Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-211Б-23

Студент: Малеев В.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 13.11.24

Москва, 2024

Постановка задачи

Вариант 19.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют все гласные из строк.

Общий метод и алгоритм решения

Кратко опишите системные вызовы, которые вы использовали в лабораторной работе.

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int fd[2]) – создание неименованного канала для передачи данных между процессами.
- ssize_t read(int fd, void *buf, size_t count); – считывает до count байт из файлового дескриптора fd в буфер buf.
- ssize_t write(int fd, const void *buf, size_t count); – записывает до count байт из буфера buf в файловый дескриптор fd.
- int execl(char *fname, char *arg0, ..., char *argN, NULL); – замена образа памяти процесса.
- int dup2(int oldfd, int newfd); – переназначение файлового дескриптора.
- int open(const char *pathname, int flags, mode_t mode) – открытие\создание файла.
- int close(int fd); – Закрывает файловый дескриптор fd.
- void exit(int status) – завершения выполнения процесса и возвращение статуса.

Создание каналов: Программа создает два канала (pipe1 и pipe2), чтобы организовать передачу данных между родительским процессом и двумя дочерними процессами.

Ввод и открытие файлов: Программа запрашивает у пользователя имена файлов, в которые будут записаны выходные данные каждого из дочерних процессов. Открывает эти файлы с режимом O_WRONLY | O_CREAT | O_TRUNC | O_APPEND что позволяет создавать новый файл или перезаписывать существующий.

Создание дочерних процессов:

Первый дочерний процесс (child1): Порождается с помощью вызова fork(). Дочерний процесс перенастраивает стандартный ввод на чтение из pipe1 и стандартный вывод на запись в file1_fd. Запускает внешний исполняемый файл ./child1 с помощью execl(). Программа обрабатывает входные данные и удаляет гласные.

Второй дочерний процесс (child2): Аналогично первому, процесс перенастраивает стандартный ввод на чтение из pipe2 и стандартный вывод на запись в file2_fd. Запускает внешний исполняемый файл ./child2.

Родительский процесс: Закрывает неиспользуемые окончания каналов. В цикле считывает строки с стандартного ввода и направляет их в один из каналов, выбирая с вероятностью PROBABILITY (0.8), отправляет строку в pipe1, иначе — в pipe2. Ожидание завершения дочерних

процессов: Родительский процесс использует wait() дважды, чтобы дождаться завершения обоих дочерних процессов. :)

Код программы

parent.c

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/wait.h>
#include <errno.h>

#define BUFFER_SIZE 1024
#define PROBABILITY 0.8

void print_error(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, strerror(errno), strlen(strerror(errno)));
    write(STDERR_FILENO, "\n", 1);
    exit(EXIT_FAILURE);
}

void print_message(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
}

int main() {
    int pipe1[2], pipe2[2];

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        print_error("Ошибка при создании pipe.\n");
    }

    srand(getpid());

    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;

    // Ввод имени файла для child1
    print_message("Введи имя файла для child1: ");
    bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE);
    if (bytes_read <= 0) {
        print_error("Ошибка при чтении имени файла для child1.\n");
    }
    buffer[bytes_read - 1] = '\0';

    int file1_fd = open(buffer, O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file1_fd == -1) {
        print_error("Ошибка при открытии файла для child1.\n");
    }

    // Ввод имени файла для child2
    print_message("Введи имя файла для child2: ");
    bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE);
    if (bytes_read <= 0) {
        print_error("Ошибка при чтении имени файла для child2.\n");
    }
    buffer[bytes_read - 1] = '\0';

    int file2_fd = open(buffer, O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
```

```
    if (file2_fd == -1) {
        print_error("Ошибка при открытии файла для child2.\n");
    }

    pid_t pid1 = fork();
    if (pid1 == -1) {
        print_error("Ошибка при создании дочернего процесса child1");
    } else if (pid1 == 0) { // Первый дочерний процесс (child1)
        close(pipe1[1]);
        dup2(pipe1[0], STDIN_FILENO); // Читаем из pipe1
        dup2(file1_fd, STDOUT_FILENO); // Пишем в файл для child1
        execl("./child1", "./child1", NULL);
        print_error("Ошибка при запуске программы первого дочернего процесса.\n");
    }

    pid_t pid2 = fork();
    if (pid1 == -1) {
        print_error("Ошибка при создании дочернего процесса child2");
    } else if (pid2 == 0) { // Второй дочерний процесс (child2)
        close(pipe2[1]);
        dup2(pipe2[0], STDIN_FILENO); // Читаем из pipe2
        dup2(file2_fd, STDOUT_FILENO); // Пишем в файл для child2
        execl("./child2", "./child2", NULL);
        print_error("Ошибка при запуске программы второго дочернего процесса.\n");
    }

