CS444/544 Operating Systems II

Prof. Sibin Mohan

Spring 2022 | Lec 9: System Calls and Page Faults

Adapted from content originally created by: Prof. Yeongjin Jang

Recap | System Calls, printf()?



We're not done with printf() though!



Today's Topic

More about System Calls

• Privilege separation and call gate

Page Faults

- OS fault handling and resume execution?
- For what purpose?
 - Automatic stack allocation
 - Copy-on-write
 - Swap

Why have Privilege Separation?

5

- Security!
- We do not know what application will do
 - Dangerous operations
 - Flash BIOS, format disk, deleting system files, etc.
 - Let only OS access hardware
 - Apply access control for hardware resources!
 - E.g., only the administrator can format disk

• OS mediates hardware access **System Calls!**



User Level [Ring 3]

 \bigcirc

Library vs System Calls

• Library Calls

- APIs available in Ring 3
- **Do not** include operations in Ring 0
- Cannot access hardware directly
- Could be a **wrapper** for
 - some computation or
 - for system calls
 - E.g., printf() internally uses write()

6





Library Calls > System Calls

7

System Calls

- APIs available in Ring 0
- OS abstraction for hardware interface
- Ring 3 application \rightarrow Ring 0 operations



Library Calls > System Calls

8

- App shouldn't call arbitrary function!
- Else privilege separation meaningless

- Apps/libraries can invoke system calls
- But no other kernel functions!



Library Calls -> System Calls ---- GATE!

9

- App shouldn't call arbitrary function!
- Else privilege separation meaningless

- Apps/libraries can invoke system calls
- But no other kernel functions!



Secure System Call Design: Call Gate via Interrupt Handling

• Call gate: a secure method to control access to Ring 0!



- Call Gate
- System call invoked
 - only with trap handler
 - int \$0x30 in JOS
 - int \$0x80 in Linux [32-bit]
 - int \$0x2e in Windows [32-bit]



- Call Gate
- System call invoked
 - only with trap handler
 - int \$0x30 in JOS
 - int \$0x80 in Linux [32-bit]
 - int \$0x2e-in Windows [32-bit]

program executior \rightarrow interrupt	\rightarrow call trap gate	\rightarrow create/store trap frame	ightarrow load trap handler in mem	\rightarrow execute
Pop context → iret → resume	execution	···	< <i>`</i>	
L/	J	Ον	erheads!	

- Call Gate
- System call invoked
 - only with trap handler
 - int \$0x30 in JOS
 - int \$0x80 in Linux [32-bit]
 - int \$0x2e-in Windows [32-bit]

sysenter/sysexit(32-bit)
syscall/sysret(64-bit)
10x faster than ints

 program executior \rightarrow interrupt
 \rightarrow call trap gate
 \rightarrow create/store trap frame
 \rightarrow load trap handler in mem
 \rightarrow execute

 Pop context
 \rightarrow iret
 \rightarrow resume execution

 Overheads!

10s of thousands of cycles!

- Call Gate
- System call invoked
 - only with trap handler
 - int \$0x30 in JOS
 - int \$0x80 in Linux [32-bit]
 - int \$0x2e-in Windows [32-bit]
 - sysenter/sysexit (32-bit)
 - syscall/sysret (64-bit)



int 0x30

- OS performs checks
 - if userspace app/lib is doing a right thing
 - Before performing important ring 0 operations



Protecting Syscalls via Call Gate

• Consider the 'read()' system call?

- read(int fd, void *buf, size_t count)
- Read count bytes from a file pointed by fd and store those in buf

Usage

```
// buffer at the stack
char buf[512];
// read 512 bytes from standard input
read(0, buf, 512);
```

Protecting Syscalls via Call Gate

// kernel address will points to a dirmap of // the physical address at 0x100000 char kernel_address = KERNBASE + 0x100000; // read 512 bytes from standard input read(0, buf, 512);

- This will **overwrite kernel code** with your keyboard inputs!!!
 - Changing kernel code from Ring 3 is possible!

Use the Call Gate!

• Call gate: a secure method to control access to Ring O!



Use the Call Gate!

