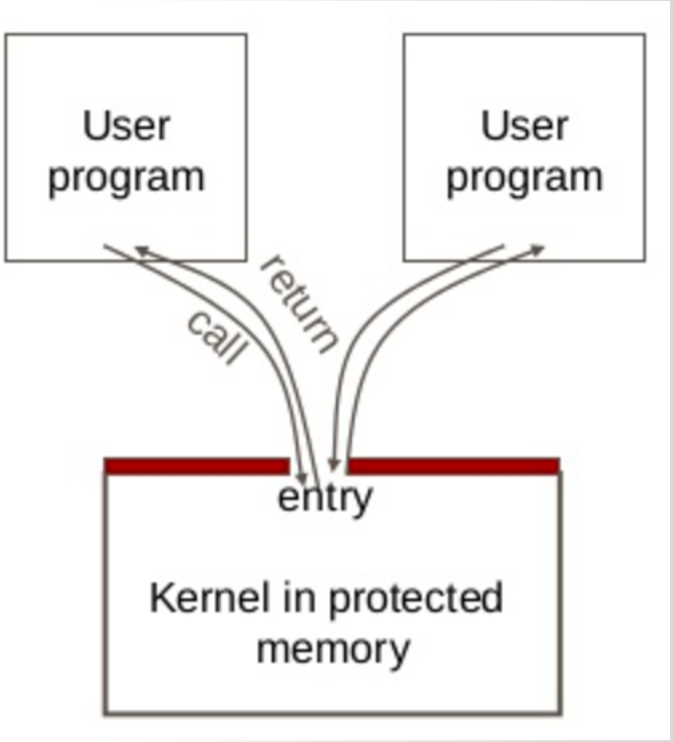
Dr. Si Chen (schen@wcupa.edu)

System Call


Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged

System Call

- User code can be arbitrary
- User code cannot modify kernel memory
- The call mechanism switches code to kernel mode



What is System Call?

- System resources (file, network, IO, device) may be accessed by multiple applications at the same time, can cause confliction.
- Modern OS protect these resources.

- E.g. How to let a program to wait for a while?

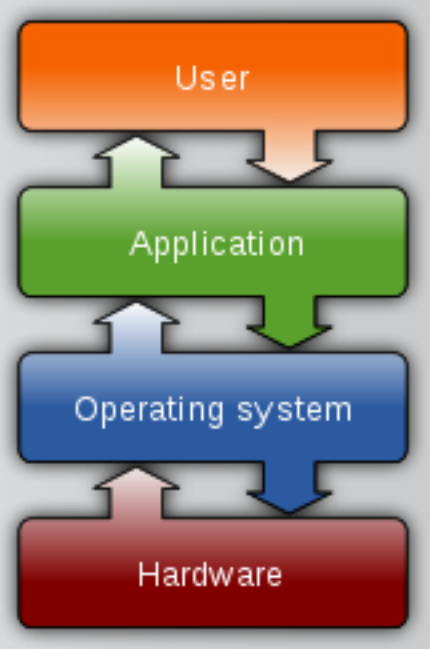
```
1 int i;
2 for(int = 0; i < 100000; ++i);
```