    // Родительский процесс
    close(pipe1[0]);
    close(pipe2[0]);
    close(file1_fd);
    close(file2_fd);

    print_message("Вводите строчечки:\n");

    while ((bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE)) > 0) {
        if (((double)rand() / RAND_MAX) < PROBABILITY) {
            if (write(pipe1[1], buffer, bytes_read) != bytes_read) {
                print_error("Ошибка при записи в pipe1.\n");
            }
        } else {
            if (write(pipe2[1], buffer, bytes_read) != bytes_read) {
                print_error("Ошибка при записи в pipe2.\n");
            }
        }
    }

    close(pipe1[1]);
    close(pipe2[1]);

    wait(NULL);
    wait(NULL);

    return 0;
}
```

child1.c/child2.c

```
#include <unistd.h>
#include <string.h>

#define BUFFER_SIZE 1024
#define VOWELS "AEIOUaeiou"
```

```
void remove_vowels(char *str) {
    char *ptr = str;
    while (*str) {
        if (!strchr(VOWELS, *str)) {
            *ptr++ = *str;
        }
        str++;
    }
    *ptr = '\0';
}

int main() {
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;

    while ((bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE)) > 0) {
        buffer[bytes_read] = '\0';
        remove_vowels(buffer);
        write(STDOUT_FILENO, buffer, strlen(buffer));
    }

    return 0;
}
```

Протокол работы программы

Тестирование:

```
vladislavmaleev@192 src % ./parent
Введи имя файла для child1: output1.txt
Введи имя файла для child2: output2.txt
Вводите строчечки:
Hello World!
Test String
Test String
Test String
Test String
Test String
Test String
Test String
Test String
Test String
1234!@#$
1234!@#$
1234!@#$
1234!@#$
Beautiful piece of code!
Beautiful piece of code!
Beautiful piece of code!
```

```
Beautiful piece of code!
Beautiful piece of code!
```

Dtruss:

```
vladislavmaleev@192 src % sudo dtruss ./parent
SYSCALL(args)           = return
open("output1.txt\0", 0x609, 0x180)             = 7 0
write(0x2, "\320\222\320\262\320\265\320\264\320\270 \320\270\320\274\321\217
\321\204\320\260\320\271\320\273\320\260 \320\264\320\273\321\217 child2: \0",
0x2C)               = 44 0
read(0x0, "output2.txt\n\0", 0x400)             = 12 0
open("output2.txt\0", 0x609, 0x180)             = 8 0
fork()          = 4967 0
fork()          = 4968 0
close(0x3)              = 0 0
close(0x5)              = 0 0
close(0x7)              = 0 0
close(0x8)              = 0 0
write(0x2, "\320\222\320\262\320\276\320\264\320\270\321\202\320\265
\321\201\321\202\321\200\320\276\321\207\320\265\321\207\320\272\320\270:\n\0",
0x23)               = 35 0
read(0x0, "Hello World!\n\0", 0x400)            = 13 0
write(0x4, "Hello World!\n\0", 0xD)             = 13 0
read(0x0, "Test String\n\0", 0x400)             = 12 0
write(0x6, "Test String\n\0", 0xC)              = 12 0
read(0x0, "Test String\n\0", 0x400)             = 12 0
write(0x4, "Test String\n\0", 0xC)              = 12 0
read(0x0, "Test String\n\n\0", 0x400)           = 12 0
write(0x4, "Test String\n\n\0", 0xC)            = 12 0
read(0x0, "Test String\n\0", 0x400)             = 12 0
write(0x4, "Test String\n\0", 0xC)              = 12 0
read(0x0, "Test String\n\0", 0x400)             = 12 0
write(0x4, "Test String\n\0", 0xC)              = 12 0
read(0x0, "Test String\n\0", 0x400)             = 12 0
write(0x4, "Test String\n\0", 0xC)              = 12 0
read(0x0, "Test String\n\n\0", 0x400)           = 12 0
write(0x6, "Test String\n\n\0", 0xC)            = 12 0
read(0x0, "Test String\n\0", 0x400)             = 12 0
write(0x6, "Test String\n\0", 0xC)              = 12 0
read(0x0, "Test String\n\0", 0x400)             = 12 0
write(0x6, "Test String\n\0", 0xC)              = 12 0
close(0x4)              = 0 0
close(0x6)              = 0 0
```

Output1.txt:

```
Hll Wrld!
Tst Strng
Tst Strng
Tst Strng
Tst Strng
Tst Strng



1234!@#$
1234!@#$
1234!@#$
1234!@#$
Btfl pc f cd!
Btfl pc f cd!
Btfl pc f cd!
Btfl pc f cd!
```

Output2.txt:

```
Tst Strng
Tst Strng
Tst Strng
Tst Strng

Btfl pc f cd!
```

Вывод

Таким образом, программа иллюстрирует использование базовых системных вызовов для создания процессов и организации их работы с файлами и межпроцессным взаимодействием с помощью каналов.