**Operating Systems**

Lab Report #3

**Process Control**

Name

Student ID

# Objectives

Briefly describe the objective of this lab assignment.

# Code and Execution

## Assignment 1: Basic fork() Usage

I#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

int x = 100;

pid\_t pid = fork();

if (pid < 0) {

// fork 失败

fprintf(stderr, "Fork failed");

return 1;

} else if (pid == 0) {

// 子进程

printf("child fork: x = %d\n", x);

x += 10;

printf("After child fork changed: x = %d\n", x);

} else {

// 父进程

printf("parent fork: x = %d\n", x);

x += 20;

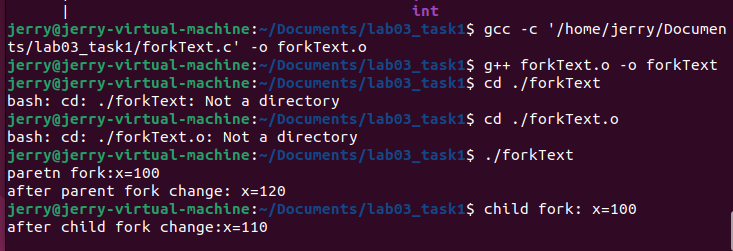
printf("After parent changed: x = %d\n", x);

}

return 0;

}

Provide the output or screenshots of your program execution.



## Assignment 2: File Descriptor Inheritance

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

int fd;

fd = open("testfile.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd < 0) {

perror("open");

exit(1);

}

pid\_t pid = fork();

if (pid < 0) {

perror("fork");

close(fd);

exit(1);

} else if (pid == 0) {

// 子进程

const char \*child\_msg = "Message from child fork\n";

write(fd, child\_msg, sizeof(child\_msg));

printf("Child fork: load success\n");

} else {

// 父进程

const char \*parent\_msg = "Message from parent fork\n";

write(fd, parent\_msg, sizeof(parent\_msg));

printf("Parent fork: load success\n");

}

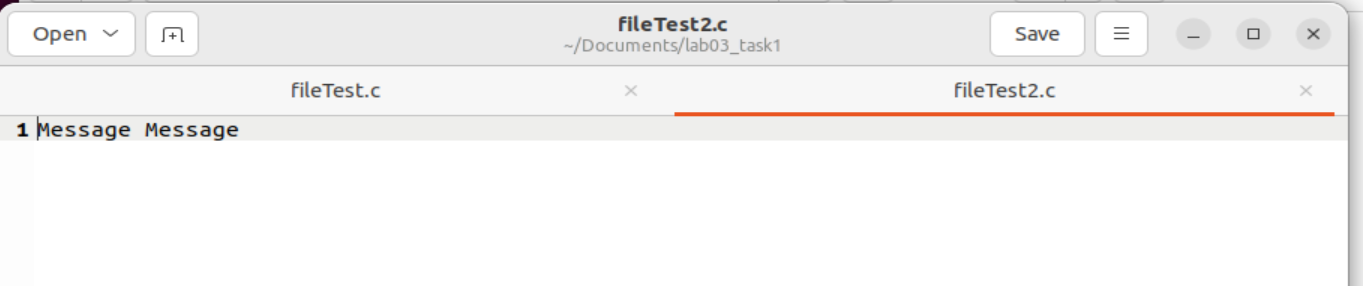
close(fd);

return 0;

}

Provide the output or screenshots of your program execution.

