

# System Security - Attack and Defense for Binaries

---

CS 4390/5390, Spring 2026

Instructor: MD Armanuzzaman (*Arman*)

# Agenda

- Background knowledge
  - Compiler, linker, loader
  - x86 and x86-64 architectures and ISA
  - Linux fundamentals
    - Linux file permissions
    - Set-UID programs
    - Memory map of a Linux process
    - System calls
    - Environment and Shell variables
    - ELF files
    - Reverse engineering tools

# Background Knowledge: Memory Map of a Linux Process

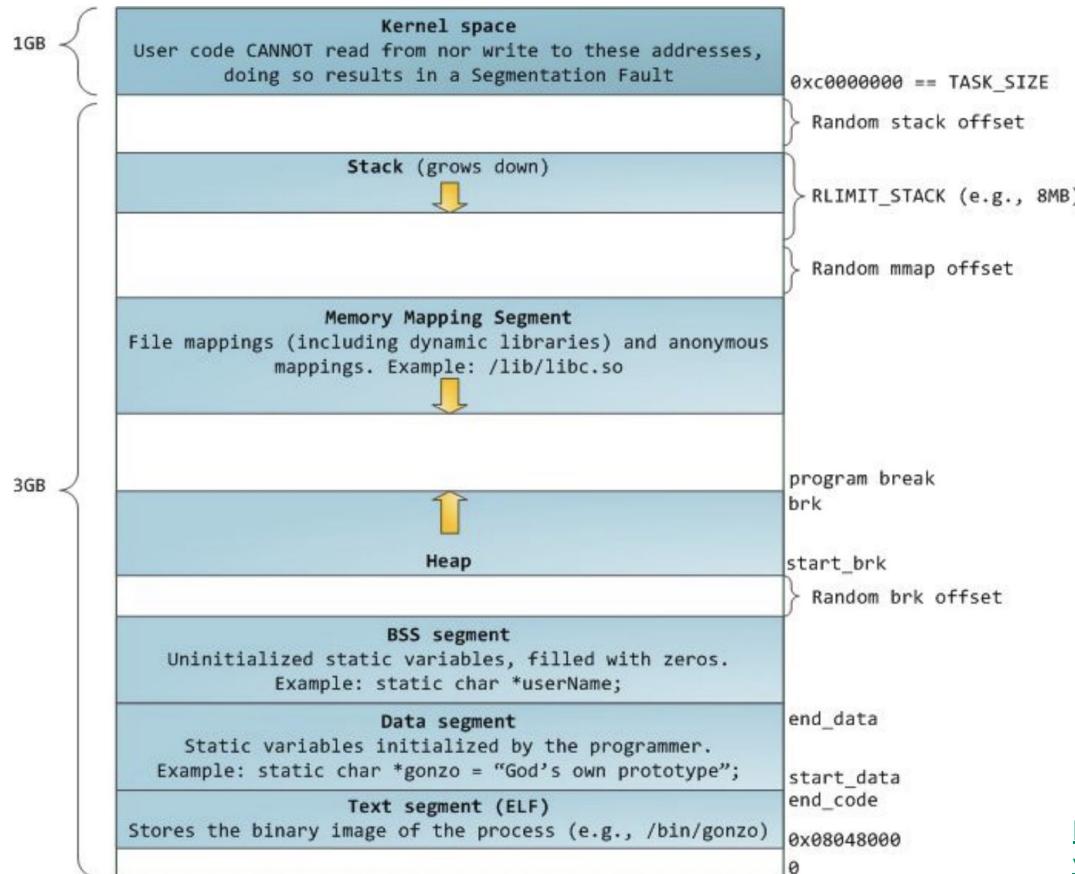
# Memory Map of Linux Process (32 bit)

Each process in a multi-tasking OS runs in its own memory sandbox.

This sandbox is the **virtual address space**, which in 32-bit mode is **always a 4GB block of memory addresses**.

These virtual addresses are mapped to physical memory by **page tables**, which are maintained by the operating system kernel and consulted by the processor

# Memory Map of Linux Process (32 bit)



# NULL Pointer in C/C++

```
int * pInt = NULL;
```

In possible definitions of NULL in C/C++:

```
#define NULL ((char *)0)
```

```
#define NULL 0
```

```
//since C++11
```

```
#define NULL nullptr
```