100Mhz CPU -> 1s
1000Mhz CPU -> 0.1s

Use OS provide Timer

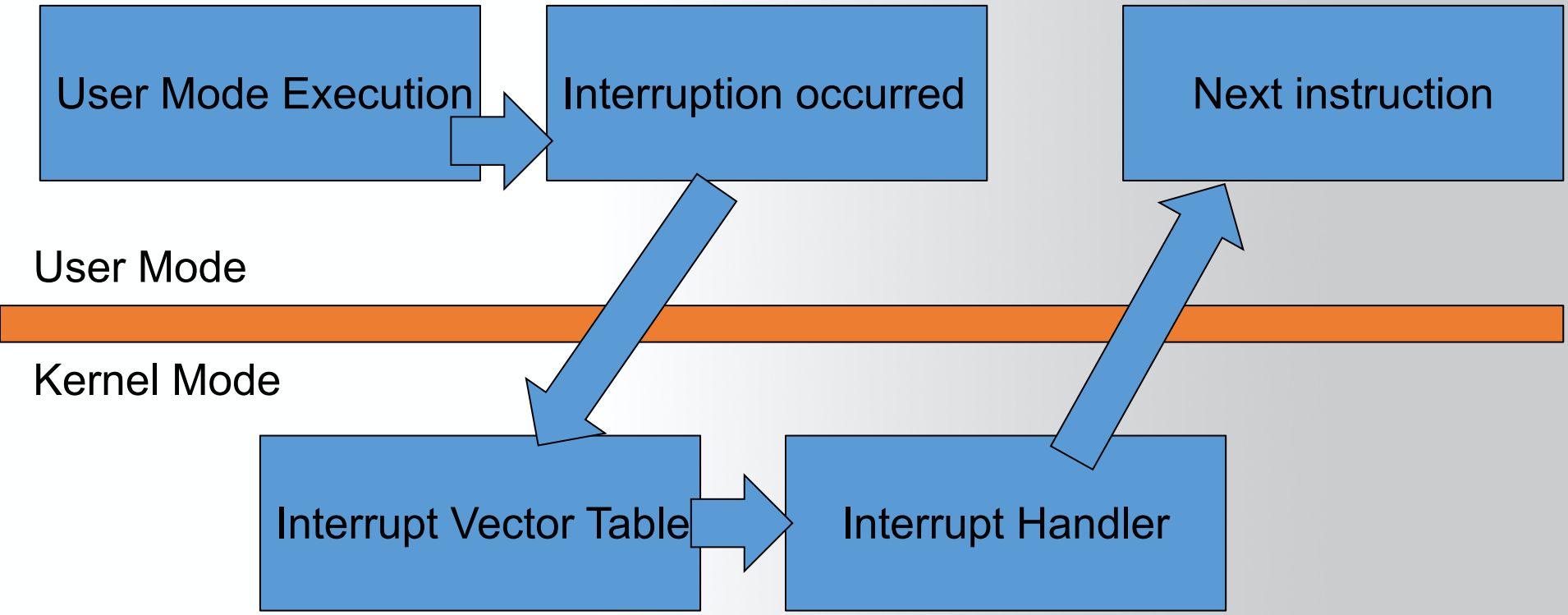
What System Call?

- Let an application to access system resources.
- OS provide an interface (**System call**) for the application
- It usually use the technique called “interrupt vector”
  - Linux use 0x80
  - Windows use 0x2E

