Introduction to Computer Systems
2018 Spring Middle Examination

Name_____________ Student No._______________ Score__________

Problem 1: (11 points)

1.

2.

3.

Problem 2: (10 points)

1. [1] [2]

2.

3.

Problem 3: (28 points)

1 [1] [2] [3]

2.

3. [1] [2] [3] [4]

[5] [6] [7] [8]

[9] [10] [11] [12]

4

5

6

7

Problem 4: (23 points)

1 [1] [2] [3] [4]

[5] [6] [7] [8]

[9] [10] [11] [12]

[13] [14] [15] [16]

[17] [18] [19] [20]

2.

3. [1] [2] [3] [4]

[5] [6] [7] [8]

4. [1] [2] [3]

[4] [5]

Problem 5: (7 points)

1 [1] [2] [3]

2

3

Problem 6: (21 points)

1. [1] [2] [3] [4]

   [5] [6] [7] [8]

   [9] [10]

2.

3. Sent: Received:

4

Problem 1: Process (11 points)

```
#include <stdio.h>
#include "csapp.h"

int main() {
    char c = 'A';
    if (Fork() == 0) {
        c++;
        if (Fork() == 0) {
            c++;
        } else {
            while (waitpid(-1, NULL, WNOHANG) > 0);
            char str[2] = {c, 0};
            char *argv[] = {"/bin/echo", str, 0};
            Execve(argv[0], &argv[0], 0);
            c += 2;
        }
    } else {
        while (wait(NULL) > 0);
    }
    printf("%c\n", c);
}
```

Note: **/bin/echo** is an executable file that will print its arguments on the screen. No error occurs in the execution.

1. Are there any **zombie** child processes in the given program? Please explain your answer. (4’)
2. Please write down **ALL** possible outputs of the give program. (3’)
3. The previous program uses **printf** function to print value on the screen and accesses variable **c** in many processes without any protection. Will they cause any concurrent problems? Why? (4’)

Problem 2: IO (10 points)

```
1. #include "csapp.h"
2.
3. int main() {
4.     int fd1, fd2;
5.     char c[4] = "ics";
6.     fd1 = open("a.txt", O_RDWR, 0);
7.     fd2 = open("b.txt", O_RDWR, 0);
8.     read(fd1, &c, 2);
9.     if (Fork() == 0) {
10.        //dup2(fd2, 1);
11.        read(fd2, &c, 1); printf("%s", c); fflush(stdout);
12.        read(fd1, &c, 3); printf("%s", c); fflush(stdout);
13.        read(fd2, &c, 2); printf("%s", c); fflush(stdout);
14.    } else {
15.        wait(NULL);
16.        dup2(fd1, fd2);
17.        fd1 = open("c.txt", O_RDWR | O_CREAT | O_TRUNC, 0);
18.        read(fd2, &c, 2);
19.        write(fd1, &c, 3);
20.    }
21.    close(fd1);
22.    close(fd2);
23. }
```

Note: Initially, **a.txt** contains "**SJTU2018**"; **b.txt** contains **"IDoLoveICS"**; **c.txt** does not exist. No error occurs in the execution.

1. Please write down the output on the **screen** and the content of **c.txt** when the program runs normally. (4')

   screen: ___[1]___ **c.txt**: ___[2]___

2. After executed this program, we use command "**cat c.txt**" to check the text in **c.txt**. However, we got the following error message on the screen

   cat: c.txt: Permission denied

   Please explain why this happened and modify previous code to make **cat** works. (3') (**NOTE**: **cat** is a command line program that read a file and print its data to standard output. Any reasonable modification will be considered as correct.)

3. Please write down the content of **b.txt** after the program runs normally when line 10 "**dup2(fd2, 1)**" is uncommented (3').

Problem 3: Cache (28 points)

Jack has a **64-bit machine** with a **2-way set** associative cache. There are **4 sets**. Each block is **8 bytes**. The following table shows the content of the data cache at time T. **ByteX** is the byte value stored at offset **X**. Assume the cache uses **LRU** and **write back** policy.

