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-213Б-23

Студент: Арсельгов А. Б.

Преподаватель: Бахарев В.Д

Оценка: _____

Дата: 06.10.24

Москва, 2024

Постановка задачи

Вариант 21.

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.

Вариант 21) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает неименованный канал для передачи данных между процессами.
- void exit(int status) — завершение выполнения процесса и возвращение статуса.
- int dup2(int oldfd, int newfd) — переназначение файлового дескриптора.
- int close(int fd) — закрыть файл.
- int execl() - заменяет текущий процесс на новый процесс, загружая исполняемый файл.
- int open() - открытие/создание файла.
- int write() - вывод на экран сообщение.
- int read() - ввод с клавиатуры.

Код программы

Parent.c

```
#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <stdlib.h>

#include <errno.h>
```

```
#include <sys/wait.h>

#define BUFFER_SIZE 1024

void reverse_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}

int main() {
    int pipe1[2], pipe2[2];
    char buffer[BUFFER_SIZE];
    int count = 1;
    char filename1[BUFFER_SIZE];
    char filename2[BUFFER_SIZE];

    const char* pipe_error = "Error creating pipe.\n";
    const char* child1_error = "Error creating process 1.\n";
    const char* child11_error = "Error opening file for child1.\n";
    const char* child2_error = "Error creating process 2.\n";
    const char* child22_error = "Error opening file for child2.\n";

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        write(2, pipe_error, strlen(pipe_error));
```

```
    exit(EXIT_FAILURE);
}

write(1, "Enter a filename for child1: ", strlen("Enter a filename for child1: "));
read(0, filename1, BUFFER_SIZE);
filename1[strcspn(filename1, "\n")] = '\0';

write(1, "Enter a filename for child2: ", strlen("Enter a filename for child2: "));
read(0, filename2, BUFFER_SIZE);
filename2[strcspn(filename2, "\n")] = '\0';



pid_t pid1 = fork();
if (pid1 == -1) {
    write(2, child1_error, strlen(child1_error));
    exit(EXIT_FAILURE);
}

if (pid1 == 0) {
    int fd1 = open(filename1, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd1 == -1) {
        write(2, child11_error, strlen(child11_error));
        exit(EXIT_FAILURE);
    }

    close(pipe1[1]);
    dup2(pipe1[0], 0);
    dup2(fd1, 1);
    close(pipe1[0]);
```

```
    close(fd1);

    execl("./child1", "child1", NULL);
    write(2, "exec error for child1.\n", strlen("exec error for child1.\n"));
    exit(EXIT_FAILURE);
}

pid_t pid2 = fork();
if (pid2 == -1) {
    write(2, child2_error, strlen(child2_error));
    exit(EXIT_FAILURE);
}

if (pid2 == 0) {
    int fd2 = open(filename2, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd2 == -1) {
        write(2, child22_error, strlen(child22_error));
        exit(EXIT_FAILURE);
    }

    close(pipe2[1]);
    dup2(pipe2[0], 0);
    dup2(fd2, 1);
    close(pipe2[0]);
    close(fd2);

    execl("./child2", "child2", NULL);
    write(2, "exec error for child2.\n", strlen("exec error for child2.\n"));
```

```
    exit(EXIT_FAILURE);
}



close(pipe1[0]);
close(pipe2[0]);



while (1) {
    write(1, "Enter the line: ", 16);
    ssize_t bytes_read = read(0, buffer, BUFFER_SIZE);
    if (bytes_read <= 0) {
        exit(EXIT_SUCCESS);
    }
    buffer[bytes_read - 1] = '\0';

    if (strlen(buffer) == 0) {
        break;
    }

    if (count % 2 == 0) {
        write(pipe2[1], buffer, bytes_read);
    } else {
        write(pipe1[1], buffer, bytes_read);
    }
    count++;
}
```

```
    close(pipe1[1]);
    close(pipe2[1]);
  return 0;
}
```

