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-214Б-23

Студент: Ткаченко Е.А.

Преподаватель:

Оценка: _____

Дата:

Москва, 2024

Постановка задачи

Вариант 19.

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.

Общий метод и алгоритм решения

- **sem_open()** – Создает или открывает именованный семафор.
- **sem_post()** – Увеличивает значение семафора (сигнализирует о доступности ресурса).
- **sem_wait()** – Уменьшает значение семафора (ожидание доступа к ресурсу).
- **sem_close()** – Закрывает дескриптор семафора.
- **sem_unlink()** – Удаляет именованный семафор из системы.
- **shm_open()** – Создает или открывает объект разделяемой памяти.
- **ftruncate()** – Устанавливает размер разделяемой памяти.
- **mmap()** – Отображает разделяемую память в адресное пространство процесса.
- **munmap()** – Удаляет отображение разделяемой памяти из адресного пространства.
- **shm_unlink()** – Удаляет объект разделяемой памяти из системы.

Алгоритм решения

Программа начинается с создания разделяемой памяти и двух семафоров, которые используются для обмена данными между родительским процессом и двумя дочерними процессами. Разделяемая память служит общей областью для передачи строк от родителя к дочерним процессам, а семафоры обеспечивают синхронизацию доступа к этим данным.
Родительский процесс сначала создает дочерние процессы с помощью fork().
После этого он ожидает пользовательский ввод. Введенные строки записываются в разделяемую память, а затем, с вероятностью 80%, передаются на обработку первому дочернему процессу, иначе — второму. Выбор осуществляется случайным образом, используя генерацию случайного числа.
Каждый дочерний процесс работает параллельно. Получив строку из разделяемой памяти через свой семафор, процесс удаляет из нее все гласные буквы и записывает результат в файл, имя которого пользователь вводит при запуске программы.
Родительский процесс продолжает передавать строки до тех пор, пока пользователь не введет команду "exit". После этого он отправляет специальный сигнал завершения дочерним процессам, завершает их выполнение и корректно
освобождает все ресурсы (разделяемую память и семафоры).
В итоге программа организует параллельное выполнение двух дочерних процессов, которые обрабатывают строки и записывают результаты в свои файлы, а родительский процесс управляет их работой и обеспечивает распределение задач.

Код программы

parent.c

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

```c
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <time.h>

#define SHM_NAME "/shm_example"
#define SEM_CHILD1 "/sem_child1"
#define SEM_CHILD2 "/sem_child2"

#define SHM_SIZE 1024

void HandleError(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, "\n", 1);
    exit(1);
}

void Print(const char *msg) {
    write(STDOUT_FILENO, msg, strlen(msg));
}

ssize_t Getline(char **lineptr, size_t *n, int fd) {
    if (*lineptr == NULL) {
        *lineptr = malloc(128);
        *n = 128;
    }

    size_t pos = 0;
    char c;
    while (read(fd, &c, 1) == 1) {
        if (pos >= *n - 1) {
            *n *= 2;
            *lineptr = realloc(*lineptr, *n);
        }
        (*lineptr)[pos++] = c;
        if (c == '\n') {
            break;
        }
    }

    if (pos == 0) {
        return -1;
    }

    (*lineptr)[pos] = '\0';
    return pos;
}

int main() {
    char *file1 = NULL, *file2 = NULL;
    size_t file_len = 0;
    char *input = NULL;
    size_t len = 0;
    ssize_t nread;

    Print("Введите имя файла для дочернего процесса 1: ");
    Getline(&file1, &file_len, STDIN_FILENO);
    file1[strcspn(file1, "\n")] = 0;

    Print("Введите имя файла для дочернего процесса 2: ");
    Getline(&file2, &file_len, STDIN_FILENO);
    file2[strcspn(file2, "\n")] = 0;

    // Создаем shared memory
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0644);
    if (shm_fd == -1) {
```

```
        HandleError("Ошибка создания разделяемой памяти");
    }

    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        HandleError("Ошибка установки размера разделяемой памяти");
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shm_ptr == MAP_FAILED) {
        HandleError("Ошибка отображения разделяемой памяти");
    }

    // Создаем семафоры
    sem_t *sem_child1 = sem_open(SEM_CHILD1, O_CREAT, 0644, 0);
    sem_t *sem_child2 = sem_open(SEM_CHILD2, O_CREAT, 0644, 0);
    if (sem_child1 == SEM_FAILED || sem_child2 == SEM_FAILED) {
        HandleError("Ошибка создания семафоров");
    }

    pid_t child1, child2;

    if ((child1 = fork()) == 0) {
        execlp("./child1", "./child1", file1, NULL);
        HandleError("Ошибка запуска дочернего процесса 1");
    }

    if ((child2 = fork()) == 0) {
        execlp("./child2", "./child2", file2, NULL);
        HandleError("Ошибка запуска дочернего процесса 2");
    }

    srand(time(NULL));

    while (1) {
        Print("Введите строку (или 'exit' для завершения): ");
        nread = Getline(&input, &len, STDIN_FILENO);
        input[strcspn(input, "\n")] = 0;

        if (strcmp(input, "exit") == 0) {
            break;
        }

        // Копируем данные в shared memory
        strncpy(shm_ptr, input, SHM_SIZE - 1);

        int r = rand() % 5 + 1;
        //printf("r = %d\n", r);
        if (r == 3) {
            sem_post(sem_child2);
        } else {
            sem_post(sem_child1);
        }
    }

    Print("Работа завершена.\n");

    // Удаляем ресурсы
    munmap(shm_ptr, SHM_SIZE);
    shm_unlink(SHM_NAME);
    sem_unlink(SEM_CHILD1);
    sem_unlink(SEM_CHILD2);


    free(file1);
    free(file2);
    free(input);
```

```
    return 0;
}
```

