上 海 交 通 大 学 试 卷（ A 卷）

（ 2020 至 2021 学年 第 2 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称 计算机系统基础（系统软件） 成绩 _____

Problem 1: IO

1.

2. [1] [2]

[3] [4]

3.

4.

我承诺，我将严格遵守考试纪律。

题号	1	2	3	4	5				
得分									
批阅人(流水阅卷教师签名处)									

Problem 2: Process and Signal

1.

2.

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

[5] [6] [7]

4.

5.

Problem 3: Network

1.

2.

3. [1]

   [3]

4. [2]

Problem 4: Schedule

1. [1] [2] [3]

   [4] [5] [6]

   [7] [8] [9]

   [10] [11] [12]

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

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

3. Average turnaround time:

   Average response time:

4.

Problem 5: Lock

1. **[1]**

   [2]

2.

3. **a) lock->flag : _____**

   b) lock->flag : _____

4. **[3]**

   [4]

5.

6.

Problem 1: IO (20 points)

```
1.  #include "csapp.h"
2.  int main() {
3.      int fd1, fd2;
4.      char str[] = "abc";
5.      char c;
6.      fd1 = Open("ics.txt", O_RDWR, 0);
7.      fd2 = Open("ics.txt", O_RDWR, 0);
8.      Write(fd1, str, 3);
9.      Read(fd1, &c, 1); Write(fd2, &c, 1);
10.     Read(fd1, &c, 1);
11.     printf("%c", c); fflush(stdout);
12. A:
13.     Dup2(1, fd2);
14.     Read(fd1, &c, 1); Write(fd2, &c, 1);
15. B:
16.     if (Fork() == 0) {
17.         Dup2(fd1, 1);
18.     } else {
19.         Wait(NULL);
20.     }
21.     Close(fd1); Close(fd2);
22.     return 0;
23. }
```

Assumptions:

- Initially, **ics.txt** contains "**123456789**".
- No error occurs in the execution, and the read and write operations are atomic.

1. What are the values of **fd1** and **fd2** after the program executes to **Line 8**? (4')
2. Before the program executes to certain lines specified below, write down the output on the screen and the contents of **ics.txt**. (8')
   a) Before execution to **label A (Line 12):**

      Output on the screen: ___[1]___ Contents of **ics.txt**: ___[2]___

   b) Before execution to **label B (Line 15):**

      Output on the screen: ___[3]___ Contents of **ics.txt**: ___[4]___
3. **Before** the program executes to **label B (Line 15)**, draw the kernel data structures of open files (just like **Figure 10.14** in our textbook). (**Note**: you can use "terminal" to refer to ***standard output***). (4')

4. **Before** the **child process** closes fds **(Line 21)**, draw the kernel data structures of open files for **both parent and child** processes (just like **Figure 10.14** in our textbook). (4')

Problem 2: Process and Signals (19 Points)

```
1.  #include "csapp.h"
2.  volatile int sig_user1_cnt = 0;
3.  int child_exit_num = 0;
4.  int num = 3;
5.  int status = 40;
6.  pid_t pid[3];
7.
8.  void handle_siguser1(int num) { sig_user1_cnt++; }
9.
10. int main() {
11.     Signal(SIGUSR1, handle_siguser1);
12.     for (int i = 0; i < 3; i++) {
13.         status++;
14.         if ((pid[i] = Fork()) == 0) { // Child Process
15.             num++;
16.             printf("num=%d\n", num); fflush(stdout);
17.             kill(getppid(), SIGUSR1);
18.             exit(status);
19.         } else  // Parent Process
20.             num += 10;
21.     }
22.     while (waitpid(-1, &status, 0) > 0) {
23.         child_exit_num++;
24.         if (WIFEXITED(status)) {
25.             printf("status=%d\n", WEXITSTATUS(status)); fflush(stdout);
26.         }
27.     }
28.     printf("num=%d\n", num); fflush(stdout);
29.     printf("sig_user1_cnt=%d\n", sig_user1_cnt); fflush(stdout);
30.     return 0;
31. }
```

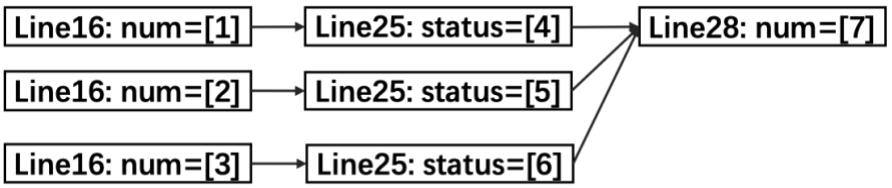
Assumptions:

- ✧ We assume OS may stop and resume the execution of each process arbitrarily.
- ✧ The kernel checks the pending bits when it switches from kernel to use (e.g., returning from system call). The return of signal handler will trigger **`sigreturn`** system call.