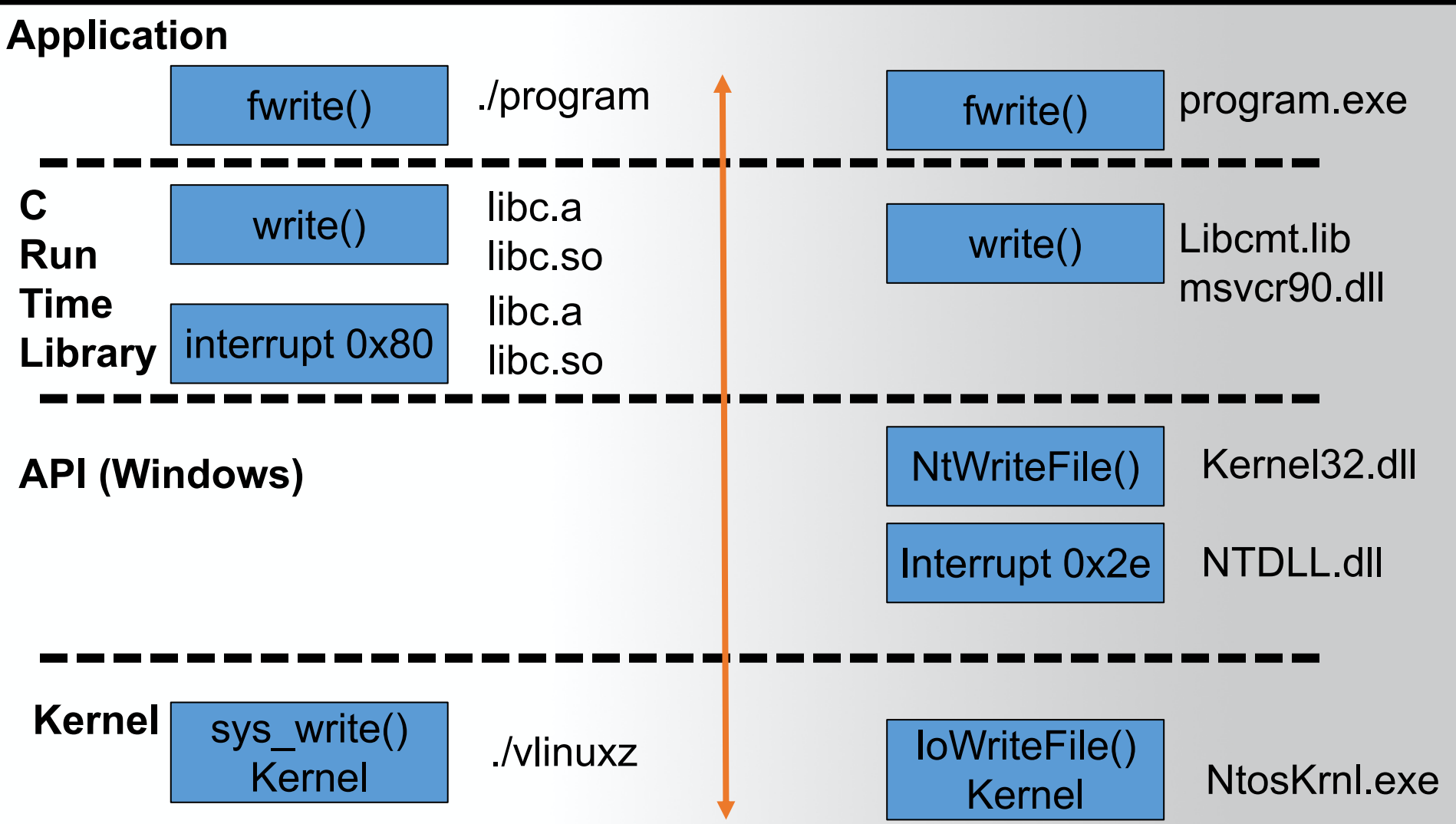


In system programming, an **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an *interrupt handler* (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.[1] There are two types of interrupts: hardware interrupts and software interrupts. – From Wikipedia

CPU Interrupt



fwrite() path in both Linux and Windows



Linux System Call

Linux Syscall Reference

http://syscalls.kernelgrok.com

Show All entries Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	–	–	–	–	–	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	–	–	–	–	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	–	–	–	–	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	–	–	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	–	–	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	–	–	fs/open.c:900
6	sys_close	0x06	unsigned int fd	–	–	–	–	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	–	–	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	–	–	–	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	–	–	–	fs/namei.c:2520
10	sys_unlink	0x0a	const char __user *pathname	–	–	–	–	fs/namei.c:2352
11	sys_execve	0x0b	char __user *	char __user *__user *	char __user *__user *	struct pt_regs *	–	arch/alpha/kernel/entry.S:925
12	sys_chdir	0x0c	const char __user *filename	–	–	–	–	fs/open.c:361
13	sys_time	0x0d	time_t __user *tloc	–	–	–	–	kernel/posix-timers.c:855
14	sys_mknod	0x0e	const char __user *filename	int mode	unsigned dev	–	–	fs/namei.c:2067
15	sys_chmod	0x0f	const char __user *filename	mode_t mode	–	–	–	fs/open.c:507
16	sys_lchown16	0x10	const char __user *filename	old_uid_t user	old_gid_t group	–	–	kernel/uid16.c:27
17	not implemented	0x11	–	–	–	–	–	
18	sys_stat	0x12	char __user *filename	struct __old_kernel_stat __user *statbuf	–	–	–	fs/stat.c:150
19	sys_lseek	0x13	unsigned int fd	off_t offset	unsigned int origin	–	–	fs/read_write.c:167
20	sys_getpid	0x14	–	–	–	–	–	kernel/timer.c:1337
21	sys_mount	0x15	char __user *dev_name	char __user *dir_name	char __user *type	unsigned long flags	void __user *data	fs/namespace.c:2118
22	sys_oldumount	0x16	char __user *name	–	–	–	–	fs/namespace.c:1171