Set	Tag	Valid	Dirty	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0	--	0	--	--	--	--	--	--	--	--	--
	0x401b5	1	0	0x46	0xaf	0xfa	0x36	0x92	0x45	0x01	0x37
1	0x40136	1	1	0x46	0x4a	0x11	0x08	0x33	0x47	0x04	0xda
	--	0	--	--	--	--	--	--	--	--	--
2	0x401b5	0	0	0x42	0x08	0x07	0x66	0x62	0x6a	0x6f	0x9f
	0x8036b	1	0	0x23	0x2a	0x73	0x75	0xe7	0xef	0x2e	0x88
3	0x2009c	1	1	0x05	0x81	0xa2	0xa3	0x83	0x30	0x40	0x32
	0x40139	1	1	0xdd	0xda	0xcd	0xc5	0x6c	0xaf	0x9d	0x90

1. How would a 64-bit physical machine memory address be split into tag/set-index/block-offset fields in this machine? (2’*3=6’)

   tag _[1]_ bits **set-index** _[2]_ bits **block-offset** _[3]_ bits

2. Why caches use the **middle bits** for the set index instead of the **high-order bits**? (2’)

3. A short program will access memory in the following sequences starting from time T. Assume there are **no other memory accesses** from other programs or kernel. Each memory access will **read or write 1 byte**. Please fill in the following blanks. If the value does not exist or cannot be determined, fill the blank with ‘--’. (1’*12=12’)

Order	Operation	Hit/Miss	Byte Returned
1	Read 0x401368	[1]	[2]
2	Read 0x8036a5	[3]	[4]
3	Write 0x8036b3	[5]	[6]
4	Write 0x4014aa	[7]	[8]
5	Write 0x40139b	[9]	[10]
6	Read 0x8026cf	[11]	[12]

4. Assume the program will **flush all data back to memory** in this cache after the memory accesses in question 3. How many cache lines will be written to memory **totally**? (Note: **Calculate from time T.**) (2′)

Jack writes a program and test it on this cache. The size of **`int`** value is **4 bytes**. The cache is **empty** before each execution. Please only consider **data cache** access. Assume other processes do not modify the memory and cache during this execution.

```
int a[8][2];
int b[8][2];

int fun(void) {
    int i, j, sum = 0;

    for (j = 0; j < 2; j++)
        for (i = 0; i < 8; i++)
            sum += a[i][j] * b[i][j];

    return sum;
}
```

Assume the address of **`a[0][0]`** is **`0x00`**. The address of **`b[0][0]`** is **`0x40`**. Answer the following questions:

5. Please calculate the cache miss rate. (2′)
6. How to optimize the cache miss rate if we only **reorder** some part of the code? (2′)
7. Now Jack can enlarge the cache size through **one of** the following ways:

   A. Double the number of sets.

   B. Double the number of cache lines in each set.

   C. Double the block size of each cache line.

   Please choose all options above that can optimize the miss rate of the **original program**. (2′)

Problem 4: Relocation (23 points)