4/28/22

• Call gate: a secure method to control access to Ring O!



Test

// buffer at the stack
char buf[512];
// read 512 bytes from standard input to stack
int ret = read(0, buf, 512);

printf("Read to stack memory returns: %d\n", ret);

// read 512 bytes from standard input to kernel memory
ret = read(0, (void *) 0xfffffff01000000, 512);

printf("Read to kernel memory returns: %d\n", ret);
perror("Reason for the error:");

[blue9057@blue9057-vm-jos ~/test\$] ./a
asdfzxcv
Read to stack memory returns: 9
Read to kernel memory returns: -1
Reason for the error:: Bad address

Check How System Calls are Invoked in Linux Kernel

- Use strace in Linux
- e.g., \$ strace /bin/ls

read(0, "asdfzxcvn", 512) = 9 $fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0$ $= 0 \times 18 \times 5000$ brk(NULL) brk(0x18e6000) $= 0 \times 18e6000$ write(1, "Read to stack memory returns: $9\n$ ", 32) = 32read(0, 0xffffff01000000, 512) = -1 EFAULT (Bad address)write(1, "Read to kernel memory returns: -"..., 34) = 34 dup(2)= 3fcntl(3, F_GETFL) = 0x8001 (flags 0_WRONLY | 0_LARGEFILE) close(3) = 0 write(2, "Reason for the error:: Bad addre"..., 35Reason for the error:: Bad address Summary: System Call / Call Gate

- Prevent Ring 3 from accessing hardware directly
 - Security reasons!
 - OS mediates hardware access via system calls
- System calls → APIs of an OS
- Prevent application from running arbitrary ring 0 operation?
 - Call gate
- Modern OSes use call gates to protect system calls
 - trap handler \rightarrow OS **applies access control** for system call

036021 0.40 186 .689 600 ols . 07 .1 541.48 50 SI 10° - 00 12.926 80. 75.278 5. 13:291 52.625 10.001 18.988

19.973

291

5

2.514

16.55

92.908

25.08

10.102

122

3.

32.029

38.713

20.282

12.101

01:01

7.321

21

2.291,94

0.088

21.661

19.430

293

17.653

51.919

63.81

83.155

69.193

73:01

00

5. 5.

75.99

2.

34.609

2

7.170

30.932

201

23.121

00.

シ .62

100

2

500

S.

02.00

cil

6

00

5

U

15

.0

.....

· 26

N

2

3

J. C.L

220,25

.6

.Su

50.00

602

860.21

23.03

695

57.10.

.c.2

20

.18

120

VLV V

224

0.

126.96

.cl

c1.09

10.07

·vo.

681.98

2

19.11

0.00

.0.

520.000

100

Coo Coo

050

.ch

12

120.00

2.0.00

VE?

·28

.00

50.7°

.2

.ch

.20

102

25000

oly

S.

SSY

100 V2 607

204

20

10.

·62.

265

· V .c.8

619

OCY

16.91

10

.S

12

696.97

6.62

I'V

0360 050 120.27 · v

.00.00 00.00

.8

5

12.48

1

.10

642 01-02

10.0L

.2

29011 55

.2

01002

200.84

.02

80

.20

818.14

0.91

001.98

ist

206.26

2.20

LYC

c.99.16

29

7IG

.84

202

.~

91.9

.50

CVL

.2

200.11

224

889.11

.00

620-20

264

SVE

609

5.

286

.02

11

096.21

.du

153-16

So.

068.91

es

810

612 61.09

055.16

679.1V

6.

2.

686

.20

00

·20

16.62

65.77

2002

127.18

.20

.51

6.

10 V

.6

12. 12. 19. 12

0.0

.0

00.00

8.0

12800

.63.

640

162

250

.00

62 2.0

ese ll

3

20

.2

20 .07

.2

105:16

.20

50.01 808

ST

.19

101

520

er

.8

822

5-5-5 P

0.

128

V2614

2

570

9999

.2

25

192.52

250

¥1651

1

N.S.S.

.Su

000

.2

000

241.65

.0

202 269.92

.ve

60.62

.0

.08

652.05

set

12.00

6CV

629.68

20

cost

.01

200

13.00

.02

002

.20

262

10

S

.56

E.

