EE209 Fall 2022 Final

SOLUTION

Section: ________

Student id: ________

Name: ________

1: ____/10

2: ____/12

3: ____/12

4: ____/16

5: ____/14

6: ____/22

7: ____/14

Total: ____/100

Please write CLEARLY

Unreadable solutions are assumed incorrect

SOLUTION

1. Execeptions (10 points)

1.1 Please fill each blank with the appropriate exception classes (4 points)

Class	Cause	Async/Sync	Return Behavior
(1)	Non-recoverable error	Sync	Do not return
(2)	(Maybe) recoverable error	Sync	(Maybe) return to current instr
(3)	Intentional	Sync	Return to next instr
(4)	Signal from I/O device	Async	Return to next instr

(1) Abort

(2) Fault

(3) Trap

(4) Interrupt

1.2. Please write assembly code to call exit(1234) with a trap in 32bit x86. Never use external system-level function (e.g., exit). Note that the system call number for exit is 1. (6 points)

```
movl $1, %eax
movl $1234, %ebx
int     $128
```

2. Memory management (12 points)

Answer the following questions assuming that the current page table is given as shown below. Assume the size of page is 4KB. If you need any new page, assume that the entry #4 in the page table will be swapped out to the disk address **zz**.

#	V	Physical or disk address
0	0	xx
1	1	2
2	0	yy
3	0	null
4	1	1

2.1 movl 0x00001104, %eax (4 points)

(1) Will page fault occur? No

(2) Will your program crash? If not, fill the page table after this instruction.

#	V	Physical or disk address
0	0	xx
1	1	2
2	0	yy
3	0	null
4	1	1

2.2 movl 0x00002104, %eax (4 points)

(1) Will page fault occur? Yes

(2) Will your program crash? If not, fill the page table after this instruction.

#	V	Physical or disk address
0	0	xx
1	1	2
2	1	1
3	0	null
4	0	zz

2.3 movl 0x00003104, %eax (4 points)

(1) Will page fault occur? Yes

(2) Will your program crash? If not, fill the page table after this instruction.

It will crash

#	V	Physical or disk address

3. Process Management (12 points)

3.1. Here are codes for exe_a and exe_b. What happens if we execute `./exe_a` if alphabet.txt contains "abcdefghijklmnopqrstuvwxyz". Assume that unexpected errors will not happen (e.g., read, fork, execvp will be all successful) (6 points)

```
// exe_a.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
  int fd = open("./alphabet.txt", O_RDONLY);
  int status = 0;
  char* argv[] = {NULL};

  char buf[0x100] = {0};
  read(fd, buf, 3);

  printf("1: %s₩n", buf);

  if (!fork()) {
    execvp("./exe_b", argv);
  }

  wait(&status);
}
```

```c
// exe_b.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char** argv) {
  char buf[0x100] = {0};
  int fd = open("./alphabet.txt", O_RDONLY);

  read(3, buf, 3);
  printf("2: %s₩n", buf);

  read(fd, buf, 3);
  printf("3: %s₩n", buf);
}
```

1: _______abc__________

2: _______def__________

3: _______abc___________

3.2. How many 'A's would be printed out in this program? (4 points)

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

int main(int argc, char** argv) {
  for (int i = 0; i < 3; i++) {
    if (fork()) {
      fprintf(stderr, "A");
    }
    else {
      fprintf(stderr, "AA");
    }
  }
}
```

________21________

3.3. What is the output of this program if we run this program with `./test 1 2`? (2 points)

```
// test.c
#include <stdio.h>

int main(int argc, char** argv) {
  printf("argc: %d₩n", argc);
}
```

argc:__3____

4. Signals (16 points)

4.1 Please make a program that ticks every two seconds using alarm (4 points)

```
// alarm.c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <signal.h>
#include <unistd.h>

static void myHandler(int iSig)
{
   printf("Tick by alarm...₩n");
   // (1)
}



int main() {
  signal(SIGALRM, myHandler);
  // (2)

  for(;;)
    ;
}
```

(1) alarm(2);

(2) alarm(2);

4.2 Please make a program that ticks every two seconds using setitimer. Your answer can be multiple statements. (6 points)

```
// signal.c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <signal.h>
#include <unistd.h>
#include <sys/time.h>

static void myHandler(int iSig)
{
    printf("Tick by interval timer...₩n");
}

int main() {
   struct itimerval sTimer;
   signal(SIGPROF, myHandler);