# /proc/pid\_of\_process/maps

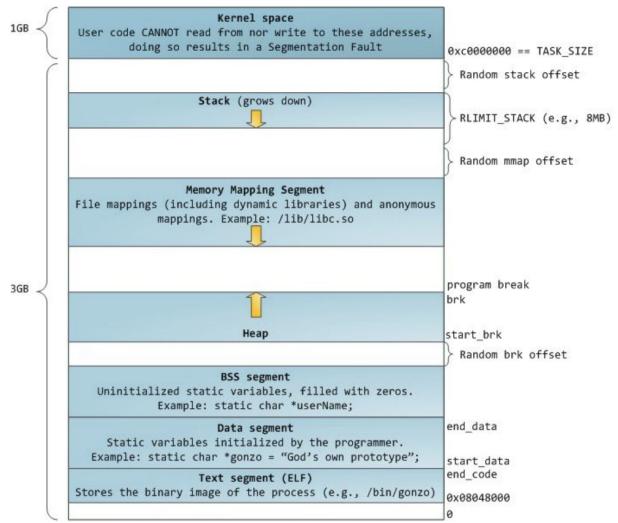
Example processmap.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    getchar();
    return 0;
}
```

cat /proc/pid/maps

pmap -X pid

pmap -X `pidof pm`



# Memory Map of Linux Process (64 bit system)

Address	Perm	Offset	Device	Inode	Size	Rss	Pss	Pss_Dirty	Referenced	Anonymous	KSM	LazyFree	ShmemPmdMapped	FilePmdMapped	Shared_Hugetlb	Private_Hugetlb	Swap	SwapPss	Locked	THPeligible	ProtectionKey	Mapping
56e4fc8ed000	r--p	00000000	fc:00	3146755	4	4	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0 [processmap]	
56e4fc8ee000	r-xp	00001000	fc:00	3146755	4	4	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0 [processmap]	
56e4fc8ef000	r--p	00002000	fc:00	3146755	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 [processmap]	
56e4fc8f0000	r--p	00002000	fc:00	3146755	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0 [processmap]	
56e4fc8f1000	rw-p	00003000	fc:00	3146755	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0 [processmap]	
56e51f43c000	rw-p	00000000	00:00	0	132	4	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0 [heap]	
7d0ef6400000	r--p	00000000	fc:00	5540086	160	160	3	0	160	0	0	0	0	0	0	0	0	0	0	0	0 [libc.so.6]	
7d0ef6428000	r-xp	00028000	fc:00	5540086	1568	864	19	0	864	0	0	0	0	0	0	0	0	0	0	0	0 [libc.so.6]	
7d0ef65b0000	r--p	001b0000	fc:00	5540086	316	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 [libc.so.6]	
7d0ef65ff000	r--p	001fe000	fc:00	5540086	16	16	16	16	16	16	0	0	0	0	0	0	0	0	0	0	0 [libc.so.6]	
7d0ef6603000	rw-p	00202000	fc:00	5540086	8	8	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0 [libc.so.6]	
7d0ef6605000	rw-p	00000000	00:00	0	52	20	20	20	20	20	0	0	0	0	0	0	0	0	0	0	0	
7d0ef67fa000	rw-p	00000000	00:00	0	12	8	8	8	8	8	8	0	0	0	0	0	0	0	0	0	0	
7d0ef6804000	rw-p	00000000	00:00	0	8	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0	0	
7d0ef6806000	r--p	00000000	fc:00	5540083	4	4	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0 [ld-linux-x86-64.so.2]	
7d0ef6807000	r-xp	00001000	fc:00	5540083	172	172	3	0	172	0	0	0	0	0	0	0	0	0	0	0	0 [ld-linux-x86-64.so.2]	
7d0ef6832000	r--p	0002c000	fc:00	5540083	40	40	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0 [ld-linux-x86-64.so.2]	
7d0ef683c000	r--p	00036000	fc:00	5540083	8	8	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0 [ld-linux-x86-64.so.2]	
7d0ef683e000	rw-p	00038000	fc:00	5540083	8	8	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0 [ld-linux-x86-64.so.2]	
7ffe0eaec000	rw-p	00000000	00:00	0	132	12	12	12	12	12	0	0	0	0	0	0	0	0	0	0	0 [stack]	
7ffe0fb9000	r--p	00000000	00:00	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 [var]	
7ffe0afbd000	r-xp	00000000	00:00	0	8	4	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0 [vds]	
ffffffffffff600000	--xp	00000000	00:00	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 [vsyscall]	
<hr/>																						
					2684	1348	129	96	1348	96	0	0	0	0	0	0	0	0	0	0	0 KB	

# Background Knowledge: System Calls

# What is System Call?

When a process needs to invoke a kernel service, it invokes a procedure call in the operating system interface using special instructions (not a **call** instruction in x86). Such a procedure is called a system call.

The system call enters the kernel; the kernel performs the service and returns. Thus a process alternates between executing in user space and kernel space.

System calls are generally not invoked directly by a program, but rather via wrapper functions in glibc (or perhaps some other library).

# Popular System Call

On Unix, Unix-like and other POSIX-compliant operating systems, popular system calls are open, read, write, close, wait, exec, fork, exit, and kill.

Many modern operating systems have hundreds of system calls. For example, Linux and OpenBSD each have over 300 different calls, FreeBSD has over 500, Windows 7 has close to 700.

# Glibc interfaces

Often, but not always, the name of the wrapper function is the same as the name of the system call that it invokes

For example, glibc contains a function `chdir()` which invokes the underlying "chdir" system call.

# Tools: strace & ltrace

## misc/firstflag main.c

```
int main(int argc, char *argv[])
```

1

```
printf("Congratulations on getting your first flag!!\n");
print_flag();
```

3

flag.h