.00

200

Lest.

200

280-18

.20

11V

.5

050

.02

264

09.131

520

168

.20

77

2

520.02

1CV

101.28

.ch

150

.5

10.18

S.

600 87.61

.60

250

63.032

00

29.131

219.19

.56

521

S

282

.5

SV

1000

208

.2

690 5.71

2.

519

.0

212.01

637

.2

250,04

860

.24

V68-28

8.

Sch

.25

120.01

51-28

\$

10.0

.cd

WIL

01614

44.

626

1000

171.08

010

.02

09.

.22

54.52

.60

050

101-28

.00

603.18

000-151

.0

.29

10.00 266

00-69

818

N.

10:00

280

.5

150

~

W.V.S

.08

eve

686-1

.cs

6.

872

.

.8

.0

0.00

Sch

.5

20,5

S 6 £10.0 26.00

.c.

CC-10

128.57

.11

252

.0

Ser

V 168 025.02

à.

650 .00

192.68

159.10

60.69

2

.c.

1.12

. 8º

50.02

612.19

2

81-251

V Ver 114.14

.2

27.21

100

WW 8.

510.22

.18

012

.00

0.05

11005

996.92

162

11.00

506.52

V05.24

15

.0.

510 VI 896

29.99

.c.

851-14

688

.05

.20

4 CC 809.18

854

12:09

592.24

.09

-5-1-6-

11V

500.050

.0.57 77

.8

012

250

.12

0500

101

10.

52.625

30

16.60

1 AS

S.

16.372

6.129

67.672

50.

2º

75.278

216

80.

23.765

51.322

73.76

No.

38.161

2

901

10.

176

30.32

06.924

0.109

AA.TA

2

36.55

rad

18.

6.

25

1.100

973

3.

63.91

69.593

291

61.

2.54

10.500

032.00

105

1.

321

85.08

2.

024

122

N

32.029

33

38.713

688

20.282

12.101

d'

198

0.

23.323

72.036

76.198

17.653

51.949

83.150

1.

73:01

102

10.34

15

1.

81.201

23.121

200

55

1.00

53.026

01.886

5

2.

N-502

MA

8.

94.933

265

00

131

19.50

26.039

111

5

20.13

de

32.11

89. 99.

16.899

23.889

LOS L

Nº.

63.032

29.131

SI

25.50

12.926

32

2

13.291

555

3.

23.22

72.036

51.345

1.000

6.818

0.18

1:05

60

15.026

20.

0,988

2

198

76.198

120

00

8 05.12

.00

256.69

6 Sil

2

.00 2

51.21

1054

8.

0,92

.60

200.76

2057 529

V

V12.09

. So

026.16

cess

Vesi 24

462.46

0.67

æ.

050

VA

.07

10

.0.

.8

009 90.07

.~

671

50.02

8

.80

1951

-55

192.68

44

0.

128

.26

020 106.04

250.00

.05

53.026

23.765

921

0.109

NA.TA

311

S.

298.25

2000

67.859

25.013

72.098

200

13.65

28.001

12.518

22.089

180

53.53

32.121

139.821

71.173

51.322

23.

36.551

110

10.

38.161

0.60

A1.717

24.584

91.674

10.110

57.322

ngo

16.600

56.265

89.

5.151

2.

13.994

0.033

on.

2.

13.39

00

84.191

51.5

on.

30.

SIA

21.00

0.6.019

95.052

328

2.

6.

69.932

199. 199.

10

58.21

37.912

51.381

22.

9.012

VV.

17.11

2.

51.734

51.

1.014

31.

200

57.199

M

30.534

0

2.02.

oly

.00

602

Sp.

OLY

.cs

820

225.15

0.94

00.

158.10

• **v**

032

121.09

1112 26

-c8

c16

1.69

412.08

52

N

.6

S.

.0.5 577

250

250

60.65

6 610 111.05

.18

WW

5

000000

.28

012

200

N:02

91.833

29.889

2.

300

003

21.599

.8

205

69.057

11005

9960

162

con SYS

.22

162

851-14

.2

005

.c.