   /* Send the first signal in 2 second, 0 microseconds. */

   (1)


  /* Send subsequent signals in 2 second, 0 microseconds intervals. */

   (2)

   setitimer(ITIMER_PROF, &sTimer, NULL);

   for(;;)
     ;
}
```

(1)

```
   sTimer.it_value.tv_sec = 2;
   sTimer.it_value.tv_usec = 0;
```

(2)

```
   sTimer.it_interval.tv_sec = 2;
   sTimer.it_interval.tv_usec = 0;
```

4.3. When we execute both './alarm' and './signal', we found that one program ticks slower than the other. (6 points)

(1) Which one is slower?

'signal'

(2) Why?

It uses cpu time, which does not include other processes' time or OS's time unlike alarm.

5. Assembly 1 (14 points)

5.1 Below are the initial memory and register values. After the sequence of instructions below, what are the values stored in memory addresses and registers in the table? (4 pts)

Memory Address	Value
0x278	0x77
0x270	0x116
0x268	0x0
0x260	0x828
0x258	0x51
0x250	0x7
0x248	0x27
0x240	0x5
0x238	0x65
0x230	0x6
0x228	0x10
0x220	0x81
0x218	0x124
0x210	0x4
0x208	0x45
0x200	0x109

Register	Value
%rdi	0x10
%rsi	0x20
%rcx	0x208
%rdx	0x200

```
leaq (%rdx, %rsi, 2), %rcx
movq %rcx, 0x8(%rcx)
leaq 0x10(%rcx, %rdi), %rdx
movq $0x25, 0x8(%rcx, %rsi)
```

Fill in the table with only memory addresses and registers with changed values. Answer in hex.

Memory address or Register	Value
0x268	0x25
0x248	0x240
%rcx	0x240
%rdx	0x260

5.2 Assume %rax==a, %rbx==b, %rcx==c.

(1) Using at most three calls to leaq (and no other instructions), store (48*a) into %r8. You may use other registers for temporary storage If needed. (2 points)

```
leaq (%rax, %rax, 2), %rax
leaq (, %rax, 4), %rax
leaq (, %rax, 4), %r8
```

(2) Using at most three calls to leaq (and no other instructions), store (3*b+9*c+13) into %r8. You may use other registers for temporary storage If needed. (2 points)

```
leaq (%rbx, %rbx, 8), %rbx
leaq (%rcx, %rcx, 2), %rcx
leaq 0xd(%rbx, rcx), %r8
```

5.3 Consider the C code below (left) where M and N are constants declared with #define. Right is the corresponding X86-64 (long is 8 bytes). What is the value of M and N? (6 points)

M= 9   N= 6

```c
long mat1[M][N];
long mat2[N][M];

long copy_element(int i, int j)
{
    mat1[i][j] = mat2[j][i];
}
```

```
copy_element:
        leaq    (%rsi,%rsi,8),%rax
        addq    %rdi,%rax
        movq    mat2(,%rax,8),%rcx
        leaq    (%rdi,%rdi,2),%rdx
        leaq    (%rdx,%rdx,1),%rax
        addq    %rsi,%rax
        movq    %rcx,mat1(,%rax,8)
        retq
```

6. Assembly 2 (22 points)

6.1 After the following is run, indicate the condition flags that are set. If no flags are set, write none. Set means equal to 1. (2 points each)

(1)

```
movq    $0, %rax
movq    $7, %rbx
cmpq    %rbx, %rax
```

SF, CF

(2)

```
movq  $0xffffffffffffffff, %rbx
movq  $0x8000000000000001, %rcx
addq  %rbx, %rcx
leaq  1(%rbx), %rcx
```

SF, CF

6.2 Consider the silly read function in C and X86-64.