```
int print_flag()
```

1

```
FILE *fp = NULL;
```

```
char buff[MAX FLAG SIZE] = {0};
```

```
fp = fopen("/flag", "r")
```

```
if (fp == NULL)
```

1

```
printf("Error: Cannot open the flag file!!!\n");  
return 1;
```

10

```
fread(buff, MAX_FLAG_SIZE - 2, 1, fp)
```

```
printf("The flag is: %s\n", buff)
```

`fclose(fp)`

```
return 0
```

# Tools: strace & ltrace

## Execve - first system call

## Access - check file permission

## Brk - check data segment/heap

## Arch\_prctl - set architecture-specific thread state

## Fcntl - manipulate file descriptor

## Openat - similar to open

## Fstat - get file status

# Mmap

## Close Read

Read  
Broad/4 - similar to read

**Protect** - set protection on a region of memory

Mprotect - set protection on a region of memory  
Munmap - map files or devices into memory

## Plain Write

## Write

Use “man 2 syscall name” to check out its usage

# Making a System Call in x86/64 Assembly

On x86/x86-64, most system calls rely on the software interrupt.

A software interrupt is caused either by an **exceptional condition** in the processor itself, or a **special instruction** (the `int 0x80` instruction or `syscall` instruction)

For example: a divide-by-zero exception will be thrown if the processor's arithmetic logic unit is commanded to divide a number by zero as this instruction is in error and impossible.

# Making a System Call in x86 Assembly (INT 0x80)

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	<a href="#">man/cs/</a>	0x00	-	-	-	-	-	-
1	exit	<a href="#">man/cs/</a>	0x01	int error_code	-	-	-	-	-
2	fork	<a href="#">man/cs/</a>	0x02	-	-	-	-	-	-
3	read	<a href="#">man/cs/</a>	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	<a href="#">man/cs/</a>	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	<a href="#">man/cs/</a>	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	<a href="#">man/cs/</a>	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	<a href="#">man/cs/</a>	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	<a href="#">man/cs/</a>	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	<a href="#">man/cs/</a>	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	<a href="#">man/cs/</a>	0x0a	const char *pathname	-	-	-	-	-
11	execve	<a href="#">man/cs/</a>	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	<a href="#">man/cs/</a>	0x0c	const char *filename	-	-	-	-	-
13	time	<a href="#">man/cs/</a>	0x0d	time_t *tloc	-	-	-	-	-
14	mknod	<a href="#">man/cs/</a>	0x0e	const char *filename	umode_t mode	unsigned dev	-	-	-
15	chmod	<a href="#">man/cs/</a>	0x0f	const char *filename	umode_t mode	-	-	-	-
16	lchown	<a href="#">man/cs/</a>	0x10	const char *filename	uid_t user	gid_t group	-	-	-
17	break	<a href="#">man/cs/</a>	0x11	?	?	?	?	?	?

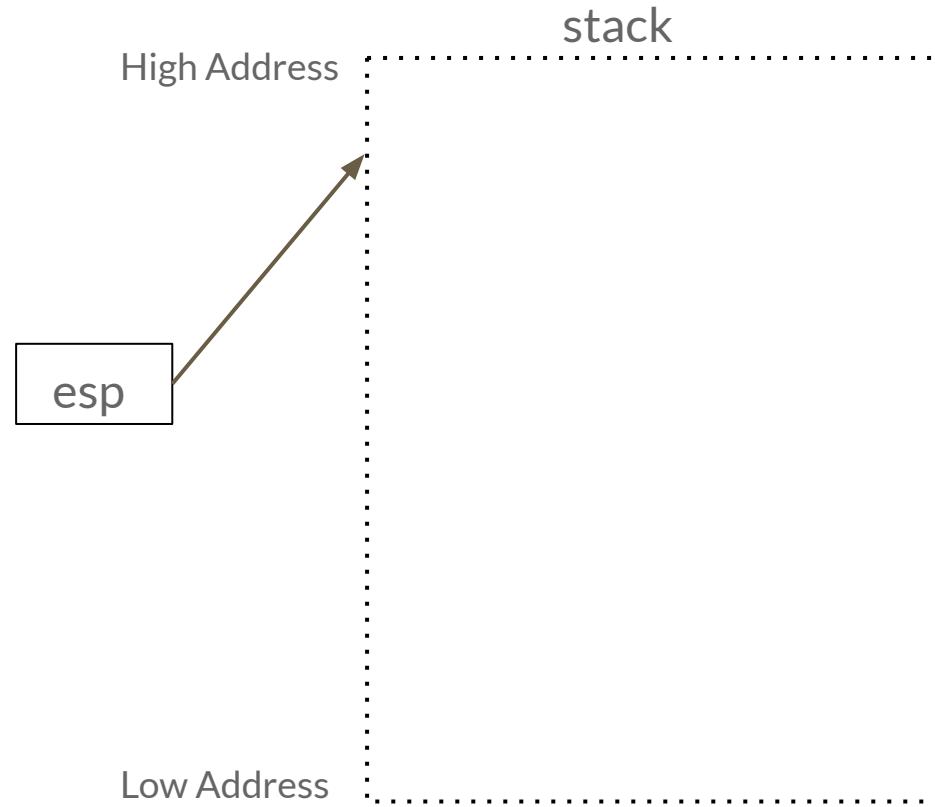
# Making a System Call in x86 Assembly