The following program consists of two source files: **`main.c`** and **`draw.c`**, the relocatable object files are also listed. (All the process of linking runs on an x86_64 little-endian machine)

```c
/*main.c*/
int y = 5;
static int x = 200;
int a[4];
int *ap = &a[1];
const int num = 8;
extern int draw(int n);

void main(){
    int i = draw(x);
    printf("Get %s
        using x = %d\n",
        (char*)a[i], x);
}
```

```
/*main.o*/
.text:
0000000000000000 <main>:
 0: 55                   push   %rbp
 1: 48 89 e5             mov    %rsp,%rbp
 4: 48 83 ec 10          sub    $0x10,%rsp
 8: 8b 05 00 00 00 00    mov    0x0(%rip),%eax
 e: 89 c7                mov    %eax,%edi
10: e8 00 00 00 00       callq  15 <main+0x15>
15: 89 45 fc             mov    %eax,-0x4(%rbp)
18: 8b 15 00 00 00 00    mov    0x0(%rip),%edx
1e: 8b 45 fc             mov    -0x4(%rbp),%eax
21: 48 98                cltq
                         //sign extend eax to rax
23: 8b 04 85 00 00 00 00  mov 0x0(,%rax,4),%eax
2a: 89 c6                mov    %eax,%esi
2c: bf 00 00 00 00       mov    $0x0,%edi
31: b8 00 00 00 00       mov    $0x0,%eax
36: e8 00 00 00 00       callq  3b //printf
...
.data:
...
0000000000000008 <ap>:
 8: 00 00 00 00 00 00 00 00
```

```c
/*draw.c*/
char *a[]=
 {"BaiQi","XuMo",
 "LiZeyan","ZhouQiluo"};
long y;
static long x = 20;
extern int num;

int draw(int n){
    static long x = 0;
    x = 234;
    const int num = 4;
    y = x - n;
    return y % num;
}
```

```
/*draw.o*/
.text:
0000000000000000 <draw>:
 0: 55                   push %rbp
 1: 48 89 e5             mov %rsp,%rbp
 4: 89 7d ec             mov %edi,-0x14(%rbp)
 7: 48 c7 05 00 00 00 00 movq $0xea,0x0(%rip)
 d: 00
 e: ea 00 00 00
12: c7 45 fc 04 00 00 00 movl $0x4,-0x4(%rbp)
18: 00
19: 48 8b 15 00 00 00 00 mov 0x0(%rip),%rdx
1f: 00
20: 8b 45 ec             mov -0x14(%rbp),%eax
23: 48 98                cltq
```

	25: 48 29 c2 sub %rax,%rdx 28: 48 89 d0 mov %rdx,%rax 2b: 48 89 05 00 00 00 mov %rax,0x0(%rip) 31: 00 ... //calculate y % num and return the value

1. For symbols that are defined and referenced in **main.o** and **draw.o**, please complete the symbol tables. The format of them are same as ones in **section 7.5** of CSAPP. (0.5'*20=10')

Module	Name	Value(Hex)	Size	Type	Bind	Ndx
main.o	main	00000000	61	[1]	[2]	.text
	num	[3]	[4]	[5]	[6]	[7]
	x	00000004	[8]	[9]	[10]	[11]
	draw	00000000	[12]	[13]	GLOBAL	[14]
draw.o	a	00000000	[15]	[16]	[17]	[18]
	y	00000008	[19]	OBJECT	GLOBAL	[20]

2. Please write down the output of **main.c**. (4')
   NOTE: You don't need to concern on the .o files for this problem.

3. Fill in the relocation entries of **main.o** and **draw.o**.(0.5'*8 = 4')

Relocation entries of **main.o**

Section	Offset	Type	Symbol Name
.data	00000008	R_X86_64_64	[1]
.text	00000011	[2]	[3]
.text	00000026	[4]	[5]

Relocation entries of **draw.o**

Section	Offset	Type	Symbol Name
.text	0000000a	R_X86_64_PC32	[6]
.text	0000002e	[7]	[8]

4. After relocation and the program is built, some changes will happen to the underlined instructions/data. Part of the symbol table and some comparison of relocations are given below. Fill in the blanks. (5')

Name	Section	Type	Value
num	.rodata	OBJECT	00400624
x	.bss	OBJECT	00600a20
a	.data	OBJECT	006009e0
y	.data	OBJECT	00600a08
draw	.text	FUNC	00400506
main	.text	FUNC	0040054f

Comparison of relocations of **main.o**

Section	Before relocation	After Relocation
.text	8: 8b 05 00 00 00 00	[1]
.text	10: e8 00 00 00 00	[2]
.data	8: 00 00 00 00 00 00 00 00	[3]

Comparison of relocations of draw.o

Section	Before relocation	After Relocation
.text	19: 48 8b 15 00 00 00 00	[4]
.text	2b: 48 8b 05 00 00 00 00	[5]

