1. *Enhance the hello.c program to open a file, read from the file, write to the file, and close the file. Understand how a system call is invoked and how it works by generating and reading an ASM file. Identify and mark the system calls in your ASM file. Submit your hello.c and ASM files showing the system calls (Use Linux).*
   1. **hello.c**

#include <stdio.h>

#include <stdlib.h>

int main ()

{

// Open file "file.txt"

char file\_name[] = "file.txt";

FILE \*fp = fopen(file\_name, "r+a");

// Check for error opening file

if(fp == NULL)

{

perror("Error opening file.\n");

exit(EXIT\_FAILURE);

}

// Print file contents

printf("File %s opened and reads as follows: \n", file\_name);

char \*line = NULL;

size\_t length = 0;

ssize\_t read;

while((read = getline(&line, &length, fp)) != -1)

{

printf("%s", line);

}

// Write to file

char message[] = "Hello to you too!\n";

printf("Writing to file: %s", message);

fprintf(fp, message);

// Close file

fclose(fp);

// Cleanup

if(line != NULL)

free(line);

// Return

return 0;

}

* 1. **file.txt before run**

Hello World!

* 1. **file.txt after run**

Hello World!

Hello to you too!

* 1. **hello.s (ASM file)**

.file "hello.c"

.section .rodata

.LC0:

.string "r+a"

.LC1:

.string "Error opening file.\n"

.align 8

.LC2:

.string "File %s opened and reads as follows: \n"

.LC3:

.string "%s"

.LC4:

.string "Writing to file: %s"

.text

.globl main

.type main, @function

main:

.LFB2:

.cfi\_startproc

pushq %rbp

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.cfi\_def\_cfa\_register 6

subq $80, %rsp

movabsq $8392585648223840614, %rax

movq %rax, -32(%rbp)

movb $0, -24(%rbp)

leaq -32(%rbp), %rax

movl $.LC0, %esi

movq %rax, %rdi

call fopen

movq %rax, -8(%rbp)

cmpq $0, -8(%rbp)

jne .L2

movl $.LC1, %edi

call perror

movl $1, %edi

call exit

.L2:

leaq -32(%rbp), %rax

movq %rax, %rsi

movl $.LC2, %edi

movl $0, %eax

call printf

movq $0, -40(%rbp)

movq $0, -48(%rbp)

jmp .L3

.L4:

movq -40(%rbp), %rax

movq %rax, %rsi

movl $.LC3, %edi

movl $0, %eax

call printf

.L3:

movq -8(%rbp), %rdx

leaq -48(%rbp), %rcx

leaq -40(%rbp), %rax

movq %rcx, %rsi

movq %rax, %rdi

call getline

movq %rax, -16(%rbp)

cmpq $-1, -16(%rbp)

jne .L4

movabsq $8031079698440938824, %rax

movq %rax, -80(%rbp)

movabsq $8029764343382898976, %rax

movq %rax, -72(%rbp)

movw $2593, -64(%rbp)

movb $0, -62(%rbp)

leaq -80(%rbp), %rax

movq %rax, %rsi

movl $.LC4, %edi

movl $0, %eax

call printf

leaq -80(%rbp), %rdx

movq -8(%rbp), %rax

movq %rdx, %rsi

movq %rax, %rdi

movl $0, %eax

call fprintf

movq -8(%rbp), %rax

movq %rax, %rdi

call fclose

movq -40(%rbp), %rax

testq %rax, %rax

je .L5

movq -40(%rbp), %rax

movq %rax, %rdi

call free

.L5:

movl $0, %eax

leave

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

.LFE2:

.size main, .-main

.ident "GCC: (GNU) 4.8.3 20140911 (Red Hat 4.8.3-9)"

.section .note.GNU-stack,"",@progbits

1. *Use the above hello.exe file and objdump command to create an asm file in Linux and mark all system calls in this program. Notice that some are system calls and some are local calls in the asm file. System calls have UND symbols.*
   1. **hello objdump**

hello.o: file format elf64-x86-64

SYMBOL TABLE:

0000000000000000 l df \*ABS\* 0000000000000000 hello.c

0000000000000000 l d .text 0000000000000000 .text

0000000000000000 l d .data 0000000000000000 .data

0000000000000000 l d .bss 0000000000000000 .bss

0000000000000000 l d .rodata 0000000000000000 .rodata

0000000000000000 l d .note.GNU-stack 0000000000000000 .note.GNU-stack

0000000000000000 l d .eh\_frame 0000000000000000 .eh\_frame

0000000000000000 l d .comment 0000000000000000 .comment

0000000000000000 g F .text 0000000000000126 main

0000000000000000 \*UND\* 0000000000000000 fopen

0000000000000000 \*UND\* 0000000000000000 perror

0000000000000000 \*UND\* 0000000000000000 exit

0000000000000000 \*UND\* 0000000000000000 printf

0000000000000000 \*UND\* 0000000000000000 getline

0000000000000000 \*UND\* 0000000000000000 fprintf

0000000000000000 \*UND\* 0000000000000000 fclose

0000000000000000 \*UND\* 0000000000000000 free

Disassembly of section .text:

0000000000000000 <main>:

0: 55 push %rbp

1: 48 89 e5 mov %rsp,%rbp

4: 48 83 ec 50 sub $0x50,%rsp

8: 48 b8 66 69 6c 65 2e movabs $0x7478742e656c6966,%rax

f: 74 78 74

12: 48 89 45 e0 mov %rax,-0x20(%rbp)

16: c6 45 e8 00 movb $0x0,-0x18(%rbp)