11000

12 M 15.92

698

.20

0200

SIL

.c.

520

105.2V

40.

89.99

C10.V1

10

VS 860

10.00

80.97

OLY

.ch

0.4.0

012

10.00

.c.>

220 45

0.91

.0,

2300

80.

6.92 0.09

.5

.5

01.9V

1054

.0

35

Se.

205.2

.

529.25

812

.20

19Y

N.

27

.cl

102

00.

250.00

.0

100

6.

051

81.859

292

8.050

39.774

73.991

691

91.255

82.891

12.003

33.31

23.164

13.003

29.608

250

85.005

34.68

d'

26.554

17.121

15

17.609

2.510

01.

14.595

09.683

72.732

av

VIN

602.0

128

.cb

020

30.

721

5.

1.285

513

86.11

89:39

15.982

67.301

14.661

51.014

15.991

155

6.

So

v.

00.736

82.623

5.

510

66.281

293

16.911

1.718

10.

63.129

92.179

31.59

31.12

20.

63.140

00.

60.103

8.

09.538

Nº.

85.

23

S.

55.

10.

03.68

63.575

51.081

84.745

69.306

55.57

52.

150.18

121-230

29.013

2.

76.76

81.375

21.221

51.651

13.278

02.199

N7.078

39.695

16.021

152.098

10:32

23.662

10.139

óp.

08.013

68.683

23.972

D.

6

20

100

8

42.

200

.2

.09

619

.0

128.09

6.

64.19

V62

.20

800

5.61

NYS.

.2

.2

259.11

2U

65 864

.6

864

83.152

110

30.932

96.924

0.

22

MA.AT

5. 5. 50 5. 50 . 50

609

61 - 0PM

3

10

25

de de

2

0

3.

63.81

19:593

So.

10.102

50.00

73.601

81.201

S.

75.99

291

5.

514

10.50

92.908

5.

7.321

0.088

21.664

19.39

34.

10.000

122

5.

32.029

38.13

198

2.

23.323

72.036

57.345

14.000

8-9-10 B.

1.605

15.020

67.217

2.291

96.194

76.293

032

S.

20.039

15.11

20.13

32.11

20.20

16.899

20.1.48

76.561

78.152

86.56

760

A.878

51.10

77.355

31.121

P

14.29

3

29.03

d'

10.00

051

2.

73.971

28.12

20

28.00

300

6.

0

3

81.617

2.217

12.962

13.013

22.211

65.634

3.

18.971

S.

131

09.

16.372

6.129

67.672

81.91

55.194

293

22.682

05.751

66.519

8.519

12.56

91.883

91.891

81.382

31.018

31.251

28.119

62.311

nº.

92.992

10.86

39.76

7.579

Sr.

1.50

2.

84.511

66.986

16.020

25

8.

83.101

77.916

80.11

61.61.5

A1.754

153

2

82.101

10.706

76.693

8.202

59.12

15.011

232

23.264

13.088

11.924

821

0.

629

228

P.

3

64.555

83.579

33

13.051

1.50

70.814

N1-819

01.550

18.003

70.310

17.052

19:548

29.131

76.198

1.653

2.

20.282

12.07

91:501

51.919

Handling Faults | Page Fault

- Faulting instruction has not executed [e.g., page fault]
- Resume execution after handling the fault

Page Fault: A Case of Handling Faults

• Occurs when paging [address translation] fails

- ! (pde & PTE_P) or ! (pte & PTE_P) → invalid translation
- Write access but ! (pte & PTE W) → access violation
- Access from user but ! (pte & PTE U) → protection violation

Page Fault: an Example

Accessing a Kernel address from User



TRAP frame at 0xf01c0000 0x00000000 edi 0x00000000 esi 0xeebfdfd0 ebp oesp 0xeffffdc 0x00000000 ebx edx 0x00000000 ecx 0x00000000 eax 0xeec00000 0x---0023 es 0x---0023 ds trap 0x0000000e Page Fault cr2 0xf0100064 0x00000007 [user, write, protection] eip 0x00800039 0x - - - 001bCS flag 0x00000096 0xeebfdfb8 esp 0x----0023 SS [00001000] free env 00001000

Page Fault: What Does CPU Do?

