Stanley George
ECE 460: Computer Operating Systems
Professor Jeff Hakner
Problem Set 8: System Calls

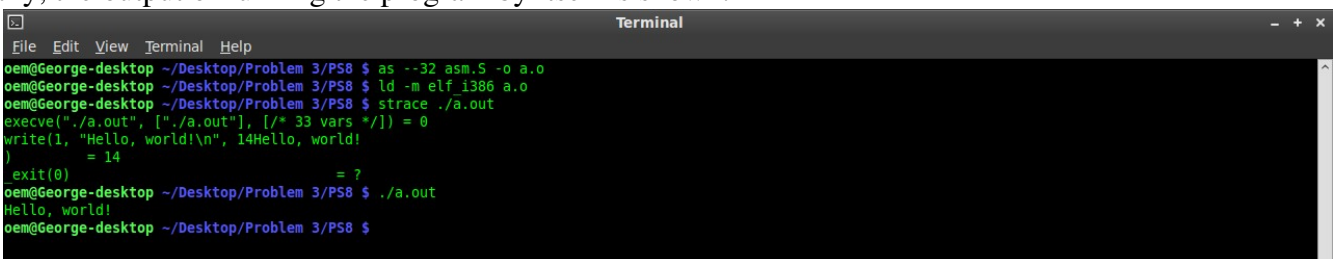
Problem 1:
The following code was the simple program that was run by `strace`:

```
#include <stdio.h>
main()
{
   printf("Hello, world!\n");
}
```

Problem 2:
The following screen shot shows the assemble/link build process. It also shows the output of running this program with `strace`. `strace` shows that three system calls were made so that this program could correctly run:

1. `execve()`
2. `write()`
3. `_exit()`

Finally, the output of running the program by itself is shown.



This was the program code:

```
.text
.globl _start

_start:
      movl  $4, %eax    #put write syscall # (4) in eax
      movl  $1, %ebx    #put fd=stdout into ebx
      movl  $msg, %ecx  #put string into ecx
      movl  $len, %edx  #put string length into edx
      int   $0x80       #call kernel

      movl  $1, %eax    #store exit syscall # (1) in eax
      movl  $0, %ebx    #put exit code into ebx
      int   $0x80       #call kernel

.data
msg:
      .ascii "Hello, world!\n"
      len = . - msg
```

Problem 3:

When the program just contains a write system call, a **`SIGSEGV`** is incurred as shown by the following screen shot

```
Terminal
File  Edit  View  Terminal  Help
oem@George-desktop ~/Desktop/Problem 3/PS8 $ as --32 asm.S -o a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ld -m elf_i386 a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ./a.out
Hello, world!
Segmentation fault
oem@George-desktop ~/Desktop/Problem 3/PS8 $ strace ./a.out
execve("./a.out", ["./a.out"], [/* 33 vars */]) = 0
write(1, "Hello, world!\n", 14Hello, world!
)         = 14
--- SIGSEGV (Segmentation fault) @ 0 (0) ---
+++ killed by SIGSEGV +++
oem@George-desktop ~/Desktop/Problem 3/PS8 $
```

Next, the `_exit` system call was made with an exit status of 7. The next screen shows the terminal output after running this program with, and then without `strace`. The value of the shell variable `$?` is also shown in this screen shot.

```
Terminal
File  Edit  View  Terminal  Help
oem@George-desktop ~/Desktop/Problem 3/PS8 $ as --32 asm.S -o a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ld -m elf_i386 a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ strace ./a.out
execve("./a.out", ["./a.out"], [/* 33 vars */]) = 0
write(1, "Hello, world!\n", 14Hello, world!
)         = 14
 exit(7)                                 = ?
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ./a.out
Hello, world!
oem@George-desktop ~/Desktop/Problem 3/PS8 $ echo $?
7
oem@George-desktop ~/Desktop/Problem 3/PS8 $
```

When the `_exit` system call was put into the code, the program did not receive any fatal signal. Also, the shell variable `$?` had a value of 7.

strace shows that the last system call made is exit_group which is similar to exit but instead of only terminating the calling thread, it also terminates all threads in the calling process's thread group. This system call performs the clean up operations after a program has returned from main. This cleanup is necessary in order for the OS to reclaim the resources it had allocated for the process. Without this exit call, the program and the kernel does not know that the process has actually finished. Thus, upon reentry into user mode, the process's instruction pointer will attempt to fetch the next instruction from the .text region. However this region is not populated with any valid instruction beyond the instruction for jumping to the interrupt vector. Therefore a SIGSEGV is incurred.