Linux Syscall Reference

Show All entries Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	–	–	–	–	–	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	–	–	–	–	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	–	–	–	–	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	–	–	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	–	–	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	–	–	fs/open.c:900
6	sys_close	0x06	unsigned int fd	–	–	–	–	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	–	–	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	–	–	–	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	–	–	–	fs/namei.c:2520
10	sys_unlink	0x0a	const char __user *pathname	–	–	–	–	fs/namei.c:2352
11	sys_execve	0x0b	char __user *	char __user *__user *	char __user *__user *	struct pt_regs *	–	arch/alpha/kernel/entry.S:925
12	sys_chdir	0x0c	const char __user *filename	–	–	–	–	fs/open.c:361
13	sys_time	0x0d	time_t __user *tloc	–	–	–	–	kernel/posix-timers.c:855
14	sys_mknod	0x0e	const char __user *filename	int mode	unsigned dev	–	–	fs/namei.c:2067
15	sys_chmod	0x0f	const char __user *filename	mode_t mode	–	–	–	fs/open.c:507
16	sys_lchown16	0x10	const char __user *filename	old_uid_t user	old_gid_t group	–	–	kernel/uid16.c:27
17	not implemented	0x11	–	–	–	–	–	
18	sys_stat	0x12	char __user *filename	struct __old_kernel_stat __user *statbuf	–	–	–	fs/stat.c:150
19	sys_lseek	0x13	unsigned int fd	off_t offset	unsigned int origin	–	–	fs/read_write.c:167
20	sys_getpid	0x14	–	–	–	–	–	kernel/timer.c:1337
21	sys_mount	0x15	char __user *dev_name	char __user *dir_name	char __user *type	unsigned long flags	void __user *data	fs/namespace.c:2118
22	sys_oldumount	0x16	char __user *name	–	–	–	–	fs/namespace.c:1171