```
xor eax,eax  
push eax  
push 0x68732f2f  
push 0x6e69622f  
mov ebx,esp  
push eax  
push ebx  
mov ecx,esp  
mov al,0xb  
int 0x80
```

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0 000	NUL	(null)	32	20 040	&#32;	Space		64	40 100	&#64;	Ø	96	60 140	&#96;	`		
1	1 001	SOH	(start of heading)	33	21 041	&#33;	!	!	65	41 101	&#65;	A	97	61 141	&#97;	a		
2	2 002	STX	(start of text)	34	22 042	&#34;	"	"	66	42 102	&#66;	B	98	62 142	&#98;	b		
3	3 003	ETX	(end of text)	35	23 043	&#35;	#	#	67	43 103	&#67;	C	99	63 143	&#99;	c		
4	4 004	EOT	(end of transmission)	36	24 044	&#36;	\$	\$	68	44 104	&#68;	D	100	64 144	&#100;	d		
5	5 005	ENQ	(enquiry)	37	25 045	&#37;	%	%	69	45 105	&#69;	E	101	65 145	&#101;	e		
6	6 006	ACK	(acknowledge)	38	26 046	&#38;	&	&	70	46 106	&#70;	F	102	66 146	&#102;	f		
7	7 007	BEL	(bell)	39	27 047	&#39;	'	'	71	47 107	&#71;	G	103	67 147	&#103;	g		
8	8 010	BS	(backspace)	40	28 050	&#40;	(	(	72	48 110	&#72;	H	104	68 150	&#104;	h		
9	9 011	TAB	(horizontal tab)	41	29 051	&#41;	)	)	73	49 111	&#73;	I	105	69 151	&#105;	i		
10	A 012	LF	(NL line feed, new line)	42	2A 052	&#42;	*	*	74	4A 112	&#74;	J	106	6A 152	&#106;	j		
11	B 013	VT	(vertical tab)	43	2B 053	&#43;	+	+	75	4B 113	&#75;	K	107	6B 153	&#107;	k		
12	C 014	FF	(NP form feed, new page)	44	2C 054	&#44;	,	,	76	4C 114	&#76;	L	108	6C 154	&#108;	l		
13	D 015	CR	(carriage return)	45	2D 055	&#45;	-	-	77	4D 115	&#77;	M	109	6D 155	&#109;	m		
14	E 016	SO	(shift out)	46	2E 056	&#46;	.	.	78	4E 116	&#78;	N	110	6E 156	&#110;	n		
15	F 017	SI	(shift in)	47	2F 057	&#47;	/	/	79	4F 117	&#79;	O	111	6F 157	&#111;	o		
16	10 020	DLE	(data link escape)	48	30 060	&#48;	0	0	80	50 120	&#80;	P	112	70 160	&#112;	p		
17	11 021	DC1	(device control 1)	49	31 061	&#49;	1	1	81	51 121	&#81;	Q	113	71 161	&#113;	q		
18	12 022	DC2	(device control 2)	50	32 062	&#50;	2	2	82	52 122	&#82;	R	114	72 162	&#114;	r		
19	13 023	DC3	(device control 3)	51	33 063	&#51;	3	3	83	53 123	&#83;	S	115	73 163	&#115;	s		
20	14 024	DC4	(device control 4)	52	34 064	&#52;	4	4	84	54 124	&#84;	T	116	74 164	&#116;	t		
21	15 025	NAK	(negative acknowledge)	53	35 065	&#53;	5	5	85	55 125	&#85;	U	117	75 165	&#117;	u		
22	16 026	SYN	(synchronous idle)	54	36 066	&#54;	6	6	86	56 126	&#86;	V	118	76 166	&#118;	v		
23	17 027	ETB	(end of trans. block)	55	37 067	&#55;	7	7	87	57 127	&#87;	W	119	77 167	&#119;	w		
24	18 030	CAN	(cancel)	56	38 070	&#56;	8	8	88	58 130	&#88;	X	120	78 170	&#120;	x		
25	19 031	EM	(end of medium)	57	39 071	&#57;	9	9	89	59 131	&#89;	Y	121	79 171	&#121;	y		
26	1A 032	SUB	(substitute)	58	3A 072	&#58;	:	:	90	5A 132	&#90;	Z	122	7A 172	&#122;	z		
27	1B 033	ESC	(escape)	59	3B 073	&#59;	;	;	91	5B 133	&#91;	[	123	7B 173	&#123;	{		
28	1C 034	FS	(file separator)	60	3C 074	&#60;	<	<	92	5C 134	&#92;	\	124	7C 174	&#124;			
29	1D 035	GS	(group separator)	61	3D 075	&#61;	=	=	93	5D 135	&#93;	]	125	7D 175	&#125;	}		
30	1E 036	RS	(record separator)	62	3E 076	&#62;	>	>	94	5E 136	&#94;	^	126	7E 176	&#126;	~		
31	1F 037	US	(unit separator)	63	3F 077	&#63;	?	?	95	5F 137	&#95;	_	127	7F 177	&#127;	DEL		

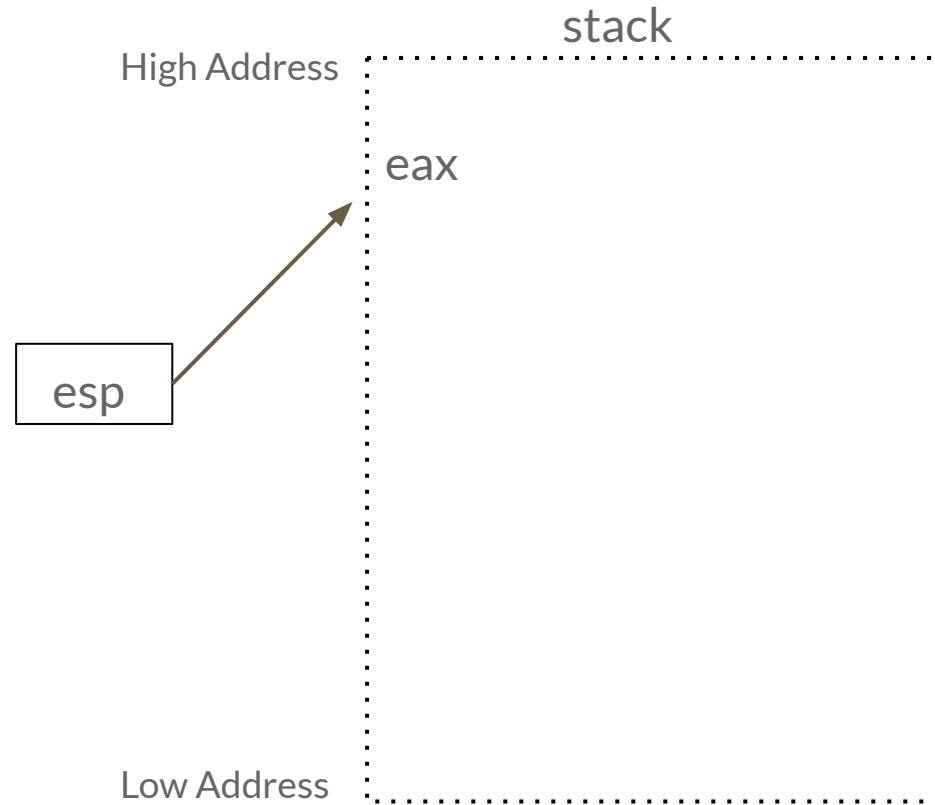
# Making a System Call in x86 Assembly