CPU informs OS → why and where a page fault happened

- CR2: stores the address of the fault
- Error code: stores the reason of the fault CR2

TRAP	frame	at 0xf0	1c0000		
edi	0x0	0000000			
esi	0x0	0000000			
ebp	o Oxee	ebfdfd0			
oes	sp 0xe [.]	ffffdc			
eb>	x 0x0	0000000			
ed>	x 0x0	0000000			
ec>	x 0x0	0000000			
ea>	c Oxee	ec00000			
es	0x-	0023			
ds	0x-	0023			
tra	ap 0x00	000000e	Page Fa	ault	
cr2	2 0xf	0100064			
eri	- 0x0	0000007	[user,	write,	protection]
eip) 0x0	0800039			
cs	0x-	001b			
fla	ag 0x0(0000096			
esp	o Oxee	ebfdfb8			
SS	0x-	0023			
[00001000] free env 00001000					

|kernel_memory[100] = '!';

31	
	Page fault virtual address 0xf0100064
31	15 543210
	Reserved Reserved Reserved
Ρ	 0 The fault was caused by a non-present page. 1 The fault was caused by a page-level protection violation.
W/R	0 The access causing the fault was a read.1 The access causing the fault was a write.
U/S	0 A supervisor-mode access caused the fault.1 A user-mode access caused the fault.
RSVD	 The fault was not caused by reserved bit violation. The fault was caused by a reserved bit set to 1 in some paging-structure entry.
I/D	0 The fault was not caused by an instruction fetch.1 The fault was caused by an instruction fetch.
PK	0 The fault was not caused by protection keys.1 There was a protection-key violation.
SGX	 0 The fault is not related to SGX. 1 The fault resulted from violation of SGX-specific access-control

requirements

CPU/OS Execution Example

- User program accesses 0xf0100064
- CPU generates **page fault** (pte & PTE_U == 0)
 - Stores the faulting address in CR2
 - Put out an error code
 - Calls page fault handler in IDT
- OS: calls page_fault_handler
 - Read CR2 [address of the fault, 0xf0100064]
 - Read error code [reason of the fault]
 - Resolve error [if not possible, destroy the environment]
 - Continue user execution
- User: **resume** instruction in CR2 [or destroyed by the OS]

Fault Resume Example: Stack Overflow

- inc/memlayout.h
- We (initially) allocate one [1] page [4 kb] for the user stack



• If you use a large local variable on the stack

• Stack overflow! Page Fault!

4/28/22

int func() {

char buf[8192];

buf[0] = '1';



Allocating New Stack Automatically

- Detect such an access?
- Allocate a **new page** for the stack **automatically**?
- Yes!
- 'Page Fault'
- Observations
 - Stack overflow is sequential \rightarrow access pages adjacent to stack
 - We should catch both read/write access \rightarrow both should fault



Example: New Stack Allocation by Fault (CPU)

- Lookup page table → no translation!
- Store **Oxeebfcff0** in the CR2 register
- Set error code
 - "The fault was caused by a page that wasn't present!"
- Raise page fault exception [interrupt #14] → call page fault handler



Example: New Stack Allocation by Fault (OS)

- Interrupt will force CPU to invoke the page_fault_handler()
- Read CR2
 - **Oxeebfcff0**, the page right next to current stack end
 - The current stack end is: **0xeebfd000**
- Read error code
 - "The fault was caused by a page that wasn't present!"
- Let's allocate a new page for the stack!

ller() Oxeebfe000	
0xeebfd000	STACK
	No mapping!

Example: New Stack Allocation using Fault

• Allocate a new page for the stack 0xeebfe000 • struct PageInfo *pp = page alloc(ALLOC ZERO); STACK • Get a new page, and wipe it to zero all its contents 0xeebfd000 • page_insert(env_pgdir, pp, **0xeebfc000**, PTE U|PTE W); • Map a new page to that address! 0xeebfc000 • iret! **NEW STACK PAGE**

New Stack Allocation Using Fault (User-Return)



Other Useful Examples of Using Page Fault (in Modern OSes)

• Copy-on-Write [CoW]



4/28/22

- Share pages read-only
- Create a private copy when the first write access happens



- Share pages read-only
- Create a private copy when the first write access happens



- Share pages read-only
- Create a private copy when the first write access happens



- Share pages read-only
- Create a private copy when the first write access happens



- Share pages read-only
- Create a private copy when the first write access happens



- Share pages read-only
- Create a private copy when the first write access happens



- Share pages read-only
- Create a private copy when the first write access happens

Fork!



