Can you answer these tricky questions?

Let's give a try

FILES

Q1. Creating a file with read permissions in open system call and performing write. Will this work?

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

int main(int argc, char *argv[])
{
        int fd = open("hello.txt", O_WRONLY|O_CREAT, 0444);
        if (fd < 0)
                perror("failed to create file");
        else {
                printf("File created successfully\n");
                write(fd, "hello", sizeof("hello"));
                perror("write");
                close(fd);
        }
        return 0;
}
```

Q2. What will be the permissions of the file "hello.txt", if we don't pass the permissions in the open system call while creating file?

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

int main(int argc, char *argv[])
{
   int fd = open("hello.txt", O_RDWR|O_CREAT);
   if (fd < 0)
      perror("failed to create file");
   else
      printf("File created successfully\n");
   close(fd);
   return 0;
}
```

Q3. Will the "hello world" be printed on the console, if we write on stdin?

```
#include <stdio.h>

int main()
{
        write(0, "hello world", 12);
        return 0;
}
```

Q4. Will Open works in the below code?

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

int main(int argc, char *argv[])
{
        int fd;

        fd = open(".", O_RDWR|O_DIRECTORY);
        if (fd < 0) {
                perror("open");
        } else {
                printf("Opening directory successful\n");
                close(fd);
        }
        return 0;
}
```

Q5. Why the file created using the below code is not having the same permissions as mentioned in the third argument?

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

int main(int argc, char *argv[])
{
        int fd = open("hello.txt", O_RDWR|O_CREAT, 0666);
        if (fd < 0)
                perror("failed to create file");
        else
                printf("File created successfully\n");
        close(fd);
        return 0;
}
```

Q6. Will the "Hello World" printed if i run the below code in Linux?

```
#include <stdio.h>

int main () {
  freopen("/dev/null", "w", stdout);
  printf("Hello World\n");
  return 0;
}
```

Q7. Guess where "hello world" is written (on console/on hello.txt)?

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

int main()
{
	int fd;

	close(1);
	fd = open("hello.txt", O_WRONLY|O_CREAT, 0666);
	if (fd < 0) {
		perror("open");
	} else {
		printf("hello world\n");
	}
	return 0;
}
```

Q8. Consider there is a file 'linux.txt' in your home folder. Will the below code works fine in both cases (open, fopen)? (Note: You have permissions to open the file)

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

int main()
{
	int fd;
	FILE *fp;
	fd = open("~/linux.txt", O_RDWR);
	if (fd < 0) {
		perror("open failed");
	} else {
		printf("Opening file successful\n");
		close(fd);
	}

	fp = fopen("~/linux.txt", "r");
```

```
        if (fp) {
                printf("opening file successful\n");
                fclose(fp);
        } else {
                perror("open failed");
        }
        return 0;
}
```

Q9. Will the "hello world" get printed in the below code. Explain why?

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(int argc, char **argv)
{
        unlink(argv[0]);
        printf("hello world\n");
}
```

Q10. Will "hello world" get printed in the below code on Linux?

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

int main()
{
        close(1);
        printf("hello world\n");
        return 0;
}
```

Q11. Why is "bye world" printing first in the below code.

```
#include <stdio.h>

int main()
{
        printf("hello world");
        perror("bye world");
        return 0;
}
```

Q12. What happens if we perform lseek more than the file size? Guess the size of the file 'hello.txt' when we run this code and comment:

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

int main()
{
        int fd = open("hello.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666);
        write(fd, "hello", 5);
        lseek(fd, 1000, SEEK_END);
        write(fd, "bye", 3);
        close(fd);
        return 0;
}
```

Q13. Can I create a folder named 'abc' when I have a file 'abc' present at the same location in Linux?

PROCESSES

Q1. How many times "Hello world" will be printed in the below code?

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv)
{
   printf("Hello, world. ");
   fork();
}
```

Q2. Will the parent printf work if the child closes the standard ouput. Run the below code and comment?

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

int main()
{
pid_t pid;
pid = fork();
if (pid == 0) {
close(1);
_exit(0);
} else if (pid > 0) {
wait(NULL);
printf("hello world\n");
}
    return 0;
}
```

Q3. Will this below program ever ends?

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

int main(int argc, char *argv[]) {
printf("Hello Linux\n");
execl(argv[0], argv[0], NULL);
}
```

Q4. Will the file offset in the parent change when the child performs read/write operation on file. Run the below code and comment.

