Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС. Обеспечение обмена данных между процессами посредством каналов.**

Студент: Лисин Роман Сергеевич

Группа: М80-206Б-20

Вариант: 13

Преподаватель: Соколов Андрей Алексеевич

Дата: 9.10.2021

Оценка: 5

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

13 вариант: Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «\_».

**Листинг программы**

**child1.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include "ctype.h" //to\_lower

#include "string.h"

#include "unistd.h"

int main() {

int n;

read(0, &n, sizeof(int));

// fprintf(stderr, "%d\n", n);

char str[n + 1];

write(1, &n, sizeof(int));

read(0, str, sizeof(char) \* n);

for (int i = 0; i < n; ++i) {

str[i] = tolower(str[i]);

// fprintf(stderr, "%c\n", str[i]);

}

write(1, str, sizeof(char) \* n);

return 0;

}

**child2.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include "ctype.h" //to\_lower

#include "string.h"

#include "unistd.h"

int main() {

int n;

read(0, &n, sizeof(int));

// printf("%d", n);

write(1, &n, sizeof(int));

char str[n + 1];

read(0, str, sizeof(char) \* n);

for (int i = 0; i < n; ++i) {

if (str[i] == ' ') {

str[i] = '\_';

}

//fprintf(stderr, "%c\n", str[i]);

}

write(1, str, sizeof(char) \* n);

return 0;

}

**var13.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include "ctype.h" //to\_lower

#include "string.h"

#include "unistd.h"

char\* get\_string() {

int len = 0, capacity = 10;

char\* s = (char\*)malloc(10 \* sizeof(char));

if (s == NULL) {

perror("Can't read a string");

exit(6);

}

char c;

read(0, &c, sizeof(char));

while (c != '\n') {

s[len++] = c;

if (c == EOF) {

break;

}

if (len == capacity) {

capacity \*= 2;

s = (char\*)realloc(s, capacity \* sizeof(char));

if (s == NULL) {

perror("Can't read a string");

exit(6);

}

}

read(0, &c, sizeof(char));

};

s[len] = '\0';

return s;

}

int main() {

int fd1[2], fd2[2], fd3[2];

pipe(fd1);

pipe(fd2);

pipe(fd3);

if (pipe(fd1) < 0 || pipe(fd2) < 0 || pipe(fd3) < 0) {

perror("Ошибка при создании pipe");

return -1;

}

pid\_t pid1 = fork();

if (pid1 == -1) {

perror("Ребёнок не был создан");

return -1;

}

if (pid1 == 0) { // child 1

close(fd1[1]); // we only read this pipe

close(fd2[0]); // we only write to this pipe

// close(fd3[0])

// close(fd3[1]);

fflush(stdout);

// sleep(1.2);

if (dup2(fd1[0], 0) < 0) {

perror("Can't redirect stdin for child process");

}

if (dup2(fd2[1], 1) < 0) {

perror("Can't redirect stdout for child process");

}

execl("child1", "child1", NULL);

perror("Can't execute child process");

}

else {

pid\_t pid2 = fork();

if (pid2 == -1) {

perror("Ребёнок не был создан");

return -1;

}

if (pid2 == 0) { //child 2

//waitpid(pid1, 0, 0);

close(fd1[0]);

close(fd2[1]); // clean the house after party

close(fd3[0]);

close(fd1[1]);

// waitpid(pid1, 0, 0);

// sleep(3);

fflush(stdout);

if (dup2(fd2[0], 0) < 0) {

perror("Can't redirect stdin for child process");

}

if (dup2(fd3[1], 1) < 0) {

perror("Can't redirect stdout for child process");

}

execl("child2", "child2", NULL);

}

else { // parent

close(fd2[0]);

close(fd3[1]);

close(fd2[1]);

close(fd1[0]);

//fflush(stdout);

int n;

//read(0, &n, sizeof(int));

char\* str = get\_string();

n = strlen(str);

//scanf("%s:[^ ]", str);

//getline(str, n, stdin);

//char\* str;

//fgets(str, 3, stdin);

write(fd1[1], &n, sizeof(int));

//n = -1;

write(fd1[1], str, sizeof(char) \* n);

int status;

if (wait(&status) == -1){

perror("wait");

}

if (wait(&status) == -1){

perror("wait");

}

if (!WIFEXITED(status) || (WIFEXITED(status) && WEXITSTATUS(status) != 0)) {

perror("error in children");

}

// sleep(0.2);

char str\_cool[n + 1];

read(fd3[0], &n, sizeof(int));

read(fd3[0], str\_cool, sizeof(char) \* n);

//printf("%d\n", n);

//printf("\n");

str\_cool[n] = '\0';

printf("%s\n", str\_cool);

close(fd3[0]);

close(fd1[1]);

}

}

return 0;

}

**Makefile**

all: child1 child2 var13

child1:

gcc child1.c -o child1

child2:

gcc child2.c -o child2

var13:

gcc var13.c -o var13

clean:

rm -rf child1 child2 var13

**Примеры работы**

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

N waS H

n\_was\_h

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

N KHSIOSJCBKJXLZ KL

n\_khsiosjcbkjxlz\_kl

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

N

n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

LJnenso o

\_\_ljnenso\_o

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

OppO

oppo

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJWOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjwooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo

roma@DESKTOP-JD58QU2:~/os\_lab2$ ./var13

M kMla lLLL mLMLcql; npoK

\_\_m\_kmla\_llll\_mlmlcql;\_npok

**Вывод**

Данная работа была для меня сложной из-за того, что я в первый раз работал с процессами. Но в то же время было крайне интересно передавать строку между тремя процессами, контролируя последовательность выполнения программы. Эта работа научила меня правильно создавать pipes на Си, создавать новые процессы, переназначать ввод и вывод программы в pipe с помощью dup2. Думаю, что эти знания мне пригодятся в будущем.