```
xor eax,eax  
push eax  
push 0x68732f2f  
push 0x6e69622f  
mov ebx,esp  
push eax  
push ebx  
mov ecx,esp  
mov al,0xb  
int 0x80
```



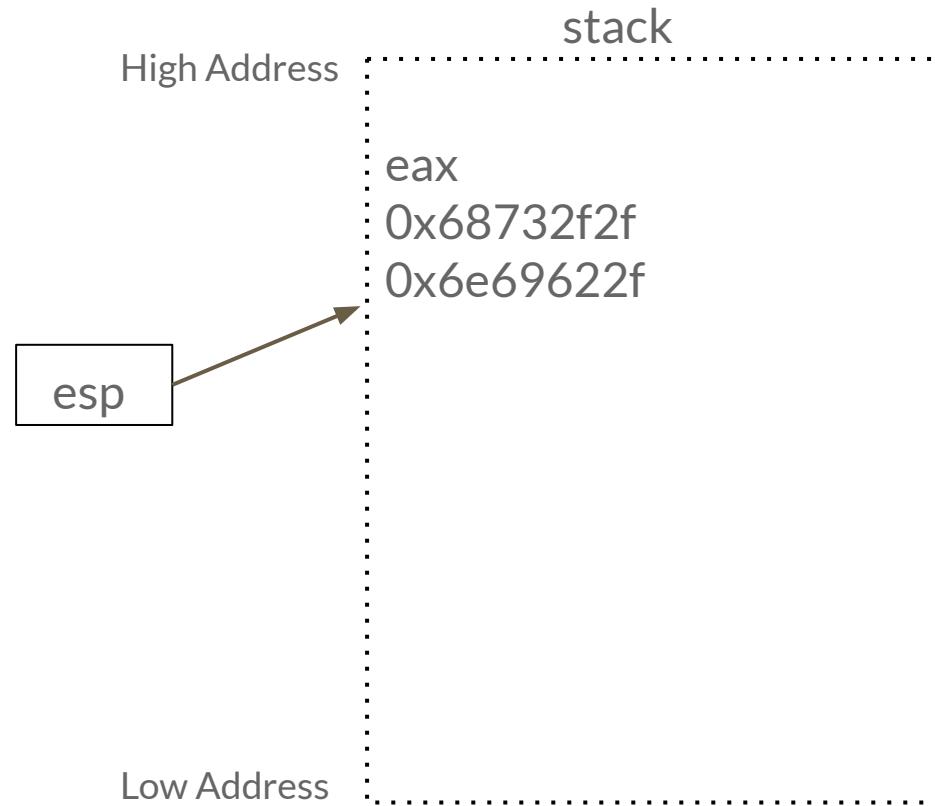
# Making a System Call in x86 Assembly