## 



## Assignment 3: Standard Output Closure

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

// 如果创建子进程失败，输出错误信息并退出程序

perror("fork");

exit(1);

} else if (pid == 0) {

// 这是子进程执行的代码块

// 关闭标准输出

close(STDOUT\_FILENO);

// 尝试打印字符串到标准输出

printf("这是子进程的输出\n");

// 为了确保printf输出，强制刷新缓冲区

fflush(stdout);

} else {

// 这是父进程执行的代码块

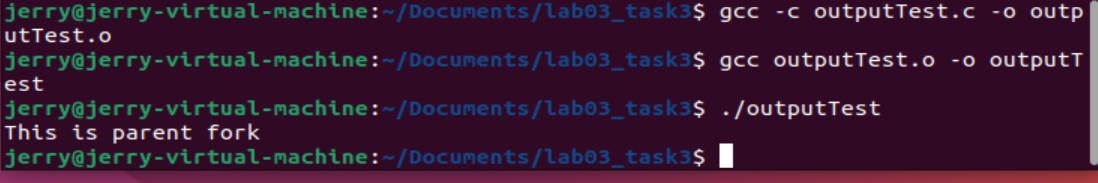
// 父进程继续正常执行

printf("这是父进程的输出\n");

}

return 0;

}



## Assignment 4: Advanced Process Control

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

// 如果创建子进程失败，输出错误信息并退出程序

perror("fork");

exit(1);

} else if (pid == 0) {

// 这是子进程执行的代码块

// 关闭标准输出

close(STDOUT\_FILENO);

// 尝试打印字符串到标准输出

printf("这是子进程的输出\n");

// 为了确保printf输出，强制刷新缓冲区

fflush(stdout);

} else {

// 这是父进程执行的代码块

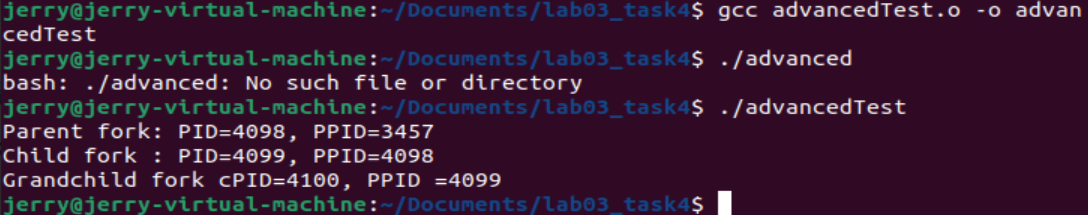
// 父进程继续正常执行

printf("这是父进程的输出\n");

}

return 0;

}



## Assignment 5: Pipes for Inter-process Communication

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int main() {

int fd[2]; // 文件描述符数组，fd[0] 为读端，fd[1] 为写端

pid\_t pid;

const char \*msg = "这是从父进程发送的消息\n";

char buffer[100];

// 创建管道

if (pipe(fd) == -1) {

perror("pipe");

exit(1);

}

// 创建子进程

pid = fork();

if (pid < 0) {

perror("fork");

exit(1);

} else if (pid == 0) {

// 子进程

close(fd[1]); // 关闭写端

// 从管道读端读取消息

read(fd[0], buffer, sizeof(buffer));

printf("子进程读取到的消息: %s", buffer);

close(fd[0]); // 关闭读端

} else {

// 父进程

close(fd[0]); // 关闭读端

// 向管道写端写入消息

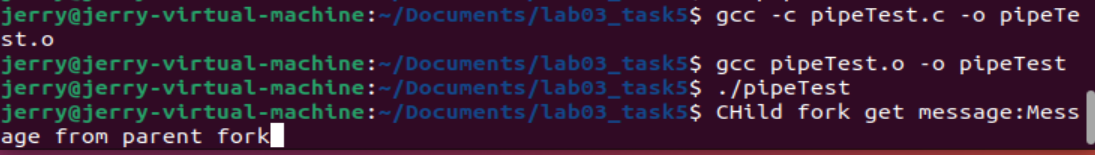
write(fd[1], msg, strlen(msg) + 1);

close(fd[1]); // 关闭写端

}

return 0;

}



# Analysis

Analyze the behavior of each program. Explain the observed outputs, discuss any challenges encountered, and how they were resolved.

# Conclusion

Summarize your findings and experiences from this lab assignment.

# References

List any references or resources you used to complete this lab assignment, if any.