1. How many times will `fork()` be called in this program. (2’)

2. Will all the forked child processes be reaped by parent process finally? Why? (3’)

3. The following figure describes parts of contents printed on the screen by **Line 16**, **Line 25** and **Line 28**. In this figure, “**OutputX** -> **OutputY**” indicates that **OutputX** must appear before **OutputY** on the screen. You should fill the blanks, and make sure that your answer matches the order of each output. (7’)



4. The child process may terminate in any order. Show how to modify the codes at **Line 22** to make the **value** of `status` at **Line 25** printed at screen **in increasing order**. (**Hint**: You can refer to **Figure 8.19** in your textbook) (3’)

5. Show the **maximum value** and **minimum value** of `sig_user1_cnt` at **Line 29**. For each value, show a possible execution order processes may execute in. (4’)

Problem 3: Network (16 points)

Suppose a tiny web server that can serve dynamic contents to the clients. The server has different executable files to serve different client requests. The following code shows the core function that handles dynamic contents, along with a CGI program that queries a student's lab score:

```
1. // main function that handle dynamic contents
2. void serve(int fd, char* host, char* exe, char* qStr) {
3.    // Omit HTTP response header construction
4.    if (fork() == 0) {
5.        ___[1]___
6.        ___[2]___
7.        execve(exe, {NULL}, environ);
8.    }
9.    Wait(NULL);
10.}
11.// main routine of the CGI program
12.int main() {
13.   int score;
14.   if (strcmp("10.0.0.2", __[3]__)!=0) {
15.      fprintf(stderr, "Invalid IP!\r\n");fflush(stderr);
16.      exit(1);
17.   }
18.   // Omit score querying logic
19.   fprintf(stdout, "Score:%d\r\n", score);fflush(stdout);
20.   exit(0);
21.}
```

Assumptions:

- All executable files reside in the **cgi-bin** directory on the server, and the server uses the same strategy to parse requests in section 11.6 of our textbook.
- For function **`serve`**, parameter **`fd`** is a connected descriptor with client, **`host`** is the host name of the client, **`exe`** is the CGI program name to execute and serve the request, **`qStr`** is the query string of the request.
- Function **`strcmp`** at line 14 returns 0 if two parameter strings are the same.
- "␣" symbol represents a **whitespace** character.

Now suppose a student issues an http request to the server via his browser, the **URL** is:

`http://ipads.se.sjtu.edu.cn/cgi-bin/labgrade.cgi?student_id=100␣&lab=9`

1. Does the above URL contain valid arguments? (2’) If not, please describe how to correct it. (2’) (**Hint**: refer to section 11.5 in our textbook)

2. For the above URL (correct it if you think it is not valid in the previous problem), what is the name of the executable file that serves the request in web server? (2’) What are the arguments? (2’)

3. Our lab score querying CGI program only serves requests issued from the host “10.0.0.2” (As **Line 14-17** shows, other hosts will be rejected and prints an error message), please fill code in the blanks of [1] and [3] that pass necessary **parameters** (between server process and CGI program process) to achieve this check. (4’)

4. Fill necessary code in the blank of [2] to ensure all possible writes in the CGI program (**Line 15** and **19**) goes directly to the client. (4’) (**Hint**: the number of statements a blank can write is not limited)

Problem 4: Schedule (26 points)

Assume we have the following two jobs in the workload and **no I/O issues** are involved. Please fill in the following tables with the execution of CPU when we decide to use different schedule policies respectively. Suppose when a job arrives, it is added to the tail of a work queue. The RR policy selects the next job of the current job in the queue. The **RR** time-slice is 2ms. (**NOTE**: Time 0 means the task running during [0ms,1ms])

