

Introduction to Computer Systems

2019 Spring Middle Examination

Name_____ Student No._____ Score_____

Problem 1:

1.

2.

3.

4.

Problem 2:

1.

2.

3.

4.

Problem 3:

1 [1] [2] [3]

2. [1] [2]

 [3] [4]

 [5] [6]

3.

4.

5.

Problem 4:

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

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

 [9] [10]

2. [1] [2] [3]

 [4] [5] [6]

 [7] [8] [9]

 [10]

3. [1] [2]

 [3] [4]

4.

Problem 5:

1. [1]

[2]

2.

3. [1]

[2]

Problem 6:

1.

2.

3.

4.

5.

Problem 1: Process (11 points)

```
1.  #include <stdio.h>
2.  #include "csapp.h"
3.
4.  char * arr;
5.  int main(void) {
6.      arr = malloc(5);
7.      for (int i = 0; i < 4; i++) {
8.          arr[i] = 'A';
9.      }
10.     arr[4] = '\0';
11.
12.     if (Fork() == 0) {
13.         for (int i = 0; i < 4; i++) {
14.             if (Fork() == 0) {
15.                 arr[i] = 'B';
16.                 printf("%d, %s\n", i, arr);
17.                 exit(0);
18.             }
19.         }
20.
21.         waitpid(-1, NULL, 0);
22.         char str[10];
23.         sprintf(str, "[%d] arr=%s\n", !getpid(), arr);
24.         char *argv[] = _____;
25.         Execve("/bin/echo", &argv[0], 0);
26.     }
27.
28.     while ( waitpid(-1, NULL, WNOHANG) > 0 );
29.     printf("[%d] arr=%s\n", !getpid(), arr);
30.     free(arr);
31. }
```

Note : `/bin/echo` is an executable file that will print its arguments on the screen.

1. Fill in the blank in **line 24** to allow **line 25** to print `str` (see **line 22**) on screen. (2')
2. Does the `free` operation in **line 30** causes problem for references in **line 15** and **16**, and/or **line 23**? Please explain your answer. (3').
3. How many **zombie** child processes remain in the end? Please explain your answer. (4')
4. How many **possible outputs** can this program produce? (2')

Problem 2: IO (13 points)

```
1 #include "csapp.h"
2
3 int main(){
4     int fd_foo, fd_bar1, fd_bar2;
5     char c[3]= "12";
6     fd_foo = Open("foo.txt", O_RDWR,0);
7     Write(fd_foo,c,1);
8     Read(fd_foo,c,2);
9     Write(1, c, 1);
10
11     if (fork() == 0) {
12         fd_bar1 = Open("bar.txt", O_RDWR,0);
13         Read (fd_bar1, c, 2);
14         Write (fd_bar1, c, 2);
15         Dup2(1,fd_foo);
16         Write(fd_foo, c, 1);
17     }
18     fd_bar2 = Open("bar.txt", O_RDWR|O_APPEND,0);
19     Write (fd_bar2, c, 2);
20     Wait(NULL);
21     return 0;
22 }
```

NOTE: Initially, `foo.txt` contains "ICS2019"; `bar.txt` contains "123"; No error occurs in the execution. NOTE: suppose that read and write operations are atomic.

1. Please write down the output on **screen**. (2')
2. **Before** the child process exit, please draw a picture to describe the status of open files in the child process with **descriptor table, file table** and **v-node table**, like **Figure 10.12** in your text book. (NOTE: you don't need to consider **fd 0,1,2**) (4')
3. Please write down all possible **content** of `bar.txt`. (3')
4. If we change **line 9** from `Write(1, c, 1)` to `printf("%c", c[0])`, write down the output on screen. (2') (**Hint**: `printf` has a buffer) If we want the same output as before, how to modify the code? (NOTE: you can't modify `printf`) (2')

Problem 3: Cache (20 points)

Jack has a **32-bit machine** with a **direct-mapped** cache. There are **8 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	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0	--	0	--	--	--	--	--	--	--	--
1	0x4f352	1	0x49	0x20	0x6c	0x6f	0x76	0x65	0x20	0x54
2	--	0	--	--	--	--	--	--	--	--
3	--	0	--	--	--	--	--	--	--	--
4	0x36112	0	0x61	0x79	0x6c	0x6f	0x20	0x6a	0x53	0x77
5	0x5e08c	1	0x69	0x66	0x73	0x74	0x21	0x56	0xf8	0x83
6	--	0	--	--	--	--	--	--	--	--
7	0x3b156	1	0xaa	0xbb	0xbe	0xbd	0xbe	0x78	0x17	0xa9

- How would a 32-bit physical machine memory address be split into tag/set-index/block-offset fields in this machine? ($2^3=8$)

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

- A short program will read 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 1 byte**. Please fill in the following blanks. ($1^6=6$).