```
xor eax,eax  
push eax  
push 0x68732f2f  
push 0x6e69622f  
mov ebx,esp  
push eax  
push ebx  
mov ecx,esp  
mov al,0xb  
int 0x80
```



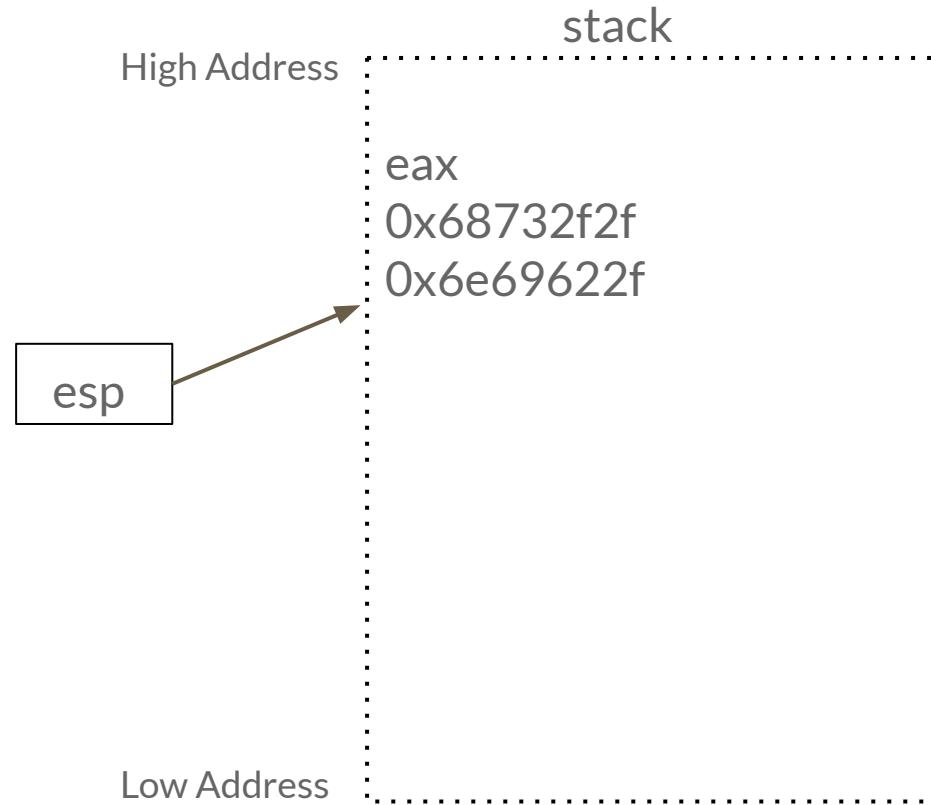
# Making a System Call in x86 Assembly

```
xor eax,eax  
push eax  
push 0x68732f2f  
push 0x6e69622f  
mov ebx,esp  
push eax  
push ebx  
mov ecx,esp  
mov al,0xb  
int 0x80
```



# Making a System Call in x86 Assembly

```
xor eax,eax  
push eax  
push 0x68732f2f  
push 0x6e69622f  
mov ebx,esp  
push eax  
push ebx  
mov ecx,esp  
mov al,0xb  
int 0x80
```



# Making a System Call in x86 Assembly

execve(2)

System Calls Manual

## NAME

execve - execute program

## LIBRARY

Standard C library (libc, -lc)

## SYNOPSIS