Problem 5: PIC (7 points)

ICSTA wrote a program which has some calls to **printf** and **atof** from shared library. Given partial .PLT **before** the execution of main:

```
<PLT[0]>:
  __[1]_: pushq  0x6009a0
  ------: jmpq   *0x6009a8
  ------: nopl   0x0(%rax)
0000000000400450<atof@plt>:
  400450: jmpq   *0x6009b0
  400456: pushq  $0x0
  40045b: jmpq   0x400440
0000000000400460 <printf@plt>:
  400460: jmpq   ____[2]____
  400466: pushq  $0x1
  40046b: jmpq   0x400440
```

and partial .GOT **before** the execution of main:

Address	Content
0x600998	0x6007c0
0x6009a0	0x7ffff7ffe1b0
0x6009a8	0x7ffff7df04e0
0x6009b0	[3]
0x6009b8	0x400466

1. Fill the blanks in .PLT and .GOT (1'*3=3')
2. What's the address of the dynamic linker? (2')
3. Please explain where and how will the table change **after** the **first** call to **atof** being executed. (2')

Problem 6: Signal (21 points)

```
volatile sig_atomic_t pre_alarm = 0, post_alarm = 0;
void handler_alrm(int sig) {
    if (post_alarm != 0) return;
    if (pre_alarm != 0) {
        sio_puts("forth alarm: "); sio_putl(alarm(0));
        sio_puts("\n");
    } else {
        pre_alarm = 1;
        sio_puts("third alarm: "); sio_putl(alarm(1));
        sio_puts("\n");
        Kill(getppid(), SIGUSR1);
        sleep(5);
        Kill(getpid(), SIGUSR2);
        post_alarm = 1;
        sio_puts("alarm done\n");
}}
volatile sig_atomic_t child_exit, pid;
void handler_usr1(int sig) { sio_puts("usr1\n"); }
void handler_usr2(int sig) { sio_puts("usr2\n"); }
void handler_chld(int sig) { sio_puts("chld\n"); child_exit = 1; }
int main() {
    sigset_t mask_all, prev_one;
    Sigfillset(&mask_all);
    Sigprocmask(SIG_BLOCK, &mask_all, &prev_one);
    Signal(SIGUSR1, handler_usr1);
    Signal(SIGUSR2, handler_usr2);
    Signal(SIGALRM, handler_alrm);
    Signal(SIGCHLD, handler_chld);
    if ((pid = Fork()) == 0) {
        printf("first alarm: %u\n", alarm(2));
        printf("second alarm: %u\n", alarm(3));
        Sigprocmask(SIG_SETMASK, &prev_one, NULL);
        Pause();
        exit(0);
    }
    printf("I'm parent\n");
    child_exit = 0;
    Kill(pid, SIGALRM);
    Sigprocmask(SIG_SETMASK, &prev_one, NULL);
    while (!child_exit)
        Pause();
}
```

Note: the kernel check the pending bits when it switches from kernel to user (e.g. returning from system call). The return of signal handler will trigger an **`sigreturn`** system call.

1. Please fill the following table: (0.5'*6+1'*4)

   For visibility, please write **T** if the line will always be printed, **F** if the line will never be printed, **M** if the line might be printed.

Line	Visibility	Possible return values of **`alarm()`**
5&6	[1]	[7]
9&10	[2]	[8]
15	[3]	------
18	[4]	------
19	[5]	------
20	[6]	------
30	**T**	[9]
31	**T**	[10]
36	**T**	------

2. Please draw the dependency diagram of output: (4')
   Note: You could use line number to represent one output line in the diagram. And you do **NOT** need to show the values in the output.

3. Please show the maximum number of SIGALRM **sent** to child and **received** by child, and give one corresponding execution flow for each. (2' * 2)

4. There are several bugs in the program, which could make the program stuck. Please figure out all the bugs, in which execution order they will happen and how to fix them with sigsuspend. (6')