Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Заитова Е.А

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 29.11.24

Москва, 2024

Постановка задачи

Вариант 11.

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 записывает строки в разделяемую память поочередно зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Общий метод и алгоритм решения

Использованные системные вызовы:

- write — используется для вывода сообщений на стандартные потоки (STDOUT и STDERR).
- read — используется для ввода строк с клавиатуры.
- shmget — для создания сегментов разделяемой памяти для передачи данных между процессами.
- shmat — для подключения процессов к разделяемой памяти.
- fork — для создания дочерних процессов, которые будут обрабатывать данные.
- execv — для запуска дочерних процессов с передачей аргументов (в данном случае имена файлов для записи).
- close — для закрытия файловых дескрипторов после завершения записи.
- shmdt — для отсоединения от сегментов разделяемой памяти.
- shmctl — для удаления сегментов разделяемой памяти после завершения работы программы.

Программа создает два дочерних процесса, которые используют разделяемую память для получения строк от родительского процесса. Родительский процесс записывает строки в разделяемую память поочередно, а дочерние процессы обрабатывают эти строки, удаляя из них гласные, и записывают результат в отдельные файлы.

Код программы

main.c

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

#define BUFFER_SIZE 1024

void handling(char *str) {
    int len = strlen(str);
    char result[BUFFER_SIZE];
    int j = 0;

    for (int i = 0; i < len; i++) {
        if (str[i] != 'a' && str[i] != 'e' && str[i] != 'i' && str[i] != 'o' && str[i] != 'u' &&
            str[i] != 'A' && str[i] != 'E' && str[i] != 'I' && str[i] != 'O' && str[i] != 'U') {
            result[j++] = str[i];
        }
    }
    result[j] = '\0';
    strcpy(str, result);
}

int main() {
    char buffer[BUFFER_SIZE];
    int count = 1;
    char filename1[BUFFER_SIZE], filename2[BUFFER_SIZE];
    const char* msg;

    int shmid1 = shmget(IPC_PRIVATE, BUFFER_SIZE, IPC_CREAT | 0666);
```

```
if (shmid1 == -1) {
    msg = "shmget failed for child 1";
    write(STDERR_FILENO, msg, strlen(msg));
    exit(EXIT_FAILURE);
}

int shmid2 = shmget(IPC_PRIVATE, BUFFER_SIZE, IPC_CREAT | 0666);
if (shmid2 == -1) {
    msg = "shmget failed for child 2";
    write(STDERR_FILENO, msg, strlen(msg));
    exit(EXIT_FAILURE);
}

char *shared_memory1 = (char *)shmat(shmid1, NULL, 0);
if (shared_memory1 == (char *)-1) {
    msg = "shmat failed for child 1";
    write(STDERR_FILENO, msg, strlen(msg));
    exit(EXIT_FAILURE);
}

char *shared_memory2 = (char *)shmat(shmid2, NULL, 0);
if (shared_memory2 == (char *)-1) {
    msg = "shmat failed for child 2";
    write(STDERR_FILENO, msg, strlen(msg));
    exit(EXIT_FAILURE);
}

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

msg = "Enter a filename for child2: ";
write(STDOUT_FILENO, msg, strlen(msg));
```

```
read(STDIN_FILENO, filename2, BUFFER_SIZE);
filename2[strcspn(filename2, "\n")] = '\0';

pid_t pid1 = fork();
if (pid1 == -1) {
    const char* msg = "Error: failed to spawn new proccess\n";
    write(STDERR_FILENO, msg, strlen(msg) + 1);
    exit(EXIT_FAILURE);
}

if (pid1 == 0) {
    int fd1 = open(filename1, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd1 == -1) {
        const char* msg = "Error opening file for child 1";
        write(STDERR_FILENO, msg, strlen(msg) + 1);
        exit(EXIT_FAILURE);
    }

    while (1) {
        if (strlen(shared_memory1) > 0) {
            handling(shared_memory1);
            write(fd1, shared_memory1, strlen(shared_memory1));
            write(fd1, "\n", 1);
            memset(shared_memory1, 0, BUFFER_SIZE);
        }
        usleep(100000);
    }

    close(fd1);
    exit(EXIT_SUCCESS);
}

pid_t pid2 = fork();
if (pid2 == -1) {
```

```
    const char* msg = "Error: failed to spawn new proccess\n";
    write(STDERR_FILENO, msg, strlen(msg) + 1);
    exit(EXIT_FAILURE);
}

if (pid2 == 0) {
    int fd2 = open(filename2, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd2 == -1) {
        const char* msg = "Error opening file for child 2";
        write(STDERR_FILENO, msg, strlen(msg) + 1);
        exit(EXIT_FAILURE);
    }

    while (1) {
        if (strlen(shared_memory2) > 0) {
            handling(shared_memory2);
            write(fd2, shared_memory2, strlen(shared_memory2));
            write(fd2, "\n", 1);
            memset(shared_memory2, 0, BUFFER_SIZE);
        }
        usleep(100000);
    }

    close(fd2);
    exit(EXIT_SUCCESS);
}

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

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

        if (count % 2 == 1) {
            strncpy(shared_memory1, buffer, BUFFER_SIZE);
        } else {
            strncpy(shared_memory2, buffer, BUFFER_SIZE);
        }

        if (count % 2 == 1) {
            const char* msg = "write in child1\n";
            write(STDOUT_FILENO, msg, strlen(msg));
        } else {
            const char* msg = "write in child2\n";
            write(STDOUT_FILENO, "write in child2\n", strlen(msg));
        }

        count++;
        usleep(100000);
    }

    shmdt(shared_memory1);
    shmdt(shared_memory2);
    shmctl(shmid1, IPC_RMID, NULL);
    shmctl(shmid2, IPC_RMID, NULL);

    return 0;
```

child.c

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

```
#include <string.h>

#define BUFSIZ 4096

void handling(char* str, char* result) {
    char* p_str = str;
    char* p_result = result;

    while (*p_str) {
        if (*p_str != 'a' && *p_str != 'e' && *p_str != 'i' && *p_str != 'o' && *p_str != 'u' &&
            *p_str != 'A' && *p_str != 'E' && *p_str != 'I' && *p_str != 'O' && *p_str != 'U') {
            *p_result++ = *p_str;
        }
        ++p_str;
    }