Order	Address	Hit/Miss
1	0x411488	[1]
2	0x411489	[2]
3	0x411490	[3]
4	0xd844a0	[4]
5	0x178232e	[5]
6	0x13cd48d	[6]

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.

```
1.  struct mystruct {
2.      int x;
3.      int y;
4.  }
5.
6.  typedef struct mystruct array[8][8];
7.
8.  int trans(array dst, array src) {
9.      int i, j;
10.     for (i = 0; i < 8; i++) {
11.         for (j = 0; j < 8; j++) {
12.             dst[j][i].x = src[i][j].x;
13.             dst[j][i].y = src[i][j].y;
14.         }
15.     }
16. }
```

Assume the address of `src[0][0]` is `0x00`. The address of `dst[0][0]` is `0x200`. Answer the following questions:

3. Please calculate the cache **miss rate** for `trans`. (2')
4. Assume the cache hit time is 4 clock cycles, miss penalty is 200 cycles. Please calculate the **average access time** for `trans`. (2')
5. With the same cache capacity and data block size, what is the miss rate of `trans` if it is **2-way** associate, or **4-way** associate? (4')

Problem 4: Relocation (23 points)

The following program consists of three source files: `main.c`, `name.c` and `show.c`, the relocatable object files are also listed. (All the process of linking runs on an x86-64 little-endian machine) **NOTE:** On a x86-64 machine, `sizeof(short)=2`; `sizeof(int)=4`; `sizeof(char*)=8`;

```
/*main.c*/
```

```
1. extern char* shows[];
2. extern int get_name(void);
3. short a = 1; short b = 3; int names[4];
4. int get_show(void);
5. void main(void){
6.     printf("%s perform %s\n", (char*)names[get_name()],
              (char*)shows[get_show()]);
7. }
```

```
.text:
```

```
0000000000000000 <main>:
```

```
...
```

```
9: e8 00 00 00 00      callq e <main+0xe>
e: 48 98               cltq
10: 48 8b 1c c5 00 00 00  mov    0x0(,%rax,8),%rbx
18: e8 00 00 00 00      callq 1d <main+0x1d>
1d: 48 98               cltq
1f: 8b 04 85 00 00 00 00  mov    0x0(,%rax,4),%eax
26: 48 98               cltq
28: 48 89 da            mov    %rbx,%rdx
2b: 48 89 c6            mov    %rax,%rsi
2e: bf 00 00 00 00      mov    $0x0,%edi
33: b8 00 00 00 00      mov    $0x0,%eax
38: e8 00 00 00 00      callq 3d <main+0x3d>
...
```

```
/*name.c*/
```

```
1. extern short b;
2. char *names[] = {"MAN", "LI", "KUN", "LIU"};
3. int a;
4. int get_name(void) {
5.     a = 4;
6.     return a - b;
7. }
```

```
.text:
```

```
0: 55                  push   %rbp
1: 48 89 e5            mov    %rsp,%rbp
4: c7 05 00 00 00 00 04 00 00 00  movl   $0x4,0x0(%rip)
```

e: 8b 15 00 00 00 00	mov 0x0(%rip),%edx
14: 0f b7 05 00 00 00 00	movzwl 0x0(%rip),%eax
1b: 98	cwtl
1c: 29 c2	sub %eax,%edx
1e: 89 d0	mov %edx,%eax
20: 5d	pop %rbp
21: c3	retq

/*show.c*/

```

1. char *shows[] = {"DWKM", "KLL", "JNTM", "LKH"};
2. char **gc_shows = &shows;
3. static short b = 2;
4. int get_show() {
5.     short a = 3;
6.     static short b = 1;
7.     return a - b;
8. }

```

```

.text:
0000000000000000 <get_show>:
0: 55                push  %rbp
1: 48 89 e5          mov   %rsp,%rbp
4: 66 c7 45 fe 03 00 movw  $0x3,-0x2(%rbp)
a: 0f bf 55 fe       movswl -0x2(%rbp),%edx
e: 0f b7 05 00 00 00 movzwl 0x0(%rip),%eax
15: 98               cwtl
16: 29 c2            sub   %eax,%edx
18: 89 d0            mov   %edx,%eax
1a: 5d              pop   %rbp
1b: c3              retq

.data:
...
0000000000000020 <gc_show>:
20: 00 00 00 00 00 00 00 00

```

1. For symbols that are defined and referenced in `main.o`, please complete the symbol tables. The format of them are same as ones in section 7.5 of our book. (5')

Module	Name	Value (Hex)	Size	Type	Bind	Ndx
main.o	a	00000000	2	OBJECT	GLOBAL	.data
	names	00000010	[1]	[2]	[3]	[4]
	shows	00000000	[5]	[6]	GLOBAL	[7]
	get_name	00000000	[8]	[9]	GLOBAL	[10]