Job	Arrival Time	Length of Run-time
A	0ms	4ms
B	2ms	2ms
C	5ms	3ms
D	9ms	4ms

1. Assume we are using different scheduling policies, please calculate the **Average Turnaround Time, Average Response Time** and fill the **Job name** in the below table (when the CPU runs at Time 6): (12')

Policy	Average Turnaround Time	Average Response Time	CPU Time: 6ms
FIFO	[1]	[5]	[9]
SJF	[2]	[6]	[10]
STCF	[3]	[7]	[11]
RR	[4]	[8]	[12]

2. We decide to use **MLFQ** scheduling policy with **two priority queues**, the highest one has time-slice of **1ms**, the lowest one has time-slice of **2ms**. We use RR in each queue and priority boost **isn't supported**. Following table shows the execution of CPU. Please fill in the blanks. (8')

Assumptions:

- ✧ The arrival of new task triggers CPU reschedule.
- ✧ The time-slice of task interrupted by CPU reschedule will be reserved for next use.

Time(ms)	0	1	2	3	4	5	6
CPU	A	A	[1]	[2]	[3]	[4]	A
Time(ms)	**7**	**8**	**9**	**10**	**11**	**12**	
CPU	C	[5]	[6]	[7]	[8]	D	

3. Please calculate the average turnaround time and average response time of the

MLFQ scheduling policy we mentioned in the previous problem. (4')

4. What do you think are the advantages of **MLFQ** compared to other ones (e.g., FIFO)? (Note: You need to give **at least two** advantages) (2')

Problem 5: Lock (19 points)

Sam wants to implement a spinlock with **compare-and-swap(CAS)** instructions.

✧ **CAS** and **AtomicAdd** perform the same function as the code below shows, but in an **atomic** manner.

atomic instructions	spinlock with CAS

`int CAS(int *p, int exp, int new)`
`{`
`  int actual = *p;`
`  if (actual == exp)`
`  *p = new;`
`  return actual;`
`}`
 	
`void AtomicAdd(int *p, int value)`
`{`
`  *p = *p + value;`
`}`
	`typedef struct __lock_t {`
	`  int flag;`
	`} lock_t; // init to 0`
	
	`void lock(lock_t *lock) {`
	`  while(CAS(&lock->flag, [1], [2])`
	`  == 1); //spin`
	`}`
	
	`void unlock(lock_t *lock) {`
	`  lock->flag = 0;`
	`}`

1. Fill the blanks in `lock`. (2*2'=4')
2. Does this lock gaurantee **fairness**? Explain your answer through an example. (3')

Sam then extends the simple spinlock above to a **read-write lock** (rwlock). **Readers,** holding the **read locks,** should only perform read-only operations. The **writer,** holding the **write lock**, can read and modify the object. The rwlock guarantees the following two principles:

✧ **Principle 1**: The read lock can be acquired by multiples reader threads, i.e., multiple readers can run **concurrently**.

✧ **Principle 2**: The write lock must be held exclusively, i.e., a writer can **NOT** run concurrently with other writers and readers.

The code for **rwlock** is shown below. Read the code and answer the questions.

Hint: `0x80000000` is the 4-byte int with the most significant bit set to 1.

readers-writer lock

```
// lock->flag init to 0
void write_lock(lock_t *lock) {
    while(CAS(&lock->flag, 0, 0x80000000) != 0);
}

void read_lock(lock_t *lock) {
    int tmp;
    while(true) {
        tmp = lock->flag;
        if (tmp >= 0 && CAS(&lock->flag, tmp, tmp+1) == tmp)
            return;
    }
}

void write_unlock(lock_t *lock) {
    _________[3]___________
}

void read_unlock(lock_t *lock) {
    AtomicAdd(&lock->flag, __[4]__);
}
```

3. What is the value of `lock->flag` under each of the following cases? (2*1'=2')
   a) A single writer holds the lock.
   b) Five readers hold the lock simultaneously.
4. Fill the blanks in `write_unlock` and `read_unlock`. (2'*2=4')
5. The code above implements a **reader-friendly** rwlock (i.e., the lock perfers readers to writers). Please explain the reason. (Hint: consider the **fairness** for writers) (3')
6. How can you modify the implementation to make it a **writer-friendly** rwlock? Show your code and explain the reason.(3')