```
#include <unistd.h>

int execve(const char *pathname, char *const _Nullable argv[],  
          char *const _Nullable envp[]);
```

/bin/sh, 0x0

0x00000000

Address of /bin/sh, 0x00000000

execve("/bin/sh", address of string "/bin/sh", 0)

# Making a System Call in x86\_64 (64-bit) Assembly

x86\_64 (64-bit)

Compiled from [Linux 4.14.0 headers](#).

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
0	read	<a href="#">man/cs/</a>	0x00	unsigned int fd	char *buf	size_t count	-	-	-
1	write	<a href="#">man/cs/</a>	0x01	unsigned int fd	const char *buf	size_t count	-	-	-
2	open	<a href="#">man/cs/</a>	0x02	const char *filename	int flags	umode_t mode	-	-	-
3	close	<a href="#">man/cs/</a>	0x03	unsigned int fd	-	-	-	-	-
4	stat	<a href="#">man/cs/</a>	0x04	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
5	fstat	<a href="#">man/cs/</a>	0x05	unsigned int fd	struct __old_kernel_stat *statbuf	-	-	-	-
6	Istat	<a href="#">man/cs/</a>	0x06	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
7	poll	<a href="#">man/cs/</a>	0x07	struct pollfd *ufds	unsigned int nfds	int timeout	-	-	-
8	Iseek	<a href="#">man/cs/</a>	0x08	unsigned int fd	off_t offset	unsigned int whence	-	-	-
9	mmap	<a href="#">man/cs/</a>	0x09	?	?	?	?	?	?
10	mprotect	<a href="#">man/cs/</a>	0x0a	unsigned long start	size_t len	unsigned long prot	-	-	-
11	munmap	<a href="#">man/cs/</a>	0x0b	unsigned long addr	size_t len	-	-	-	-
12	brk	<a href="#">man/cs/</a>	0x0c	unsigned long brk	-	-	-	-	-
13	rt_sigaction	<a href="#">man/cs/</a>	0x0d	int	const struct sigaction *	struct sigaction *	size_t	-	-
14	rt_sigprocmask	<a href="#">man/cs/</a>	0x0e	int how	sigset_t *set	sigset_t *oset	size_t sigsetsize	-	-
15	rt_sigreturn	<a href="#">man/cs/</a>	0x0f	?	?	?	?	?	?
16	ioctl	<a href="#">man/cs/</a>	0x10	unsigned int fd	unsigned int cmd	unsigned long arg	-	-	-

[https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32\\_bit](https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit)

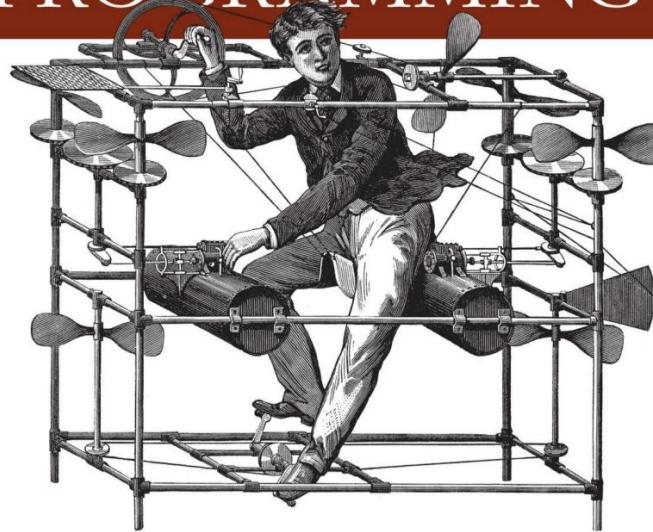
# Making a System Call in x86\_64 (64-bit) Assembly

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
59	execve	<a href="#">man/ cs/</a>	0x3b	const char *filename	const char *const *argv	const char *const *envp	-	-	-

```
push rax  
xor rdx, rdx  
xor rsi, rsi  
mov rbx, '/bin//sh'  
push rbx  
push rsp  
pop rdi  
mov al, 59  
syscall
```

SYSTEM AND LIBRARY CALLS EVERY PROGRAMMER NEEDS TO KNOW

# LINUX SYSTEM PROGRAMMING



O'REILLY®

ROBERT LOVE

# Background Knowledge: System Calls

# Channels of Communication for Linux Process

Every process in Linux has three initial, standard channels of communication:

- Standard Input (stdin, fd=0) is the channel through which the process takes input.  
For example, your shell uses Standard Input to read the commands that you input.
- Standard Output (stdout, fd=1) is the channel through which processes output normal data, such as the flag when it is printed to you in previous challenges or the output of utilities such as ls.
- Standard Error (stderr, fd=2) is the channel through which processes output error details. For example, if you mistype a command, the shell will output, over standard error, that this command does not exist.

# Examples

## Redirecting output > or 1>

```
echo hi > asdf echo hi 1> asdf
```

## Appending output >>

```
echo hi >> asdf
```

## Redirecting errors 2>

```
/challenge/run 2> errors.log
```

## Redirecting input <

```
rev < messagefile
```

# Channels of Communication for Linux Process

Process can also take input from command line arguments

*ls -al c*

*at /flag*

*cat 1.txt 2.txt 3.txt*

# Pipe

The | (pipe) operator. Standard output from the command to the left of the pipe will be connected to (piped into) the standard input of the command to the right of the pipe.