Problem 4:

Deliberate error #1: passing an invalid address for the write string

```
.text
.globl _start

_start:
      movl  $4, %eax
      movl  $1, %ebx
      movl  $msg+0xFFF, %ecx  #store invalid string address into ecx
      movl  $len, %edx
      int   $0x80

      movl  $1, %eax
      movl  $0, %ebx
      int   $0x80
```

```
.data
msg:
      .ascii "Hello, world!\n"
      len = . - msg
```

```
Terminal
File  Edit  View  Terminal  Help
oem@George-desktop ~/Desktop/Problem 3/PS8 $ as --32 asm.S -o a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ld -m elf_i386 a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ strace ./a.out
execve("./a.out", ["./a.out"], [/* 33 vars */]) = 0
write(1, 0x804a097, 14)                 = -1 EFAULT (Bad address)
 exit(0)                                = ?
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ./a.out
oem@George-desktop ~/Desktop/Problem 3/PS8 $
```

`strace` shows that `errno` is set to `EFAULT` for bad address and that `write` returns -1

Deliberate error #2: passing an invalid system call number

```
.text
.globl _start

_start:
      movl  $337, %eax  #store invalid syscall # into eax
      movl  $1, %ebx
      movl  $msg, %ecx
      movl  $len, %edx
      int   $0x80

      movl  $1, %eax
      movl  $0, %ebx
      int   $0x80

.data
msg:
      .ascii "Hello, world!\n"
      len = . - msg
```

```
Terminal
File  Edit  View  Terminal  Help
oem@George-desktop ~/Desktop/Problem 3/PS8 $ as --32 asm.S -o a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ld -m elf_i386 a.o
oem@George-desktop ~/Desktop/Problem 3/PS8 $ strace ./a.out
execve("./a.out", ["./a.out"], [/* 33 vars */]) = 0
SYS_337(0x1, 0x8049098, 0xe, 0, 0)      = -1 ENOSYS (Function not implemented)
 exit(0)                                = ?
oem@George-desktop ~/Desktop/Problem 3/PS8 $ ./a.out
oem@George-desktop ~/Desktop/Problem 3/PS8 $
```

`strace` shows that `errno` is set to `ENOSYS` because no system call exists for 337.

Problem 5:

```
Terminal
george@george-Vostro-1400 ~/Desktop/ECE460/PS8 $ gcc -c p5.c
george@george-Vostro-1400 ~/Desktop/ECE460/PS8 $ gcc -o syscall_cost p5.o
george@george-Vostro-1400 ~/Desktop/ECE460/PS8 $ ./syscall_cost A
Empty loop with 10000000 iterations took 67550629 ns to complete
The average time per loop iteration was 6.755063 ns
george@george-Vostro-1400 ~/Desktop/ECE460/PS8 $ ./syscall_cost B
time_diff = 75757329 fcn_time = 14631888 loop_time = 61125441
Empty function called in loop with 10000000 iterations took 14631888 ns to complete
The average time per empty function call was 1.463189 ns
george@george-Vostro-1400 ~/Desktop/ECE460/PS8 $ ./syscall_cost C
time_diff = 1879996122 syscall_time = 1817554802 loop_time = 62441320
System call getuid() called in loop with 10000000 iterations took 1817554802 ns to complete
The average time per system call was 181.755480 ns
george@george-Vostro-1400 ~/Desktop/ECE460/PS8 $
```

According to the above screen shot:

- a single loop iteration takes about 6.76 nanoseconds (ns)
- a single empty function call takes about 1.46 ns
- a single system call takes about 181.76 ns