```c
void read()
{
    char buf[3];
    gets(buf);
};
```

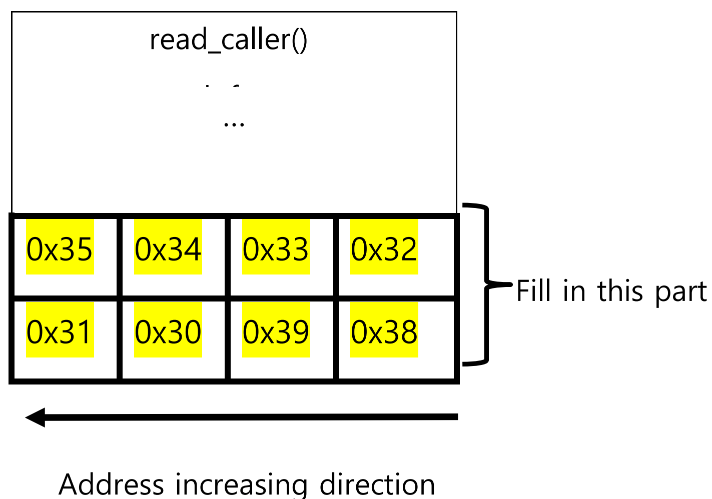
```
read:
        subq    $0x8, %rsp
        movq    %rsp, %rdi
        callq   gets
        addq    $0x8, %rsp
        retq
```

(1) Would the program crash if the input is: abcdefgh<enter>? If the answer is yes, also explain why. (2 points)

The program crashes. Last newline input "<enter>" corrupts the return address.

(2) Consider the input: 01234567890123456789<enter>. Assuming read() was called by read_caller(), fill in the last 8 bytes of read_caller()'s stack frame. Each box should hold one byte represented in hex. (4 points)

(NOTE: the ASCII code for the character '0' is 0x30)



6.3 Using the assembly code on the left, complete the C code on the right (4points each)

(1)

```
f1:
    cmpq    %rsi, %rdi
    jle     .L2
    addq    %rsi, %rdi
    jmp     .L3
.L2:
    addq    %rdi, %rsi
.L3:
    leaq    (%rsi,%rdi), %rax
    retq
```

```
int f1 (int x, int y) {

if (x > y)
  x = x + y;
else
  y = x + y;

return x + y;
```

(2)

```
f2:
    cmpq    %rsi, %rdi
    jle     .L6
.L9:
    addq    $1, %rsi
    cmpq    %rsi, %rdi
    jg      .L9
.L6:
    leaq    (%rsi,%rdi), %rax
    retq
```

```
int f2 (int x, int y) {

while (x > y)
  y ++;

return x + y;
```

6.4 Quickies. Assume X86-64.

(1) If we changed the function-calling convention to treat every single register as CALLEE-SAVE, and recompiled all our code in that fashion, the code would still run the same. True or false? (2 points)

False, but both can be correct

(2) MOV can move bits from memory to memory in a single call. True or false? (2 points)

False

7. Assembler and Stack (14 points)

7.1 Consider the following assembly language program.

```
        .section ".rodata"
msg1:
        .asciz  "Enter Character₩n"
msg2:
        .asciz  "This is not X₩n"
        .section ".text"
        .globl  main
main:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   $msg1
        call    printf
        addl    $4, %esp
        call    getchar
        cmpl    $'X', %eax
        je      finish
        pushl   $msg2
        call    printf
        addl    $4, %esp
finish:
        movl    $0, %eax
        movl    %ebp, %esp
        popl    %ebp
        ret
```

(1) Assembler performs 2 passes over the assembly language program. Briefly describe two passes. A single sentence is sufficient for each pass. (2 points)

1st pass: Assembler traverse the assembly code to generate symbol table.
2st pass: Assembler generate(fill-in) sections, and if the symbol cannot be resolved, fill the relocation records and add to the symbol table.

(2) Fill in the symbol table and the relocation record after the Assembler performs 2 passes. Assume the length of every instruction is 4-byte. Write '**?**' if the Assembler cannot resolve. (8 points)

Symbol Table				
Label	**Section**	**Offset**	**Local?**	**Seq#**
msg1	RODATA	0	local	0
msg2	RODATA	16	local	1
main	TEXT	0	global	2
finish	TEXT	44	local	3
printf	?	?	global	4
getchar	?	?	global	5

Relocation Records		
Section	**Offset**	**Seq#**
TEXT	**9**	0
TEXT	**13**	4
TEXT	**21**	5
TEXT	**33**	1
TEXT	**37**	4

7.2 Quickies.

(1) Calling a function without any local variable does not grow the stack size. True or false? (2 points)

False

(2) There is no .stack section in an ELF file. Why is this so? (2 points)

Necessary only at runtime