Operating Systems - Programming Assignment 1

資管二 B11705060 盧沛宏

Problem 3.13

Result

```
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ gcc p1.c
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./a.out
usage: a.out <non-negative integer>
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./a.out -5
an integer >= 0 is required
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./a.out 20
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./a.out 50
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418
 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 1655
80141 267914296 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./a.out 0

jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$
```

Code Explanation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

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

Here we take arguments from the command line.

```c
if (argc != 2) {
        fprintf(stderr, "usage: a.out <non-negative integer>\n");
        return -1;
    }

    int seq = atoi(argv[1]);

    if (seq < 0) {
        fprintf(stderr, "an integer >= 0 is required\n");
        return -1;
    }
```

We perform necessary error checks on the format of the parameters.

```c
    pid_t pid;
    pid = fork();

    if (pid < 0) // error occured
    {
        fprintf(stderr, "fork failed\n");
        return -1;
    } else if (pid == 0) // child process
    {
        unsigned long long v0 = 0, v1 = 1;

        for (int i = 0; i < seq && i < 2; i++) {
```

```
            fprintf(stdout, (i == 0) ? "%d" : " %d", i); // fib_0 = 0, fib_1 = 1
        }

        for (int i = 2; i < seq; i++) {
            unsigned long long v2 = v0 + v1;
            v0 = v1;
            v1 = v2;
            fprintf(stdout, " %lld", v2);
        }

        fprintf(stdout, "\n");
```

We perform forking just like in the course slides. Note that all Fibonacci sequence output is done by the child process.

```
    } else // parent process
    {
        wait(NULL);
        // child complete
    }
    return 0;
}
```

Parent process waits until child completes before it exits.

Problem 3.17

Result

```
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ gcc p2.c -o p2.out
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p2.out
usage: a.out <non-negative integer less than 10>
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p2.out -2
an integer >= 0 is required
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p2.out 15
an integer <= MAX_SEQUENCE = 10 is required
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p2.out 5
0 1 1 2 3
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p2.out 10
0 1 1 2 3 5 8 13 21 34
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$
```

Code Explanation

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

#define MAX_SEQUENCE 10

typedef struct {
    long fib_sequence[MAX_SEQUENCE];
    int sequence_size;
} shared_data;
```

Define the struct according to the problem description.

```c
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: a.out <non-negative integer less than %d>\n", MAX_SEQUENCE);
        return -1;
    }

    int cmdParam = atoi(argv[1]);

    if (cmdParam < 0) {
        fprintf(stderr, "an integer >= 0 is required\n");
        return -1;
    }

    if (cmdParam > MAX_SEQUENCE) {
        fprintf(stderr, "an integer <= MAX_SEQUENCE = %d is required\n", MAX_SEQUENCE);
        return -1;
    }
```

Accepts the parameter passed from the command line, and performs necessary error checking so it is a non-negative integer not greater than `MAX_SEQUENCE`.

```c
    int shm_fd = shm_open("fib", O_CREAT | O_RDWR, 0666);
    ftruncate(shm_fd, sizeof(shared_data));
    void *ptr = mmap(0, sizeof(shared_data), PROT_WRITE | PROT_READ, MAP_SHARED, shm_fd, 0);
```

Create a shared memory object named "fib" with the size `shared_data`, and attaches it to the pointer `ptr` for the parent process.

```c
    ((shared_data *)ptr)->sequence_size = cmdParam;
```

Set the value of `sequence_size` to the parameter from the command line `cmdParam`.

```c
    pid_t pid;
    pid = fork();

    if (pid < 0) // error occured
    {
        fprintf(stderr, "fork failed\n");
        return -1;
```

Fork the child process.

```c
    } else if (pid == 0) // child process
    {
        long v0 = 0, v1 = 1;

        for (int i = 0; i < ((shared_data *)ptr)->sequence_size && i < 2; i++)
            ((shared_data *)ptr)->fib_sequence[i] = i; // fib_0 = 0, fib_1 = 1

        for (int i = 2; i < ((shared_data *)ptr)->sequence_size; i++) {
            long v2 = v0 + v1;
            v0 = v1;
            v1 = v2;
            ((shared_data *)ptr)->fib_sequence[i] = v2;
        }
```

Calculate the Fibonacci sequence numbers and store it in the shared memory through the `ptr` pointer, which is accessible because the child is a copy of the parent.

```
    } else // parent process
    {
        wait(NULL);
        // child complete
```

Invoke the `wait()` system call to wait for the child to finish.