Child1.c

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

#define BUFFER_SIZE 1024

void reverse_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}

int main() {
    char buffer[BUFFER_SIZE];

    while (1) {
        ssize_t bytes_read = read(0, buffer, BUFFER_SIZE);
        if (bytes_read <= 0) {
            exit(EXIT_SUCCESS);
        }
        buffer[bytes_read - 1] = '\0';

        if (strlen(buffer) == 0) {
```

```
            break;
        }

        reverse_string(buffer);
        write(1, buffer, strlen(buffer));
        write(1, "\n", 1);
    }

    return 0;
}
```

Child2.c

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

#define BUFFER_SIZE 1024

void reverse_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}

int main() {
    char buffer[BUFFER_SIZE];

    while (1) {
        ssize_t bytes_read = read(0, buffer, BUFFER_SIZE);
```

```
        if (bytes_read <= 0) {
            exit(EXIT_SUCCESS);
        }
        buffer[bytes_read - 1] = '\0';

        if (strlen(buffer) == 0) {
            break;
        }

        reverse_string(buffer);
        write(1, buffer, strlen(buffer));
        write(1, "\n", 1);
    }

    return 0;
}
```

Протокол работы программы

Тестирование:

```
user@adamarselgov:~/MAI_OS/lab01/src$ ./test
Enter a filename for child1: 1.txt
Enter a filename for child2: 2.txt
Enter the line: teyeh
Enter the line: uyirter
Enter the line: tyuyrtr
Enter the line: rtuyrty
Enter the line: tyr
Enter the line: uhrt
Enter the line:
user@adamarselgov:~/MAI_OS/lab01/src$
```

```
user@adamarselgov:~/MAI_OS/lab01/src$ gcc parent.c -o parent
user@adamarselgov:~/MAI_OS/lab01/src$ strace ./parent
execve("./parent", ["./parent"], 0x7ffc86aec530 /* 76 vars */) = 0
brk(NULL)                               = 0x5896df433000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x73aad71a1000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или
каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=83739, ...}) = 0
mmap(NULL, 83739, PROT_READ, MAP_PRIVATE, 3, 0) = 0x73aad718c000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\
0\1\0\0\0\220\243\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x73aad6e00000
mmap(0x73aad6e28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x73aad6e28000
mmap(0x73aad6fb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x73aad6fb0000
mmap(0x73aad6fff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1fe000) = 0x73aad6fff000
mmap(0x73aad7005000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x73aad7005000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x73aad7189000
arch_prctl(ARCH_SET_FS, 0x73aad7189740) = 0
set_tid_address(0x73aad7189a10)         = 47116
set_robust_list(0x73aad7189a20, 24)     = 0
rseq(0x73aad718a060, 0x20, 0, 0x53053053) = 0
mprotect(0x73aad6fff000, 16384, PROT_READ) = 0
mprotect(0x5896dd7c2000, 4096, PROT_READ) = 0
mprotect(0x73aad71d9000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
munmap(0x73aad718c000, 83739)           = 0
pipe2([3, 4], 0)                        = 0
pipe2([5, 6], 0)                        = 0
write(1, "Enter a filename for child1: ", 29Enter a filename for child1: ) = 29
read(0,
```

```
"\n", 1024)                     = 1
write(1, "Enter a filename for child2: ", 29Enter a filename for child2: ) = 29
read(0,
"\n", 1024)                     = 1
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x73aad7189a10) = 47137
Error opening file for child1.
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x73aad7189a10) = 47138
close(3Error opening file for child2.
)                                = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=47137, si_uid=1000,
si_status=1, si_utime=0, si_stime=0} ---
close(5)                                = 0
write(1, "Enter the line: ", 16Enter the line: )        = 16
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=47138, si_uid=1000,
si_status=1, si_utime=0, si_stime=0} ---
read(0,
"\n", 1024)                     = 1
close(4)                                = 0
close(6)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
user@adamarselgov:~/MAI_OS/lab01/src$
```

Вывод

Научился, как с помощью вызова `fork()` создавать новые процессы. Это важная часть многозадачности, когда родительский процесс порождает дочерние процессы, которые могут выполнять параллельные задачи. Понимание работы `fork()` позволяет управлять процессами в ОС. Применение `pipe()` научило студента тому, как родительский процесс может передавать данные дочерним процессам через каналы. Студент увидел, как важны потоки данных и их направление (родитель -> дочерние через каналы) при межпроцессном взаимодействии. Использование системных вызовов, таких как `open()`, `dup2()`, позволило студенту понять, как файлы открываются и обрабатываются на уровне ОС. Он научился переопределять стандартные потоки ввода-вывода (stdin, stdout) для перенаправления данных в файлы и процессы.