***(it was verified with strace that the getuid() is not cached)*

The cost of a simple system call is about 124.5 times greater than an empty function call. One reason this is the case is because of context switching from user mode to kernel mode. This context switch features a number of safety checks by the operating system (such as checking user privillege rights and parameter checking on the parameters passed into the system call) to enforce the "world view" seen by the user level process and keep it inside a protective bubble. Also, since the system call jumps to a different portion of memory (the instructions of the system call are NOT in the same address space of the calling process), the calling process's state must be preserved (such as the stack pointer and instruction pointer). All this extra overhead incurs the higher cost seen for system calls. It is a necessary evil.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define MAX_ITER 10000000    /*do 10 million iterations*/
#define BILLION  1000000000  /*because it's hard to keep track of 0s*/

unsigned long gettimedif(struct timespec *start, struct timespec *end);
void error(const char *msg);
void my_emptyfcn();

int main(int argc, char *argv[])
{
      if(argc != 2)
      {
            fprintf(stderr, "usage: ./syscall_cost [ABC]\n");
            exit(1);
      }

      struct timespec start, end;
      unsigned long long i, time_diff, loop_time, fcn_time, syscall_time;
      double ns_per_operation;

      if(clock_gettime(CLOCK_REALTIME, &start) == -1)
            error("clock_gettime");
      for(i = 0; i < MAX_ITER; i++) { ; }
      if(clock_gettime(CLOCK_REALTIME, &end) == -1)
            error("clock_gettime");
      loop_time = gettimedif(&start, &end);

      switch(*argv[1])
      {
            case 'A':
                  ns_per_operation = (double)((long double)loop_time/MAX_ITER);
                  printf("Empty loop with %llu iterations took %llu ns to complete\n",
                        (unsigned long long) MAX_ITER, loop_time);
                  printf("The average time per loop iteration was %f ns\n",
```

```
                        ns_per_operation);
                return 0;
        case 'B':
                if(clock_gettime(CLOCK_REALTIME, &start) == -1)
                        error("clock_gettime");
                for(i = 0; i < MAX_ITER; i++) { my_emptyfcn(); }
                if(clock_gettime(CLOCK_REALTIME, &end) == -1)
                        error("clock_gettime");
                time_diff = gettimedif(&start, &end);

                fcn_time = time_diff - loop_time;
                ns_per_operation = (double)((long double)fcn_time/MAX_ITER);
                printf("time_diff = %llu fcn_time = %llu loop_time = %llu\n",
                        time_diff, fcn_time, loop_time);
                printf("Empty function called in loop with %llu ",
                        (unsigned long long) MAX_ITER);
                printf("iterations took %llu ns to complete\n", fcn_time);
                printf("The average time per empty function call was %f ns\n",
                        ns_per_operation);
                return 0;
        case 'C':
                if(clock_gettime(CLOCK_REALTIME, &start) == -1)
                        error("clock_gettime");
                for(i = 0; i < MAX_ITER; i++) { getuid(); }
                if(clock_gettime(CLOCK_REALTIME, &end) == -1)
                        error("clock_gettime");
                time_diff = gettimedif(&start, &end);

                syscall_time = time_diff - loop_time;
                ns_per_operation = (double)((long double)syscall_time/MAX_ITER);
                printf("time_diff = %llu syscall_time = %llu loop_time = %llu\n",
                        time_diff, syscall_time, loop_time);
                printf("System call getuid() called in loop with %llu ",
                        (unsigned long long) MAX_ITER);
                printf("iterations took %llu ns to complete\n", syscall_time);
                printf("The average time per system call was %f ns\n",
                        ns_per_operation);
                return 0;
        default:
                fprintf(stderr, "usage: ./syscall_cost [ABC]\n");
                exit(1);
        }
}

void my_emptyfcn() {}

unsigned long gettimedif(struct timespec *start, struct timespec *end)
{
        long start_s, start_ns, end_s, end_ns, tot_start_ns, tot_end_ns;

        start_s = start->tv_sec;
        start_ns = start->tv_nsec;
        end_s = end->tv_sec;
        end_ns = end->tv_nsec;

        tot_start_ns = (start_s * BILLION) + start_ns;
        tot_end_ns = (end_s * BILLION) + end_ns;

        return (tot_end_ns - tot_start_ns);
}
```

```
void error(const char *msg)
{
      perror(msg);
      exit(1);
}
```