

CMSC216: Practice Exam 2A SOLUTION

Spring 2024

University of Maryland

Exam period: 20 minutes Points available: 40 Weight: 0% of final grade

Problem 1 (20 pts): Nearby is a `main()` demonstrating the use of the function `setday()`. Below each call to `setday()`, its expected behavior and return are printed.

Implement this function according to the documentation given in **x86-64 assembly**. Comments below the `yearday_t` struct give information about how it lays out in memory and as a packed register argument.

```

1 .text ##### SOLUTION #####
2 .global setday
3 setday:
4     ## rdi is packed {int day; int year}
5     ## Extract fields from rdi
6     movq %rdi, %rdx
7     andl $0xFFFFFFFF, %edx # edx now day
8     movq %rdi, %rcx
9     shrq $32, %rcx         # ecx now year
10    andl $0xFFFFFFFF, %ecx # optional mask
11
12    ## do range checking
13    cmpl $0, %edx
14    jl .ERROR
15    cmpl $1970, %ecx
16    jl .ERROR
17
18    subl $1970, %ecx        # offset from 1970
19    imull $365, %ecx        # mult by days/year
20    addl %edx, %ecx        # add days
21
22    ## write back to global variable
23    movl %ecx, DAYS_SINCE_1970(%rip)
24
25    movl $0, %eax
26    ret
27
28 .ERROR:
29    ## error case, set global and ret
30    movl $-1, DAYS_SINCE_1970(%rip)
31    movl $1, %eax
32    ret

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // global: days since 1/1/1970
5 int DAYS_SINCE_1970 = 0;
6
7 // struct containing date info
8 typedef struct {
9     int day; int year;
10 } yearday_t;
11 // Layout of yearday_t in memory and
12 // as a packed register argument.
13 //
14 // |           | Byte |   Byte | Packed |
15 // | Field | Size | Offset | Bits |
16 // |-----+-----+-----+-----|
17 // | day   | 4   |   +0   | 0-31 |
18 // | year  | 4   |   +4   | 32-63|
19
20 int setday(yearday_t yd);
21 // DEFINED IN ASSEMBLY
22 //
23 // Extracts the day and year fields
24 // from the provided struct yd. If day is
25 // negative or year is < 1970, sets the
26 // global variable DAYS_SINCE_1970 to
27 // be -1 and returns 1. Otherwise
28 // computes the number of days since
29 // 1970 based on these fields assuming
30 // NO LEAP YEARS and 365 days per
31 // year. Sets the global variable
32 // DAYS_SINCE_1970 to this value and
33 // returns 0.
34
35 int main(int argc, char *argv[]){
36     // Demonstrate 3 examples of setday()
37     int ret;
38     yearday_t yd1 =
39         { .day = 20, .year = 1970 };
40     ret = setday(yd1);
41     printf("%3d days since 1970 (ret: %d)\n",
42           DAYS_SINCE_1970, ret);
43     // 20 days since 1970 (ret: 0)
44
45     yearday_t yd2 =
46         { .day = 3, .year = 1972 };
47     ret = setday(yd2);
48     printf("%3d days since 1970 (ret: %d)\n",
49           DAYS_SINCE_1970, ret);
50     // 733 days since 1970 (ret: 0)
51
52     yearday_t yd3 =
53         { .day = 7, .year = 1955 };
54     ret = setday(yd3);
55     printf("%3d days since 1970 (ret: %d)\n",
56           DAYS_SINCE_1970, ret);
57     // -1 days since 1970 (ret: 1)
58     return 0;
59 }

```

Problem 2 (10 pts): Below is a `main()` function which uses the function `setarray()`. As the demo shows, compiling with a C version of this function works fine but the assembly version has some problems.

```

// setarray_main.c          // setarray_c.c          ## setarray_asm.s
#include <stdio.h>          1 void setarray(long *arr,      1 .text
int main(){                2         long len,           2 .globl setarray
    long arr[3];           3         long val)          3 setarray:
    setarray(arr,3,10);    4 {                          4     movq    $0, %rax
    for(int i=0; i<3; i++){ 5     for(long i=0; i<len; i++){ 5 .LOOP:
        printf("%2d ",arr[i]); 6         arr[i] = val;       6     cmpq   %rsi, %rax
    }                       7     }                       7     jg     .DONE
    printf("\n");          8     return;                 8     movq   %rdx, (%rdi,%rax,8)
    return 0;              9 }                            9     addq   $1, %rax
}                           10                                10    jmp    .LOOP
                               11 .DONE:                       11 .DONE:
                               12     ret                       12     ret

>> gcc setarray_main.c setarray_c.c      >> gcc setarray_main.c setarray_asm.s
>> ./a.out                                >> ./a.out
10 10 10                                  10 10 10
*** stack smashing detected ***: terminated
Aborted (core dumped)

```

Describe why the assembly version causes Stack Smashing and how to fix it.

SOLUTION: The assembly instruction at line 7 in the assembly code is what terminates the loop. Unfortunately, this `jg` causes one extra loop iteration which goes out of bounds in the target array (the 4th element at index 3 in the `main` function). This changes data near the return address which is detected as a problem causing the program to terminate. The fix is to change `jg` to `jge` to stop going out of bounds in the array.

Problem 3 (10 pts): While debugging a binary program, Nils Punters encounters an assembly instruction that baffles him: `test %rax,%rax`. Nils is struggling to understand what this could possibly accomplish. **Explain** what the `testX` instruction does AND what it is likely being used to do in the code Nils is examining which is shown nearby.

```

+=====GDB=====+
|>>0x55154 <nodes_sorted+11> test   %rax,%rax # Nils: WTF? |
| 0x55157 <nodes_sorted+14> je     0x55555555167 <nodes_sorted+30> |
| 0x55159 <nodes_sorted+16> mov    (%rax),%edx |
| 0x5515b <nodes_sorted+18> cmp    %ecx,%edx |
| 0x5515d <nodes_sorted+20> jl     0x55555555173 <nodes_sorted+42> |
| 0x5515f <nodes_sorted+22> mov    0x8(%rax),%rax |
| 0x55163 <nodes_sorted+26> mov    %edx,%ecx |
| 0x55165 <nodes_sorted+28> jmp    0x55555555154 <nodes_sorted+11> |
| 0x55167 <nodes_sorted+30> mov    $0x1,%eax |
| 0x5516c <nodes_sorted+35> ret                               |
+=====+

```

SOLUTION: The `testX` instruction is equivalent to a bitwise-And but the result is discarded. It is run solely to set the `FLAGS` register. Testing a register against itself yields information such as whether it is Negative (signed) or Zero and will set the flags register accordingly. A 64-bit `test` like the one Nils is looking at on `<node_sorted+11>` could be used to check a 64-bit number for being Zero or equivalently checking to see if a Pointer is NULL (encoded as 0 in binary). The use of “nodes” in the name of the function and the fact that that the `%rax` register is used to access main memory implies that it is a pointer and the instruction is checking whether it is NULL.