1a: 48 8d 45 e0 lea -0x20(%rbp),%rax

1e: be 00 00 00 00 mov $0x0,%esi

23: 48 89 c7 mov %rax,%rdi

26: e8 00 00 00 00 callq 2b <main+0x2b>

2b: 48 89 45 f8 mov %rax,-0x8(%rbp)

2f: 48 83 7d f8 00 cmpq $0x0,-0x8(%rbp)

34: 75 14 jne 4a <main+0x4a>

36: bf 00 00 00 00 mov $0x0,%edi

3b: e8 00 00 00 00 callq 40 <main+0x40>

40: bf 01 00 00 00 mov $0x1,%edi

45: e8 00 00 00 00 callq 4a <main+0x4a>

4a: 48 8d 45 e0 lea -0x20(%rbp),%rax

4e: 48 89 c6 mov %rax,%rsi

51: bf 00 00 00 00 mov $0x0,%edi

56: b8 00 00 00 00 mov $0x0,%eax

5b: e8 00 00 00 00 callq 60 <main+0x60>

60: 48 c7 45 d8 00 00 00 movq $0x0,-0x28(%rbp)

67: 00

68: 48 c7 45 d0 00 00 00 movq $0x0,-0x30(%rbp)

6f: 00

70: eb 16 jmp 88 <main+0x88>

72: 48 8b 45 d8 mov -0x28(%rbp),%rax

76: 48 89 c6 mov %rax,%rsi

79: bf 00 00 00 00 mov $0x0,%edi

7e: b8 00 00 00 00 mov $0x0,%eax

83: e8 00 00 00 00 callq 88 <main+0x88>

88: 48 8b 55 f8 mov -0x8(%rbp),%rdx

8c: 48 8d 4d d0 lea -0x30(%rbp),%rcx

90: 48 8d 45 d8 lea -0x28(%rbp),%rax

94: 48 89 ce mov %rcx,%rsi

97: 48 89 c7 mov %rax,%rdi

9a: e8 00 00 00 00 callq 9f <main+0x9f>

9f: 48 89 45 f0 mov %rax,-0x10(%rbp)

a3: 48 83 7d f0 ff cmpq $0xffffffffffffffff,-0x10(%rbp)

a8: 75 c8 jne 72 <main+0x72>

aa: 48 b8 48 65 6c 6c 6f movabs $0x6f74206f6c6c6548,%rax

b1: 20 74 6f

b4: 48 89 45 b0 mov %rax,-0x50(%rbp)

b8: 48 b8 20 79 6f 75 20 movabs $0x6f6f7420756f7920,%rax

bf: 74 6f 6f

c2: 48 89 45 b8 mov %rax,-0x48(%rbp)

c6: 66 c7 45 c0 21 0a movw $0xa21,-0x40(%rbp)

cc: c6 45 c2 00 movb $0x0,-0x3e(%rbp)

d0: 48 8d 45 b0 lea -0x50(%rbp),%rax

d4: 48 89 c6 mov %rax,%rsi

d7: bf 00 00 00 00 mov $0x0,%edi

dc: b8 00 00 00 00 mov $0x0,%eax

e1: e8 00 00 00 00 callq e6 <main+0xe6>

e6: 48 8d 55 b0 lea -0x50(%rbp),%rdx

ea: 48 8b 45 f8 mov -0x8(%rbp),%rax

ee: 48 89 d6 mov %rdx,%rsi

f1: 48 89 c7 mov %rax,%rdi

f4: b8 00 00 00 00 mov $0x0,%eax

f9: e8 00 00 00 00 callq fe <main+0xfe>

fe: 48 8b 45 f8 mov -0x8(%rbp),%rax

102: 48 89 c7 mov %rax,%rdi

105: e8 00 00 00 00 callq 10a <main+0x10a>

10a: 48 8b 45 d8 mov -0x28(%rbp),%rax

10e: 48 85 c0 test %rax,%rax

111: 74 0c je 11f <main+0x11f>

113: 48 8b 45 d8 mov -0x28(%rbp),%rax

117: 48 89 c7 mov %rax,%rdi

11a: e8 00 00 00 00 callq 11f <main+0x11f>

11f: b8 00 00 00 00 mov $0x0,%eax

124: c9 leaveq

125: c3 retq

1. *Use at least one Windows API call in your program and run it in the Visual Studio environment. Submit your program and output. What is the difference between system call and API?*
   1. **hello\_windows.c**

#include <stdio.h>

#include <stdlib.h>

#include <stddef.h>

#include <windows.h>

int main ()

{

char line[256];

size\_t length;

FILE \*fp = NULL;

char message[] = "Hello to you too!\n";

// Open file "file.txt"

char file\_name[] = "file.txt";

fp = fopen(file\_name, "r");

// Check for error opening file

if(fp == NULL)

{

perror("Error opening file.\n");

exit(EXIT\_FAILURE);

}

// Check file type

if(GetFileType(fp) != FILE\_TYPE\_CHAR)

{

printf("File type not char\n");

}

// Print file contents

length = 0;

printf("File %s opened and reads as follows: \n", file\_name);

while (fgets(line, sizeof(line), fp))

{

printf("%s", line);

}

// Close file

fclose(fp);

// Open file for appending

fp = fopen(file\_name, "a");

// Write to file

printf("Writing to file: %s", message);

fprintf(fp, "%s", message);

// Close file

fclose(fp);

// Return

return 0;

}

* 1. **output**

File type not char

File file.txt opened and reads as follows:

Hello World!

Writing to file: Hello to you too!

* 1. **file.txt before run**

Hello World!

* 1. **file.txt after run**

Hello World!

Hello to you too!

**An API (in this case Windows API) is a way for the application to interface with an existing library or service that wraps around the kernel call. For a system call, the application calls the kernel to perform some service.**