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: Гречников А.В.

Группа: М80-206Б-20

Вариант: 20

Преподаватель: Соколов Андрей Алексеевич

Дата: 27.11.2021

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 решения 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. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

20 вариант. Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

**Алгоритм решения задачи**

Родительский процесс считывает имена двух файлов, открывает их и создает два канала. После создания, дочерние процессы переопределяют свой ввод (на канал) и вывод (на файл) и запускают саму программу дочернего процесса при помощи exec.

В это время родительский процесс начинает считывать строки. Для того, чтобы не хранить всю считанную строку в памяти, заведем буфер длиной 10. Пока длина строки не превышает 10, добавляем введенный символ в буфер. Если строка закончилась, посылаем буфер в первый дочерний процесс. Если строка не закончилась, посылаем буфер во второй дочерний процесс, а затем каждый следующий символ посылаем также во второй дочерний процесс.

Дочерние процессы считывают символы из каналов в динамически расширяющиеся массивы. О завершении строки сигнализирует символ переводы строки. При этом происходит печать строки в обратном порядке.

**Листинг программы**

**main.c**

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

char fn1[256];

char fn2[256];

if (scanf("%s", fn1) <= 0) {

perror("scanf error");

return -1;

}

if (scanf("%s", fn2) <= 0) {

perror("scanf error");

return -1;

}

int file1 = open(fn1, O\_WRONLY | O\_CREAT, S\_IWUSR | S\_IRUSR);

if (file1 < 0) {

perror("open error");

return -1;

}

int file2 = open(fn2, O\_WRONLY | O\_CREAT, S\_IWUSR | S\_IRUSR);

if (file2 < 0) {

perror("open error");

return -1;

}

int fd1[2], fd2[2];

if (pipe(fd1) != 0) {

perror("pipe error");

return -1;

}

if (pipe(fd2) != 0) {

perror("pipe error");

return -1;

}

int id1 = fork();

if (id1 == -1) {

perror("fork error");

return -1;

} else if (id1 == 0) {

close(file2);

close(fd2[0]);

close(fd2[1]);

close(fd1[1]);

if (dup2(fd1[0], fileno(stdin)) == -1) {

perror("dup2 error");

return -1;

}

if (dup2(file1, fileno(stdout)) == -1) {

perror("dup2 error");

return -1;

}

close(file1);

char \* const \* null = NULL;

if (execv("child", null) == -1) {

perror("execv error");

return -1;

}

} else {

int id2 = fork();

if (id2 == -1) {

perror("fork error");

return -1;

} else if (id2 == 0) {

close(file1);

close(fd1[0]);

close(fd1[1]);

close(fd2[1]);

if (dup2(fd2[0], fileno(stdin)) == -1) {

perror("dup2 error");

return -1;

}

if (dup2(file2, fileno(stdout)) == -1) {

perror("dup2 error");

return -1;

}

close(file2);

char \* const \* null = NULL;

if (execv("child", null) == -1) {

perror("execv error");

return -1;

}

} else {

close(fd1[0]);

close(fd2[0]);

close(file1);

close(file2);

char c;

char str[10];

str[0] = '\0';

int n = 0;

while (scanf("%c", &c) > 0) {

if (c != '\n') {

if (n < 10) {

str[n] = c;

} else if (str[0] != '\0') {

for (int i = 0; i < 10; ++i) {

write(fd2[1], &str[i], sizeof(char));

str[i] = '\0';

}

write(fd2[1], &c, sizeof(char));

} else {

write(fd2[1], &c, sizeof(char));

}

++n;

} else {

if (str[0] != '\0') {

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

write(fd1[1], &str[i], sizeof(char));

str[i] = '\0';

}

write(fd1[1], &c, sizeof(char));

} else {

write(fd2[1], &c, sizeof(char));

}

n = 0;

}

}

close(fd1[1]);

close(fd2[1]);

}

}

return 0;

}

**child.c**

#include "unistd.h"

#include "stdio.h"

#include "stdlib.h"

char \* add(char \*str, int cap, int n, char c) {

if (n == cap) {

cap \*= 2;

str = (char \*)realloc(str, sizeof(char) \* cap);

if (str == NULL) {

perror("realloc error");

exit(1);

}

}

str[n] = c;

return str;

}

int main() {

char c;

int n = 0;

int cap = 256;

char \*str = (char \*)malloc(sizeof(char) \* cap);

if (str == NULL) {

perror("malloc error");

return 1;

}

while(read(fileno(stdin), &c, sizeof(char)) > 0) {

if (c != '\n') {

str = add(str, cap, n, c);

++n;

} else {

for (int i = n-1; i >= 0; i--) {

printf("%c", str[i]);

}

printf("%c", c);

n = 0;

}

}

free(str);

}

**Пример работы**

artem@ideapad:~$ ./main

1.txt

2.txt

1234567890

12345

abcdefghijklmnop

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

sssssssssssssssssssss

sss

artem@ideapad:~$ cat 1.txt

0987654321

54321

sss

artem@ideapad:~$ cat 2.txt

ponmlkjihgfedcba

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

sssssssssssssssssssss

**Вывод**

В ходе лабораторной работы я познакомился с созданием процессов в ОС Linux, а также со способами их взаимодействия. Переопределение стандартных дескрипторов ввода/вывода является очень удобным механизмом, так как позволяет исполнять программу как в обычном режиме, так и в качестве вспомогательного процесса. Это в свою очередь позволяет делать exec\*, который запускает другой исполняемый файл, наследующий все дескрипторы. Однако при этом надо следить, чем переопределен, например, вывод и не пытаться использовать функции форматированного вывода, если переопределенный дескриптор поддерживает только побайтовую запись.