```
        for (int i = 0; i < ((shared_data *)ptr)->sequence_size; i++)
            fprintf(stdout, i ? " %ld" : "%ld", ((shared_data *)ptr)->fib_sequence[i]);
        fprintf(stdout, "\n");
```

Output the values of the Fibonacci sequence in the shared memory segment through the `ptr` pointer.

```
        shm_unlink("fib");
    }
    return 0;
}
```

Detach and remove the shared memory segment.

Problem 3.20

Result

```
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ gcc p3.c -o p3.out
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ls
p1.c  p1.out  p2.c  p2.out  p3.c  p3.out  testfile.txt
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p3.out dne.txt
usage: a.out <source file> <destination file>
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p3.out dne.txt copy.txt
open source file dne.txt failed
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p3.out testfile.txt copy.txt
wrote 129 bytes
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ diff -y -s testfile.txt copy.txt
Lorem ipsum dolor sit amet,                                     Lorem ipsum dolor sit amet,
consectetur adipiscing elit,                                    consectetur adipiscing elit,
sed do eiusmod tempor incididunt                                sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.                               ut labore et dolore magna aliqua.Files testfile.
txt and copy.txt are identical
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ ./p3.out p1.out p1.out.bak
wrote 16256 bytes
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$ diff -s p1.out p1.out.bak
Files p1.out and p1.out.bak are identical
jerrymk@jerrymk-VirtualBox:/media/sf_os-pa/pa1$
```

It tests for incorrect command parameter format and errors when reading file (destination file will not be created in this case). If the file copy is successful, the bytes written will be outputted. Binary files such as executables can also be copied.

Code Explanation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define BUFFER_SIZE 1024
#define READ_END 0
#define WRITE_END 1
```

```c
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "usage: a.out <source file> <destination file>\n");
        return -1;
    }
```

Check for command parameter format.

```c
    int pipe_fd[2];
    if (pipe(pipe_fd) == -1) {
        fprintf(stderr, "pipe failed\n");
        return -1;
    }
```

Creates an ordinary pipe and assign `pipe_fd` to the file descriptors.

```c
    FILE *f_src = fopen(argv[1], "rb"); // read file in binary mode

    if (f_src == NULL) {
        fprintf(stderr, "open source file %s failed\n", argv[1]);
        return -1;
    }

    unsigned char buffer[BUFFER_SIZE];
    size_t sz; // number of read bytes
    size_t bytes_read = 0;
    while (sz = fread(buffer, sizeof(buffer[0]), BUFFER_SIZE, f_src)) {
        write(pipe_fd[WRITE_END], buffer, sz);
        bytes_read += sz;
    }
```

Open the source file in binary read mode and read up to `BUFFER_SIZE` bytes at a time and write it to the pipe's write end.

```c
    close(pipe_fd[WRITE_END]);
```

Close the pipe's write end as it is no longer needed.

```c
    pid_t pid;
    pid = fork();

        if (pid < 0) // error occured
    {
        fprintf(stderr, "fork failed\n");
        return -1;
```

Fork out a child process.

```c
    } else if (pid == 0) // child process
    {
        FILE *f_dest = fopen(argv[2], "wb"); // write file in binary mode (overwrite mode)
        unsigned char buffer[BUFFER_SIZE];
        size_t bytes_wrote = 0;
        size_t sz; // number of read bytes

        if (f_dest == NULL) {
            fprintf(stderr, "open/create destination file %s failed\n", argv[2]);
```

```
            return -1;
        }
```

Try to open or create the destination file in binary write mode.

```
        while (sz = read(pipe_fd[READ_END], buffer, BUFFER_SIZE)) {
            if (sz < 0) {
                fprintf(stderr, "pipe read error\n");
                return -1;
            }
            fwrite(buffer, sizeof(buffer[0]), sz, f_dest);
            bytes_wrote += sz;
        }
```

Read the source file's content from the pipe's read end and write it to the destination file.

```
        if (bytes_read == bytes_wrote)
            fprintf(stdout, "wrote %ld bytes\n", bytes_wrote);
        else
            fprintf(stderr, "file copy error: read %ld bytes, wrote %ld bytes\n", bytes_read,
bytes_wrote);
```

Check the bytes read and written as a form of a very basic copy verification.

```
    } else // parent process
    {
        wait(NULL);
    }
    close(pipe_fd[READ_END]);
    return 0;
}
```

Parent process will wait until the child process is done writing the file. Then the pipe's read end will be closed as well (for both the child and the parent processes).