Memory Swapping

- Use disk as extra space for physical memory
- Limited RAM Size: 16/32/64 GB?
- We have a bigger storage: 1T SSD Hard Disk, cloud storage, etc.
- Store some 'currently unused but will be used later' pages in the disk
- Store only the active part of data in memory

Copy-on-Write (CoW) to Reduce Memory Footprint

- os2 server
- Will run many /bin/bash, /usr/bin/gdb, /usr/bin/tmux, etc.
 - Each of you will run those programs!
 - Do we need to have 826 copies of the same program in memory?
- Build an OS to **efficiently** manage programs and **minimize** memory usage?
 - Share physical pages of the same program!

```
[jangye@os2 ~$] ps aux | grep bash | wc -l
826
[jangye@os2 ~$] ps aux | grep tmux | wc -l
128
[jangye@os2 ~$] ps aux | grep gdb | wc -l
90
```

Count number of processes running bash, tmux, and gdb

A Program's Memory Layout [ELF]

.text	 Code area. Read-only and executable 	.bss [RW-]
.rodata	 Data area, Read-only and not executable 	.data [RW-]
.data	 Data area, Read/Writable (not executable) Initialized by some values 	.rodata [R]
.bss	 uninitialized data Data area, Read/Writable (not executable) Initialized as 0 	.text [R-X]



Sharing by Read-only

Create Page Directory and copy entries!

• Set page table to map to same physical address to share contents



OK for Read-only Sections

• How can Process 1 write into .bss?



Page Fault Handler

- Read CR2
- An address that is in the page cache
- Fault from a shared location!
- Read Error code
 - Write on read-only memory
 - **Process requires a private copy**! [we mark if CoW is required in PTE]
- ToDo: create a writable, private copy for that process!
 - Map a new physical page [page_alloc, page_insert]
 - Copy contents
 - Mark as read/write
 - Resume

Copy on Write

• How can Process 1 write into .bss?









Benefits? using page faults!

Better performance!

 reduce time for copying contents already in physical memory (page cache)

More **efficient**!

- reduce physical memory use
- sharing code/read-only data among multiple processes
- 1,000,000 processes, requiring only 1 copy of .text/.rodata

Additional benefits

- Can support sharing of writable pages [unless modified]
- Can create private pages seamlessly on write

Memory Swapping





Challenge

- Suppose you have 8GB of main memory
- Can you run a program that is 16GB in size?
 - Yes, you can manually load it one part at a time
 - we do not use all of data at the same time
- OS do this **seamlessly** [**transparently**] for application?





Swapping | OS

- Page fault handler
 - Read CR2
 - get address [0xf020000]
 - Read error code
- If error code \rightarrow page not present fault **and**
- faulting page is stored in the disk
- Lookup disk if it swapped out 0xf0200000 of this environment [process]
 - This must be per process because virtual address is per-process resource
- Load that page into physical memory
- Map it and then continue!



.

Page Fault | Summary

- Generated for memory errors [during paging]
- A recoverable exception
- User program may resume the execution
- Is useful for implementing
 - Automatic stack allocation
 - Copy-on-write (will do in Lab4)
 - Swapping

4/28/22

Backup Slides

Check How Library Calls are Invoked

- Use ltrace in Linux
- e.g., \$ ltrace /bin/ls

```
read(0, "asdfzxcv\n", 512) = 9
printf("Read to stack memory returns: %d"..., 9) = 32
read(0 <no return ...>
error: maximum array length seems negative
, "", 512) = -1
printf("Read to kernel memory returns: %"..., -1) = 34
perror("Reason for the error:"Reason for the error:: Bad address
) = <void>
+++ exited (status 0) +++
```