Child1.c

```
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <semaphore.h>

#define SHM_NAME "/shm_example"
#define SEM_CHILD1 "/sem_child1"
#define SHM_SIZE 1024

void HandleError(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, "\n", 1);
    exit(1);
}

void RemoveVowels(char *str) {
    char *p = str, *q = str;
    while (*p) {
        if (!strchr("AEIOUaeiou", *p)) {
            *q++ = *p;
        }
        p++;
    }
    *q = '\0';
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        HandleError("Usage: <program> <output_file>");
    }

    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        HandleError("Cannot open file");
    }

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0644);
    if (shm_fd == -1) {
        HandleError("Ошибка доступа к разделяемой памяти");
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shm_ptr == MAP_FAILED) {
        HandleError("Ошибка отображения разделяемой памяти");
    }

    sem_t *sem_child1 = sem_open(SEM_CHILD1, 0);
    if (sem_child1 == SEM_FAILED) {
        HandleError("Ошибка доступа к семафору");
    }

    while (1) {
        sem_wait(sem_child1);

        char buffer[SHM_SIZE];
        strncpy(buffer, shm_ptr, SHM_SIZE - 1);
```

```
        buffer[SHM_SIZE - 1] = '\0';

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

    munmap(shm_ptr, SHM_SIZE);
    close(fd);

    return 0;
}
```

Child2.c

```
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <semaphore.h>

#define SHM_NAME "/shm_example"
#define SEM_CHILD1 "/sem_child1"
#define SHM_SIZE 1024

void HandleError(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, "\n", 1);
    exit(1);
}

void RemoveVowels(char *str) {
    char *p = str, *q = str;
    while (*p) {
        if (!strchr("AEIOUaeiou", *p)) {
            *q++ = *p;
        }
        p++;
    }
    *q = '\0';
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        HandleError("Usage: <program> <output_file>");
    }

    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        HandleError("Cannot open file");
    }

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0644);
    if (shm_fd == -1) {
        HandleError("Ошибка доступа к разделяемой памяти");
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shm_ptr == MAP_FAILED) {
        HandleError("Ошибка отображения разделяемой памяти");
    }

    sem_t *sem_child1 = sem_open(SEM_CHILD1, 0);
    if (sem_child1 == SEM_FAILED) {
        HandleError("Ошибка доступа к семафору");
    }
```

```
    while (1) {
        sem_wait(sem_child1);

        char buffer[SHM_SIZE];
        strncpy(buffer, shm_ptr, SHM_SIZE - 1);
        buffer[SHM_SIZE - 1] = '\0';

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

    munmap(shm_ptr, SHM_SIZE);
    close(fd);

    return 0;
}
```

Протокол работы программы

Тестирование:

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab3$ ./parent

Введите имя файла для дочернего процесса 1: f1.txt

Введите имя файла для дочернего процесса 2: f2.txt

Введите строку (или 'exit' для завершения): kshasb

Введите строку (или 'exit' для завершения): im

Введите строку (или 'exit' для завершения): testing

Введите строку (или 'exit' для завершения): ^C

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab3$ cat f1.txt

kshsb

tstng

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab3$ cat f2.txt

m

Strace:

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab3$ strace ./parent

execve("./parent", ["./parent"], 0x7fffc90ec050 /* 20 vars */) = 0

brk(NULL) = 0x7fffeb36e000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffff22b8310) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc3f1a80000

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=16055, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 16055, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc3f1a8c000

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\0\226 \25\252\235\23<l\274\3731\3540\5\226\327"..., 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) = 0x7fc3f1850000

mprotect(0x7fc3f1878000, 2023424, PROT_NONE) = 0

mmap(0x7fc3f1878000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fc3f1878000

mmap(0x7fc3f1a0d000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7fc3f1a0d000

mmap(0x7fc3f1a66000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7fc3f1a66000

mmap(0x7fc3f1a6c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fc3f1a6c000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc3f1840000

arch_prctl(ARCH_SET_FS, 0x7fc3f1840740) = 0

set_tid_address(0x7fc3f1840a10) = 1132

set_robust_list(0x7fc3f1840a20, 24) = 0

rseq(0x7fc3f18410e0, 0x20, 0, 0x53053053) = -1 ENOSYS (Function not implemented)

mprotect(0x7fc3f1a66000, 16384, PROT_READ) = 0

mprotect(0x7fc3f1acf000, 4096, PROT_READ) = 0

mprotect(0x7fc3f1ac8000, 8192, PROT_READ) = 0

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

munmap(0x7fc3f1a8c000, 16055) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 79Введите имя файла для дочернего процесса 1:) = 79

getrandom("\xdc\xb8\x49\xe3\x3a\x7b\xc8\x24", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x7fffeb36e000

brk(0x7fffeb38f000) = 0x7fffeb38f000

read(0, f1.txt

"f", 1) = 1

read(0, "1", 1) = 1

read(0, ".", 1) = 1

read(0, "t", 1) = 1

read(0, "x", 1) = 1

read(0, "t", 1) = 1

read(0, "\n", 1) = 1

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 79Введите имя файла для дочернего процесса 2:) = 79

read(0, f2.txt

"f", 1) = 1

read(0, "2", 1) = 1

read(0, ".", 1) = 1

read(0, "t", 1) = 1

read(0, "x", 1) = 1

read(0, "t", 1) = 1

read(0, "\n", 1) = 1

openat(AT_FDCWD, "/dev/shm/shm_example", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0644) = 3

ftruncate(3, 1024) = 0

mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fc3f1ac7000

openat(AT_FDCWD, "/dev/shm/sem.sem_child1", O_RDWR|O_NOFOLLOW) = 4

newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fc3f1a8f000

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_child2", O_RDWR|O_NOFOLLOW) = 4

newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fc3f1a8e000

close(4) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fc3f1840a10) = 1133

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fc3f1840a10) = 1134

time(NULL) = 1735392627 (2024-12-28T16:30:27+0300)

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\321\203 (\320\270\320"..., 73Введите строку (или 'exit' для завершения):) = 73

read(0, im

"i", 1) = 1

read(0, "m", 1) = 1

read(0, "\n", 1) = 1

futex(0x7fc3f1a8f000, FUTEX_WAKE, 1) = 1

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\321\203 (\320\270\320"..., 73Введите строку (или 'exit' для завершения):) = 73

read(0, testing

"t", 1) = 1

read(0, "e", 1) = 1

read(0, "s", 1) = 1

read(0, "t", 1) = 1

read(0, "i", 1) = 1

read(0, "n", 1) = 1

read(0, "g", 1) = 1

read(0, "\n", 1) = 1

futex(0x7fc3f1a8f000, FUTEX_WAKE, 1) = 1

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\321\203 (\320\270\320"..., 73Введите строку (или 'exit' для завершения):) = 73

read(0, ^C0x7ffff22b832f, 1) = ? ERESTARTSYS (To be restarted if SA_RESTART is set)

strace: Process 1132 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, 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.