ICS Homework 7

Cao Shengcao

8.9

Process pair	Concurrent?
AB	N
AC	Y
AD	Y
BC	Y
BD	Y
CD	Y

8.18

The program can be illustrated as below:

```
3                                .-------0-------2
1         .-------atexit----------Fork----1-------2
2         |                       .-------0
0   --------Fork--------------------Fork----1
```

Performing a topological sorting, we can see ACE are possible outputs.

8.24

First let's see what's `psignal(3)` by typing `man psignal`:

```
PSIGNAL(3)                             Linux Programmer's Manual
PSIGNAL(3)

NAME
       psignal, psiginfo - print signal message

SYNOPSIS
       #include <signal.h>

       void psignal(int sig, const char *s);
       void psiginfo(const siginfo_t *pinfo, const char *s);

       extern const char *const sys_siglist[];

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       psignal(): _SVID_SOURCE || _BSD_SOURCE
       psiginfo(): _XOPEN_SOURCE >= 700 || _POSIX_C_SOURCE >= 200809L
       sys_siglist: _BSD_SOURCE

DESCRIPTION
       The  psignal() function displays a message on stderr consisting of the string s, a colon,
a
       space, a string describing the signal number sig, and a trailing newline.  If the string
s
       is  NULL  or  empty,  the colon and space are omitted.  If sig is invalid, the message
dis-
       played will indicate an unknown signal.

       The psiginfo() function is like psignal(), except that it displays  information  about
the
       signal  described  by pinfo, which should point to a valid siginfo_t structure.  As well
as
       the signal description, psiginfo() displays information about the origin of the signal,
and
       other  information  relevant to the signal (e.g., the relevant memory address for
hardware-
       generated signals, the child process ID for SIGCHLD, and the user ID and process ID of
the
       sender, for signals set using kill(2) or sigqueue(3)).

       The array sys_siglist holds the signal description strings indexed by signal number.

RETURN VALUE
       The psignal() and psiginfo() functions return no value.
...
```

Seems that this is the function that will output some error info as is shown in the exercise. Now let's modify the code.

```c
#include "csapp.h"
#define N 2

int main()
{
    int status, i;
    pid_t pid;

    /* Parent creates N children */
    for (i = 0; i < N; i++)
    if ((pid = Fork()) == 0)  /* Child */
    {
        *(int*)(0x401d36) = 0; /* Child writes in the .text section
                                  (which is certainly read-only) */
        exit(100+i);
    }

    /* Parent reaps N children in no particular order */
    while ((pid = waitpid(-1, &status, 0)) > 0)
    {
        if (WIFEXITED(status))
            printf("child %d terminated normally with exit status=%d\n",
                   pid, WEXITSTATUS(status));
        else
        {
            if (WIFSIGNALED(status))    /* Parent prints error info */
            {
                char buf[100];
                sprintf(buf, "child %d terminated by signal %d", pid,
                        WTERMSIG(status));
                psignal(WTERMSIG(status), buf);
            }
            else
                printf("child %d terminated abnormally\n", pid);
        }
    }

    /* The only normal termination is if there are no more children */
    if (errno != ECHILD)
    unix_error("waitpid error");

    exit(0);
}
```

Compile the code along with `csapp.h` `csapp.c` by typing `gcc 8.24.c csapp.h csapp.c -lpthread -o 8.24` and we shall see the result of running the program:

```
child 3993 terminated by signal 11: Segmentation fault
child 3992 terminated by signal 11: Segmentation fault
```

10.6

We can actually compile and run the program and the result will be `fd2 = 4`.

The descriptor returned by `open()` is the smallest descriptor that is not currently open in the process. Each process begins with three open files: standard input (descriptor 0), standard output (descriptor 1), and standard error (descriptor 2).

At first `fd1` is 3, after descriptor 0, 1, and 2. When `fd2` is closed, descriptor 4 can be used again. So we can see `fd2 = 4`.

10.9

"Bad file descriptor" indicates that in the child process the file does not have an independent descriptor 3. Therefore `stdin` should be redirected to the file, and the previous file descriptor is closed.

```
if (Fork() == 0)
{
    int fd = Open("foo.txt", O_RDONLY, 0);
    Dup2(fd, 0);
    Close(fd);
    Execve("fstatcheck", argv, envp);
}
```