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

int main()
{
pid_t pid;
int fd = open("hello.txt", O_WRONLY|O_CREAT, 0644);

if (fd < 0) {
perror("open failed");
exit(EXIT_FAILURE);
}

pid = fork();
if (pid == 0) {
write(fd, "hello", sizeof("hello"));
close(fd);
exit(EXIT_SUCCESS);
} else if (pid > 0) {
wait(NULL);
printf("file offset:%ld\n", lseek(fd, 0, SEEK_CUR));
exit(EXIT_SUCCESS);
}
return 0;
}
```

Q5. Run the below code and check the name of the process using ps command (ps -ef). Try passing argument and without argument

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

int main(int argc, char *argv[])
{
	printf("hello world\n");
	if (argc == 2)
		strcpy(argv[0] , argv[1]);
	pause();
}
```

Q6. What happens to the file offset in the parent when the child process writes to the file? Try this code and comment.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main() {

int fd;
pid_t pid;

fd = open("output", O_CREAT|O_TRUNC|O_WRONLY, 0666);

write(fd, "hello", 5);
printf("offset before creating child:%u\n", lseek(fd, 0, SEEK_CUR));

pid = fork();
if (pid == 0) {
write(fd, " world", 6);
} else if (pid > 0) {
wait(NULL);
printf("offset after child terminates:%u\n", lseek(fd, 0, SEEK_CUR));
} else {
perror("fork failed");
}
close(fd);
return 0;
}
```

Q7. Why is the argc address different each time we run the executable?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        printf("Argc Address:%p\n", &argc);
        return 0;
}
```

Q8. What will happen when the child updates the shared memory region, will it affect the parent?

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        int *addr;
        addr = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
        switch (fork()) {
                case -1:
                        perror("fork");
                        exit(1);
                        break;
                case 0:
                        printf("Child started, value = %d\n", *addr);
                        (*addr)++;
                        munmap(addr, sizeof(int));
                        exit(EXIT_SUCCESS);
                default:
                        wait(NULL);
                        printf("In parent, value = %d\n", *addr);
                        munmap(addr, sizeof(int));
                        exit(EXIT_SUCCESS);
        }
}
```

Q9. How many times ‘hello’ will be printed? (HINT: RLIMIT_STACK)

```
#include <stdio.h>
int hello()
{
        char buf[8192*256];
        printf("%s\n", __func__);
        hello();
}
int main()
{
        hello();
}
```

Signals

Q1. Guess the output of this code:

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

int main()
{
int *ptr = NULL;
signal(SIGSEGV, SIG_IGN);
*ptr = 10;
return 0;
}
```

Q2. What happens when signal is delivered to the process who called sleep(), will the remaining sleep continue after the signal handler?

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

void sighandler(int signum)
{
        write(1, "sighandler", 10);
}

int main(int argc, char *argv[])
{
        signal(SIGUSR1, sighandler);
        sleep(20);
        printf("After Sleep\n");
        return 0;
}
```

Q3. Will the signal handler executes when you run the below code? To find the answer use: "man -s7 signal"

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

void sighandler(int signum)
{
        printf("Signal Handler of SIGKILL\n");
}

int main()
{
        signal(SIGKILL, sighandler);
        raise(SIGKILL);
        while(1);
}
```

Q4. Will the child receive SIGALARM if the parent has started a timer using alarm system call before calling fork()? Try this code and find out.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/signal.h>

void alarm_handler(int signum) {
        printf("%s:PID:%d\n", __func__, getpid());
}

int main(){

    pid_t pid;
    printf("Process PID:%d\n", getpid());
    //set up alarm handler
    signal(SIGALRM, alarm_handler);
    //schedule alarm for 2 second
    alarm(2);
    pid = fork();
    if (pid == 0) {
        printf("Hi, I am Child with PID:%d\n", getpid());
    }
    //wait for signal
    pause();
return 0;
}
```

Q5. Try this code and check what happens:

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>

int main() {
    kill(-1, SIGKILL);
}
```

Q6. What happens to pending signals after I fork?

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

void sighandler(int signum)
{
printf("%s:PID:%d\n", __func__, getpid());
}

int main(){
   sigset_t newset, oldset;
   pid_t pid;

   printf("My PID:%d\n", getpid());
   signal(SIGINT, sighandler);
   sigemptyset(&newset);
   sigaddset(&newset, SIGINT);
   sigprocmask(SIG_SETMASK, &newset, &oldset);
   int i = 0;
   for(i=1; i<=8; i++){
printf("I am masking SIGINT for 8 seconds, please press CTRL-C!\n");
sleep(1);
  }
  pid = fork();
  printf("New process is created using fork()\n");
  sigprocmask(SIG_SETMASK, &oldset, NULL);
   for(i=1; i<=4; i++){
printf("Now I am having the old sigset without any mask\n");
sleep(1);
  }
  pause();
   return 0;
}
```

Misc

Q1. What is the size of 'a' variable? Please comment why it is so?

```
int main(void)
{
   unsigned a;
   printf("size of a:%d\n", sizeof(a));
   return 0;
}
```

Q2. Will the threadid(tid) and processid(pid) same for a process with a single thread? Run the below code and find out.

```
#include <unistd.h>
#include <stdio.h>
#include <sys/syscall.h>

int main() {

        printf("Process ID:%u\t Thread ID:%lu\n", getpid(), syscall(SYS_gettid));
        return 0;
}
```

Q3. Will heap section exists in the address space of process using the below code? Use /proc to find out.

```
#include <stdio.h>
int main()
{
   while(1);
}
```