```
echo hello-world | wc -c
```

# Background Knowledge: Environment and Shell Variables

# Environment and Shell Variables

Environment and Shell variables are a set of dynamic **named values**, stored within the system that are used by applications launched in shells.

KEY=value

KEY="Some other value"

KEY=value1:value2

The names of the variables are case-sensitive (UPPER CASE). Multiple values must be separated by the colon : character. There is no space around the equals = symbol.

# Environment and Shell Variables

Environment variables are variables that are available **system-wide** and are **inherited** by all spawned child processes and shells.

Shell variables are variables that apply only to the **current shell instance**. Each shell such as zsh and bash, has its own set of internal shell variables.

# Common Environment Variables

**USER** - The current logged in user.

**HOME** - The home directory of the current user.

**EDITOR** - The default file editor to be used. This is the editor that will be used when you type edit in your terminal.

**SHELL** - The path of the current user's shell, such as bash or zsh.

**LOGNAME** - The name of the current user.

**PATH** - A list of directories to be searched when executing commands.

**LANG** - The current locales settings.

**TERM** - The current terminal emulation.

**MAIL** - Location of where the current user's mail is stored.

# Commands

**env** – The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.

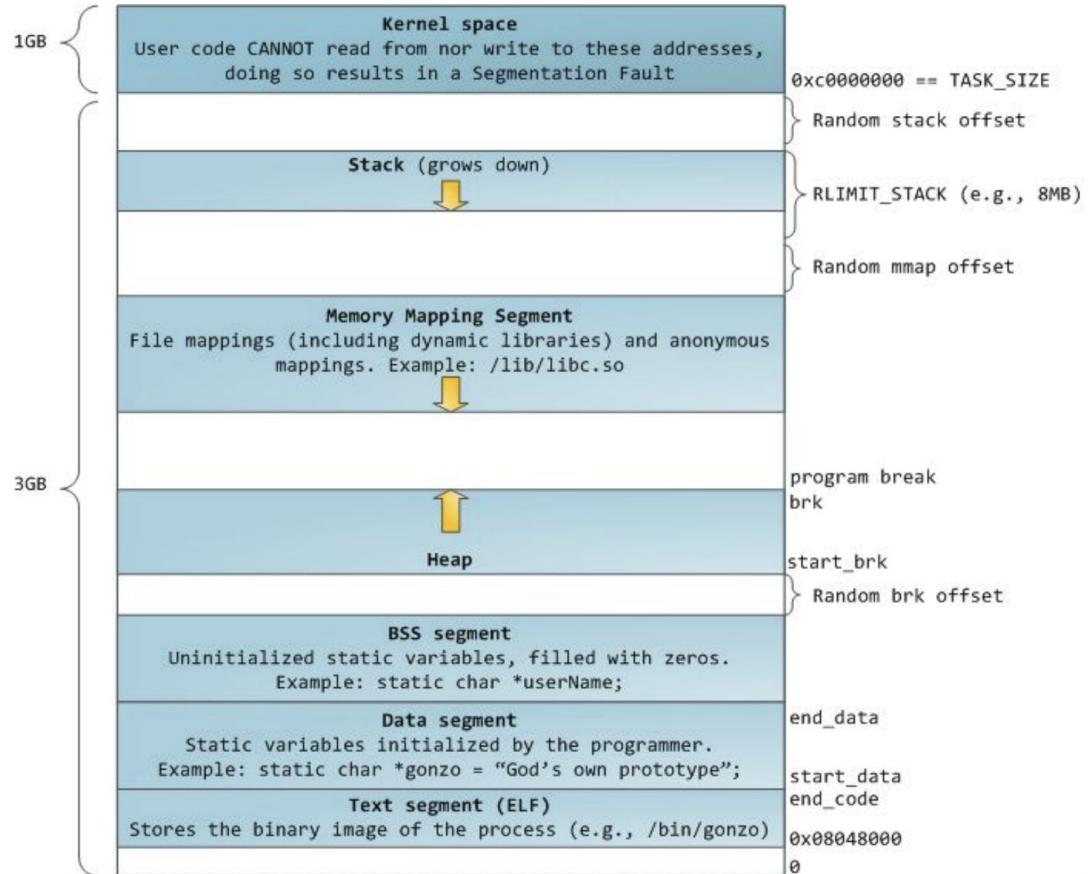
**printenv** – The command prints all or the specified environment variables.

**set** – The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.

**unset** – The command deletes shell and environment variables.

**export** – The command sets environment variables

The environment variables live towards the top of the stack, together with command line arguments



# Background Knowledge: Manual Binary Analysis Tools

# Tools for this class

*file*

*readelf*

*strings*

*nm*

*objdump*

*GDB*

[optional] *IDA Pro*

[optional] *ghidra*

[optional] *Binary Ninja*

# GDB Cheat Sheet

Start gdb using:

```
gdb <binary>
```

Pass initial commands for gdb through a file

```
gdb <binary> -x <initfile>
```

To start the program and breakpoint at main()

```
start <argv>
```

To start the program and breakpoint at \_start

```
start <argv>
```

To run the program without breakpoint

```
r <argv>
```

Use another program's output as stdin in GDB:

```
r <<< $(python2 -c "print '\x12\x34'*5")
```

# GDB Cheat Sheet

Set breakpoint at address:

`b *0x80000000`

Set breakpoint at beginning of a function:

`b main`

....

`b <filename:line number>`

`b <line number>`

Disassemble 10 instructions from an address:

`x/10i 0x80000000`

Exam 15 dword (w) from an address; show hex (x):

`x/15wx 0x80000000`

Exam 3 qword (g) from an address; show hex (x):

`x/3gx 0x80000000`

# GDB Cheat Sheet

To show breakpoints

`info b`

To remove breakpoints

`clear <function name>`

`clear *<instruction address>`

`clear <filename:line number>`

`clear <line number>`

# GDB Cheat Sheet

Use “examine” or “x” command

`x/32xw <memory location>` to see memory contents at memory location, showing 32 hexadecimal words

`x/5s <memory location>` to show 5 strings (null terminated) at a particular memory location

`x/10i <memory location>` to show 10 instructions at particular memory location

See registers

`info reg`

Step an instruction

`si`

# GDB Script

Use “examine” or “x” command

`x/32xw <memory location>` to see memory contents at memory location, showing 32 hexadecimal words

`x/5s <memory location>` to show 5 strings (null terminated) at a particular memory location

`x/10i <memory location>` to show 10 instructions at particular memory location

See registers

`info reg`

Step an instruction

`si`

# Shell Cheat Sheet

Run a program and use another program's output as a parameter

```
program $(python2 -c "print '\x12\x34'*5")
```

# Reading

1. <https://iq.thc.org/how-does-linux-start-a-process>