CS 392 Spring 2023 Systems Programming Midterm 2

- You have approximately 50 minutes;
- The exam is closed book, closed notes except your one-page crib sheet;
- Please write your full name and section number on the upperleft corner of the first page;
- By handing in this exam you acknowledge that you followed the honor code.

First name	
Last name	
Section Number	

For staff use only:

Q1.	Multiple Choices	/20
Q2.	True or False	/25
Q3.	The exec Family	/20
Q4.	Signals	/16
Q5.	Pipe Messages	/19
Q6.	Bonus	/2
	Total	/102

Q1. [20 pts] Multiple Choices

Each of the following questions has one **or** more answers. Select all that apply. When uncertain, choose the one you're most confident about.

1. [5 pts] Which of the following that are true regarding I/O?

- ☐ High level I/O deals with `FILE*` struct;
- ☐ There is one file descriptor table per system;
- ☐ Low-level I/O is buffered in user space;
- ☐ None of the above.

2. [5 pts] Which of the following statements about files are true? (choose all that apply):

- ☐ The same file descriptor number can correspond to different files for different processes;
- ☐ Reserved 0, 1, and 2 (stdin, stdout, stderr) file descriptors cannot be overwritten by a user program;
- ☐ File descriptions keep track of the file offset;
- ☐ An `lseek()` within one process may be able to affect the writing position for another process.

3. [5 pts] Assume all the function calls succeed and are executed as expected. Suppose that `montymole.txt` was empty before running this block of code. The following code was run:

```c
int main(int argc, char** argv) {
    int fd1 = open("montymole.txt", O_WRONLY);
    int fd2 = open("montymole.txt", O_WRONLY);
    write(fd1, "mole",  4);
    write(fd2, "whack", 5);
    write(fd2, "mole",  4);
    write(fd1, "mole",  4);
    write(fd1, "mole",  4);
    close(fd1);
    close(fd2);
}
```

Which of the following could be the content of "`montymole.txt`"?

- ☐ `whacmolemole`
- ☐ `whacmolek`
- ☐ `molewhackmolemolemole`
- ☐ `whackmolemole`
- ☐ Nothing

4. [5 pts] How many **new** processes could be created?

```c
#include <sys/types.h>
#include <unistd.h>
int main() {
    for (int i = 0; i < 2; i++) fork();
    return 0;
}
```

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

Q2. [25 pts] True or False

For the following questions, please explain your choice in less than 50 words. Wrong explanation and those exceeds 50 words (yes we do count it) are invalid.

1. [5 pts] After a call to `fork()`, `stdin`, `stdout`, and `stderr` are reset to their default states in the child process.

   ◯ True ◯ False

2. [5 pts] When a user program successfully calls `execv()`, its process is replaced with a new program. Everything about the old process is destroyed, including all of the CPU registers, program counter, stack, heap, background threads, file descriptors, and virtual address space.

   ◯ True ◯ False

3. [5 pts] Calling `pipe()` on an array like `int fds[2]` creates a read and write stream that local processes can use for unidirectional IPC. Under the hood, a pipe is implemented as a buffer in user space where the read end sits at the 0th index of the pipe and the write end sits at the 1st index of the pipe.

   ◯ True ◯ False

4. [5 pts] When you press `Ctrl+C` to a running program in the terminal, you are actually sending a `SIGQUIT` signal to the program, which makes it quit.

   ◯ True ◯ False

5. [5 pts] In general, communicating data between processes by FIFO is faster than using a regular file.

   ◯ True ◯ False

Q3. [20 pts] The exec Family

Command **grep** can search for a string in a file, and display the lines that include that string. For example:

```
$ grep cs392
```

will display all the lines from `stdin` that include the string `cs392`.

Below is a simple Linux C program which redirects `stdin` so that executed process greps the word “`cs392`” from the file “`cs392.txt`”. Assume:

- Calls to `open()`, `close()`, and `execvp()` succeed;
- All the necessary header files are included;
- `stdin` has a file descriptor of 0.

```
int main(int argc, char** argv) {
    char* grep_args[] = {"grep", "cs392", NULL};
    int in = open("cs392.txt", O_RDONLY);

    (Do not write your answer here)

    close(in); /* close unused file descriptors */
    execvp("grep", grep_args); /* execute grep */
}
```

1. [10 pts] Fill in the blank in the code above to finish the task. You need to provide two solutions that are equivalent. Solution 1 should have exactly one statement, while solution 2 can have as many statements as you like.

   Solution 1:

   Solution 2:

2. [10 pts] Based on the two solutions you provided above, which one do you prefer? Why?

Q4. [16 pts] Signals

Consider the following fragment of a program (which is missing lines of code). Fill in the blanks in the code to ensure that the output of this program is **always** the following two lines in the following order (under all interleavings or schedules):

```
val = 0
val = 1
```

No more than one function call (or system call) per blank! Your solution is not allowed to make assignments to the `val` variable. Also, no use of `printf()` or other print statements!

Hint: `SIGCONT` is to "wake up" a paused process. If you want to install signals, simply call `signal()` function.

```
void rectangle () {

    pid_t oval = ____________________
    kill(oval, SIGCONT);
}

void circle(int i) {
        val = 1;
}

int val = 0;
int main() {
    pid_t pid = fork();

    ____________________

    if (pid == 0) ____________________

    else ____________________

    printf("val = %d\n",val);
}
```

Q5. [19 pts] Pipe Messages

As discussed in class, strings are not the only type a pipe can transfer. We can also use pipes to transfer other types. Assume now we have two **unrelated** processes – a server and a client. The server sends a sequence of data point defined as follows where `long long int` takes 8 bytes:

```
struct Point {
    long long int x;
    long long int y;
};
```

The client will receive these data point and organize each struct object's `x` and `y` into separate lists, `list_x` and `list_y`.

1. [9 pts] Complete the server code on the left, assume function `read_point_list()` has been implemented. Each line should have only one statement.

2. [10 pts] Complete the client code on the right, **without** using `struct Point`. For the empty part, you can have as many statements as you like.

```
int main(int argc, char** argv) {
  struct Point list[10];
  read_point_list(list);
  char* path = "/tmp/whatever";

  __________________________________

  int fd = _________________________
  for (int i = 0; i < 10; i ++) {

    _________________________
  }
  close(fd);
  return 0;
}
```

```
int main(int argc, char** argv) {
    long long int list_x[10];
    long long int list_y[10];
    char* server_path = "/tmp/whatever";

    int fd = _________________________






    close(fd);
    return 0;
}
```

Q6. [2 pts] Bonus

Pikachu wrote a program where he forked a child process from the parent. He wanted to write his own signal handler, so that when he pressed `Ctrl+C`, the child process can use the handler to terminate its parent process. Assume he did successfully install the signal handler shown below before calling `fork()`. After he ran the program and pressed `Ctrl+C`, his program got stuck in the terminal. Could you help him figure out why?

```
void sig_handler(int sig) {
    if (sig == SIGINT) kill(getppid(), SIGINT);
}
```