2. fill in the relocation entries of the `.text` section of `main.o` and `name.o`. (5')

Relocation entries of `main.o`

Module	Offset	Type	Symbol Name	Addend
main.o	0000000a	R_X86_64_PC32	get_show	[1]
	00000014	[2]	[3]	[4]
	00000022	[5]	[6]	[7]
name.o	00000017	[8]	[9]	[10]

3. 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. (2'*4=8')

Name	Section	Type	Value
a	.data	OBJECT	00601050
b	.data	OBJECT	00601052
names	.data	OBJECT	00601060
shows	.data	OBJECT	00601080
main	.text	FUNC	00400526
get_name	.text	FUNC	0040056b
get_show	.text	FUNC	0040058d

Comparison of relocations

Module	Section	Before relocation	After Rel.
main.o	.text	9: e8 <u>00 00 00 00</u>	[1]
	.text	1f: 8b 04 85 <u>00 00 00 00</u>	[2]
name.o	.text	e: 8b 15 <u>00 00 00 00</u>	[3]
show.o	.data	20: <u>00 00 00 00 00 00 00 00</u>	[4]

4. Please write down the output of `main.c`. (5')

NOTE: You may need to refer to the symbol table of question 3.

Problem 5: Dynamic Linking (11 points)

ICSTA wrote two C programs as shown following: `subvec.c` and `dynamic_link.c`. We compile `subvec.c` as a shared library (`linux > gcc -shared -fpic -o libvector.so subvec.c`):

```
/* subvec.c */
1.  int delcnt;
2.  void subvec(int *x , int* y, int* z, int n) {
3.      for(int i=0;i<n; i++) {
4.          z[i] = x[i] - y [i];
5.      }
6.  }

/*dynamic_link.c */
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <dlfcn.h>
4.
5.  int x [2] = {1,2};
6.  int y [2] = {3,4}
7.  int z [2];
8.  int main(void) {
9.      /* we can call subvec() like any other function */
10.     subvec(x,y,z,2)
11.     printf("z = [%d %d]", z[0], z[1]);
12.
13.     return 0;
14. }
```

1. Please give two ways that we can linking the shared libraries `libvector.so` (NOTE: you can modify the `dynamic_link.c`) (4')

[1]

[2]

2. When using `-fpic` flag to compile the `subvec.c`, why compiler uses **GOT** to resolve reference of global variables in `libvector.so`, instead of relocating the global symbol when loading `libvector.so` (3')
3. After the shared libraries `libvector.so` was loaded in memory, please fill in the address of **GOT entry of subvec** before and after first invocation. Suppose that the address of **PLT[0]** is `0x404360`, the address of **PLT entry of subvec** is `0x404560`, and the address of **subvec ()** is `0x400128` , the value of **GOT[0]** is `0x406670` (4')

Before: [1]

After: [2]

Problem 6: Signal (22 points)

```
1. #define MAX_GENERATION 1
2. int generation = 0;
3. void divide(int n) {
4.     if (generation < MAX_GENERATION) {
5.         generation += 1;
6.         kill(-getpid(), SIGINT);
7.         if (fork() == 0) {
8.             printf("%d\n", getpid());
9.         }
10.    }
11. }
12. int main(void) {
13.     printf("%d\n", getpid());
14.     signal(SIGINT, divide);
15.     kill(getpid(), SIGINT);
16.     while (1);
17. }
```

Keith wrote a program to simulate cell division learned in the biology class. The source code of `cell.c` is shown above. NOTE:

- Assume all system calls are successful.
 - A child created via `fork` inherits a copy of its parent's set of currently blocked signals.
 - A child created via `fork` initially has an **empty** pending signal set.
 - When `./cell` is executed, its `pgid` is set to the same as its `pid`
 - We use `cell process(es)` to indicate the process of running `./cell` and all its children, grandchildren,
1. Is there any race modifying the global variable `generation` between an invoke of `divide` (when a `SIGINT` comes) and another invoke of `divide` (when another `SIGINT` comes), why? (3')
 2. Is there any race modifying the global variable `generation` between the parent process and its child process(es), why? (3')
 3. Suppose `MAX_GENERATION` is `1` in this question. Keith runs `./cell` and when the number of cell process(es) reaches stable,
 - a) How many cell process(es) are there? (2')
 - b) For each cell process, how many signal does it receive? (2')
 4. Suppose `MAX_GENERATION` is `2` in this question. When run `./cell` and the number of cell process(es) reaches stable, draw parent-child graph for possible case(s). In a parent-child graph, a process is drawn as a

circle and a parent process has an arrow pointing to his child process. (6')

5. Keith insert a line "`setpgid(0, 0);`" between **line 7** and **line 8**. Suppose **MAX_GENERATION** is **3** in this question. When run `./cell` and the number of cell process(es) reaches stable, draw parent-child graph for possible case(s). (6')