/ include / linux / syscalls.h

```
836                                     const struct itimerspec __user *utmr,
837                                     struct itimerspec __user *otmr);
838 asmlinkage long sys_timerfd_gettime(int ufd, struct itimerspec __user *otmr);
839 asmlinkage long sys_eventfd(unsigned int count);
840 asmlinkage long sys_eventfd2(unsigned int count, int flags);
841 asmlinkage long sys_fallocate(int fd, int mode, loff_t offset, loff_t len);
842 asmlinkage long sys_old_readdir(unsigned int, struct old_linux_dirent __user *, unsigned
843 asmlinkage long sys_pselect6(int, fd_set __user *, fd_set __user *,
844                              fd_set __user *, struct timespec __user *,
845                              void __user *);
846 asmlinkage long sys_ppoll(struct pollfd __user *, unsigned int,
847                           struct timespec __user *, const sigset_t __user *,
848                           size_t);
849 asmlinkage long sys_fanotify_init(unsigned int flags, unsigned int event_f_flags);
850 asmlinkage long sys_fanotify_mark(int fanotify_fd, unsigned int flags,
851                                   u64 mask, int fd,
852                                   const char  __user *pathname);
853 asmlinkage long sys_syncfs(int fd);
854
855 asmlinkage long sys_fork(void);
856 asmlinkage long sys_vfork(void);
857 #ifdef CONFIG_CLONE_BACKWARDS
858 asmlinkage long sys_clone(unsigned long, unsigned long, int __user *, int,
859                           int __user *);
860 #else
861 asmlinkage long sys_clone(unsigned long, unsigned long, int __user *,
862                           int __user *, int);
863 #endif
864
865 asmlinkage long sys_execve(const char __user *filename,
866                            const char __user *const __user *argv,
867                            const char __user *const __user *envp);
868
869 asmlinkage long sys_perf_event_open(
870                            struct perf_event_attr __user *attr_uptr,
871                            pid_t pid, int cpu, int group_fd, unsigned long flags);
872
873 asmlinkage long sys_mmap_pgoff(unsigned long addr, unsigned long len,
874                                unsigned long prot, unsigned long flags,
875                                unsigned long fd, unsigned long pgoff);
876 asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
877 asmlinkage long sys_name_to_handle_at(int dfd, const char __user *name,
878                                       struct file_handle __user *handle,
879                                       int __user *mnt_id, int flag);
880 asmlinkage long sys_open_by_handle_at(int mountdirfd,
881                                       struct file_handle __user *handle,
882                                       int flags);
883 asmlinkage long sys_setns(int fd, int nstype);
884 asmlinkage long sys_process_vm_readv(pid_t pid,
885                                      const struct iovec __user *lvec,
886                                      unsigned long liovcnt,
887                                      const struct iovec __user *rvec,
888                                      unsigned long riovcnt,
889                                      unsigned long flags);
890 asmlinkage long sys_process_vm_writev(pid_t pid,
891                                       const struct iovec __user *lvec,
892                                       unsigned long liovcnt,
893                                       const struct iovec __user *rvec,
894                                       unsigned long riovcnt,
895                                       unsigned long flags);
896
897 asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
898                          unsigned long idx1, unsigned long idx2);
899 asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
900 #endif
```

Example: Hello World

Quick review:

- **DB** - Define Byte. 8 bits
- **DW** - Define Word. Generally 2 bytes on a typical x86 32-bit system
- **DD** - Define double word. Generally 4 bytes on a typical x86 32-bit system

From x86 assembly tutorial,

```
section .text
global _start
_start:
        mov eax, 4      ; sys_write
        mov ebx, 1      ; fd
        mov ecx, msg    ; buf
        mov edx, 13     ; size
        int 0x80        ; write(1, "Hello world!\n", 13)

        mov eax, 1      ; sys_exit
        mov ebx, 0      ; status
        int 0x80        ; exit(0)
section .data
msg:
        db 'Hello world!', 0xA
```

helloworld.asm

```
[quake0day@quake0day-pc ~]$ nasm -felf32 helloworld.asm -o helloworld.o && ld helloworld.o -melf_i386 -o helloworld
[quake0day@quake0day-pc ~]$ ./helloworld
Hello world!
```

Shellcode

Shellcode is defined as a set of instructions injected and then executed by an exploited program. **Shellcode** is used to directly manipulate registers and the functionality of an exploited program.

Crafting Shellcode (the small program)

Example: Hello World

```
1   ;hello.asm
2   [SECTION .text]
3
4   global _start
5
6
7   _start:
8
9       jmp short ender
10
11      starter:
12
13      xor eax, eax    ;clean up the registers
14      xor ebx, ebx
15      xor edx, edx
16      xor ecx, ecx
17
18      mov al, 4       ;syscall write
19      mov bl, 1       ;stdout is 1
20      pop ecx         ;get the address of the string from the stack
21      mov dl, 5       ;length of the string
22      int 0x80
23
24      xor eax, eax
25      mov al, 1       ;exit the shellcode
26      xor ebx,ebx
27      int 0x80
28
29      ender:
30      call starter    ;put the address of the string on the stack
31      db 'hello'
```

hello.asm

Crafting Shellcode (the small program)