    *p_result = '\0';
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        const char* msg = "Invalid amount parametres\n";
        write(STDERR_FILENO, msg, strlen(msg));
        exit(EXIT_FAILURE);
    }

    char buffer[BUFSIZ];
    ssize_t bytes;

    int file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file == -1) {
        const char* msg = "Error: failed to open file\n";
        write(STDERR_FILENO, msg, strlen(msg));
        exit(EXIT_FAILURE);
```

```
    }

    char result[BUFSIZ];
    while(bytes = read(STDIN_FILENO, buffer, BUFSIZ)) {
        buffer[bytes - 1] = '\0';
        handling(buffer, result);

        write(STDOUT_FILENO, result, strlen(result));
        write(STDOUT_FILENO, "\n", 1);

        write(file, result, strlen(result));
        write(file, "\n", 1);

    }

    close(file);

    return 0;
}
```

} Протокол работы программы

Тестирование

$./a.out

Enter a filename for child1: child1.txt

Enter a filename for child2: child2.txt

Enter the line: hello

write in child1

Enter the line: im testing

write in child2

Enter the line: this

write in child1

Enter the line: program

write in child2

Enter the line: poka

write in child1

Enter the line: ^C

$ cat child1.txt

hll

ths

pk

$ cat child2.txt

m tstng

prgrm

Strace

execve("./a.out", ["./a.out"], 0x7ffdfc9a1a50 /* 26 vars */) = 0

brk(NULL) = 0x55a71a15e000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff9962fee0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fbcb5944000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=18139, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18139, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fbcb593f000

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\0P\237\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

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fbcb5716000

mprotect(0x7fbcb573e000, 2023424, PROT_NONE) = 0

mmap(0x7fbcb573e000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fbcb573e000

mmap(0x7fbcb58d3000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7fbcb58d3000

mmap(0x7fbcb592c000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7fbcb592c000

mmap(0x7fbcb5932000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fbcb5932000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fbcb5713000

arch_prctl(ARCH_SET_FS, 0x7fbcb5713740) = 0

set_tid_address(0x7fbcb5713a10) = 20142

set_robust_list(0x7fbcb5713a20, 24) = 0

rseq(0x7fbcb57140e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fbcb592c000, 16384, PROT_READ) = 0

mprotect(0x55a6eb144000, 4096, PROT_READ) = 0

mprotect(0x7fbcb597e000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7fbcb593f000, 18139) = 0

shmget(IPC_PRIVATE, 1024, IPC_CREAT|0666) = 52

shmget(IPC_PRIVATE, 1024, IPC_CREAT|0666) = 53

shmat(52, NULL, 0) = 0x7fbcb597d000

shmat(53, NULL, 0) = 0x7fbcb5943000

write(1, "Enter a filename for child1: ", 29Enter a filename for child1:) = 29

read(0, child1.txt

"child1.txt\n", 1024) = 11

write(1, "Enter a filename for child2: ", 29Enter a filename for child2:) = 29

read(0, child2.txt

"child2.txt\n", 1024) = 11

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fbcb5713a10) = 20186

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fbcb5713a10) = 20187

write(1, "Enter the line: ", 16Enter the line:) = 16

read(0, hello

"hello\n", 1024) = 6

write(1, "write in child1\n", 16write in child1

) = 16

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=100000000}, NULL) = 0

write(1, "Enter the line: ", 16Enter the line:) = 16

read(0, im testing

"im testing\n", 1024) = 11

write(1, "write in child2\n", 16write in child2

) = 16

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=100000000}, NULL) = 0

write(1, "Enter the line: ", 16Enter the line:) = 16

read(0, this

"this\n", 1024) = 5

write(1, "write in child1\n", 16write in child1

) = 16

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=100000000}, NULL) = 0

write(1, "Enter the line: ", 16Enter the line:) = 16

read(0, program

"program\n", 1024) = 8

write(1, "write in child2\n", 16write in child2

) = 16

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=100000000}, NULL) = 0

write(1, "Enter the line: ", 16Enter the line:) = 16

read(0, ^Cstrace: Process 20142 detached

<detached ...>

Вывод

В этой лабораторной работе реализована система межпроцессного взаимодействия с использованием разделяемой памяти, для обработки строк из ввода пользователя. Программа

создает два дочерних процесса с помощью fork() и делит данные между ними через разделяемую память, синхронизируя доступ с помощью семафоров. В ходе работы система выполняет ввод строк, фильтрует их по четности, записывая в соответствующие буферы, и передает данные дочерним процессам для дальнейшей обработки. Это дает возможность использовать различные механизмы межпроцессного взаимодействия (каналы и разделяемую память), а также показывает важность синхронизации процессов при работе с общими ресурсами.