Example: Hello (hello.asm)

To compile it use nasm:

```
→ ~ nasm -f elf hello.asm
```

Use **objdump** to get the shellcode bytes:

```
[csc495@csc495-pc ~]$ objdump -d -M intel hello.o

hello.o:     file format elf32-i386


Disassembly of section .text:

00000000 <_start>:
   0:   eb 19                   jmp    1b <call_shellcode>

00000002 <shellcode>:
   2:   31 c0                   xor    eax,eax
   4:   b0 04                   mov    al,0x4
   6:   31 db                   xor    ebx,ebx
   8:   b3 01                   mov    bl,0x1
   a:   59                      pop    ecx
   b:   31 d2                   xor    edx,edx
   d:   b2 0d                   mov    dl,0xd
   f:   cd 80                   int    0x80
  11:   31 c0                   xor    eax,eax
  13:   b0 01                   mov    al,0x1
  15:   31 db                   xor    ebx,ebx
  17:   b3 05                   mov    bl,0x5
  19:   cd 80                   int    0x80

0000001b <call_shellcode>:
  1b:   e8 e2 ff ff ff          call   2 <shellcode>
  20:   48                      dec    eax
  21:   65 6c                   gs insb DX,es:[edi]
  23:   6c                      insb   DX,es:[edi]
  24:   6f                      outsd  DX,DWORD PTR ds:[esi]
  25:   2c 20                   sub    al,0x20
  27:   77 6f                   ja     98 <call_shellcode+0x7d>
  29:   72 6c                   jb     97 <call_shellcode+0x7c>
  2b:   64 21 0a                and    DWORD PTR fs:[edx],ecx
```

Crafting Shellcode (the small program)

```
Disassembly of section .text:

00000000 <_start>:
   0:   eb 19                   jmp    1b <ender>

00000002 <starter>:
   2:   31 c0                   xor    eax,eax
   4:   31 db                   xor    ebx,ebx
   6:   31 d2                   xor    edx,edx
   8:   31 c9                   xor    ecx,ecx
   a:   b0 04                   mov    al,0x4
   c:   b3 01                   mov    bl,0x1
   e:   59                      pop    ecx
   f:   b2 05                   mov    dl,0x5
  11:   cd 80                   int    0x80
  13:   31 c0                   xor    eax,eax
  15:   b0 01                   mov    al,0x1
  17:   31 db                   xor    ebx,ebx
  19:   cd 80                   int    0x80

0000001b <ender>:
  1b:   e8 e2 ff ff ff          call   2 <starter>
  20:   68 65 6c 6c 6f          push   0x6f6c6c65
```

Extracting the bytes gives us the shellcode:

```
\xeb\x19\x31\xc0\x31\xdb\x31\xd2\x31\xc9\xb0\x04\xb3\x01\x59\xb2\x05\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80\xe8\xe2\xff\xff\xff\x68\x65\x6c\x6c\x6f
```

Test Shellcode (test.c)

```
1 char code[] = "\xeb\x19\x31\xc0\x31\xdb\x31\xd2\x31\xc9\xb0\x04\xb3\x01\x59\xb2\x05\xcd"\
2               "\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80\xe8\xe2\xff\xff\xff\x68\x65\x6c\x6c\x6f";
3 int main(int argc, char **argv)
4 {
5   int (*func)();
6   func = (int (*)()) code;
7   (int)(*func)();
8 }
```

```
➜ ~ gcc test.c -o test -fno-stack-protector -zexecstack -no-pie
➜ ~ ./test
hello%
```

Shellcode

- **Taking some shellcode from Aleph One's 'Smashing the Stack for Fun and Profit'**

```
shellcode =
("\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b" +
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd" +
"\x80\xe8\xdc\xff\xff\xff/bin/sh")
```

Q & A