Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Пономарев А.А

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 29.11.24

Москва, 2024

Постановка задачи

Вариант 13.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «_».

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создание дочернего процесса
- pid_t wait(int) - ожидание завершения дочерних процессов
- key_t ftok (const char *, int) – создание ключа System V IPC
- int shmget(key_t, size_t, int) – получение дескриптора (создание) разделяемого
- сегмента памяти
- void *shmat(int, const void*, int) – внесение разделяемого сегмента памяти в
- пространство имен процесса
- int shmdt(const void*) - удаленеи разделяемого сегмента памяти из
- пространства имен процесса
- int shmctl(int, int, struct shmid_ds *) - удаление разделяемого сегмента памяти

В рамках лабораторной работы я написал программу, которая использует механизмы межпроцессного взаимодействия через общую память в операционной системе Linux. Программа состоит из трех частей: родительского процесса и двух дочерних процессов. Задача заключалась в том, чтобы продемонстрировать взаимодействие между процессами, обработку строковых данных в общей памяти и использование системных вызовов для создания и управления общей памятью.

Структура программы:

1. **Родительский процесс (parent.c)**:
   - Создает файл для использования в качестве ключа для общедоступной памяти.
   - С помощью `ftok` генерирует ключ для дальнейшего взаимодействия с общей памятью.
   - Создает сегмент общей памяти с использованием `shmget` и получает указатель на эту память через `shmat`.
   - Читает строку с ввода пользователя, записывает её в общую память и затем запускает два дочерних процесса.
   - Ожидает завершения обоих дочерних процессов и выводит результат, который был изменен в общей памяти.
2. **Дочерний процесс 1 (child1.c)**:
   - Этот процесс читает данные из общей памяти, преобразует все символы строки в нижний регистр с помощью функции `tolower` и затем записывает измененную строку обратно в общую память.

3. **Дочерний процесс 2 (child2.c)**:
    - Этот процесс читает данные из общей памяти, заменяет все пробелы на символы подчеркивания (_) и снова записывает измененную строку обратно в общую память.

Принцип работы программы:

- Родительский процесс запускает цикл, в котором он запрашивает у пользователя ввод строки.
- После ввода строки она записывается в общую память, и родительский процесс запускает два дочерних процесса.
- Первый дочерний процесс изменяет строку, преобразуя все символы в нижний регистр.
- Второй дочерний процесс заменяет все пробелы на подчеркивания.
- После выполнения обоих дочерних процессов родительский процесс выводит результат из общей памяти и снова запрашивает ввод от пользователя.
- Если введена пустая строка, программа завершает свою работу.

Примечания:

- Каждый дочерний процесс использует один и тот же сегмент общей памяти, чтобы изменить строку. Это достигается с помощью системных вызовов `shmget`, `shmat`, и `shmdt`.
- Для синхронизации между процессами используется `wait`, чтобы родительский процесс ждал завершения каждого дочернего процесса.
- Если возникнут ошибки в любой части работы с памятью или процессами, программа выведет сообщение об ошибке и завершится с кодом ошибки.

Пример работы программы:

```
Введите строку (или пустую строку для выхода): Привет мир
Результат обработки: привет_мир
Введите строку (или пустую строку для выхода): Лабораторная работа
Результат обработки: лабораторная_работа
Введите строку (или пустую строку для выхода):
```

В результате выполнения программы, каждый вводимый текст преобразуется сначала в нижний регистр, а затем пробелы заменяются на подчеркивания.

Код программы

parent.c

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

#define SHM_SIZE 1024

void handle_error(const char * msg) {
    const char * error_message = ": Ошибка\n";
    write(2, msg, strlen(msg));
    write(2, error_message, strlen(error_message));
```

```
    _exit(EXIT_FAILURE);
}

void write_message(const char * msg) {
    write(1, msg, strlen(msg));
}

void read_message(char * buffer, size_t size) {
    ssize_t bytes_read = read(0, buffer, size - 1);
    if (bytes_read <= 0) handle_error("Ошибка чтения");
    buffer[bytes_read - 1] = '\0';
}

int main() {
    int fd = open("shared_memory", O_CREAT | O_RDWR, 0666);
    if (fd == -1) handle_error("Ошибка создания файла");
    close(fd);

    key_t key = ftok("shared_memory", 65);
    if (key == -1) handle_error("ftok");

    int shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) handle_error("shmget");

    char * shared_memory = (char * ) shmat(shmid, NULL, 0);
    if (shared_memory == (char * ) - 1) handle_error("shmat");

    write_message("Введите строку (или пустую строку для выхода): ");
    char input_buffer[SHM_SIZE];

    while (1) {
        read_message(input_buffer, SHM_SIZE);

        if (strcmp(input_buffer, "") == 0) break;

        strcpy(shared_memory, input_buffer);

        pid_t pid1 = fork();
        if (pid1 == -1) handle_error("fork");

        if (pid1 == 0) {
            execl("./child1", "./child1", NULL);
            handle_error("execl (child1)");
        }

        wait(NULL);

        pid_t pid2 = fork();
        if (pid2 == -1) handle_error("fork");

        if (pid2 == 0) {
            execl("./child2", "./child2", NULL);
            handle_error("execl (child2)");
        }

        wait(NULL);

        write_message("Результат обработки: ");
        write_message(shared_memory);
        write_message("\nВведите строку (или пустую строку для выхода): ");
    }

    if (shmdt(shared_memory) == -1) handle_error("shmdt");
    if (shmctl(shmid, IPC_RMID, NULL) == -1) handle_error("shmctl");

    return 0;
}
```

child1.c

```c
#include <unistd.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <errno.h>
#include <string.h>

#define SHM_SIZE 1024

void handle_error(const char *msg) {
    write(2, msg, strlen(msg));
    _exit(EXIT_FAILURE);
}

int main() {
    key_t key = ftok("shared_memory", 65);
    if (key == -1) handle_error("ftok");

    int shmid = shmget(key, SHM_SIZE, 0666);
    if (shmid == -1) handle_error("shmget");

    char *shared_memory = (char *)shmat(shmid, NULL, 0);
    if (shared_memory == (char *)-1) handle_error("shmat");

    for (int i = 0; shared_memory[i] != '\0'; i++)
        shared_memory[i] = tolower(shared_memory[i]);

    if (shmdt(shared_memory) == -1) handle_error("shmdt");

    return 0;
}
```

child2.c

```c
#include <unistd.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <errno.h>
#include <string.h>

#define SHM_SIZE 1024

void handle_error(const char *msg) {
    write(2, msg, strlen(msg));
    _exit(EXIT_FAILURE);
}

int main() {
    key_t key = ftok("shared_memory", 65);
    if (key == -1) handle_error("ftok");

    int shmid = shmget(key, SHM_SIZE, 0666);
    if (shmid == -1) handle_error("shmget");

    char *shared_memory = (char *)shmat(shmid, NULL, 0);
    if (shared_memory == (char *)-1) handle_error("shmat");

    for (int i = 0; shared_memory[i] != '\0'; i++) {
        if (shared_memory[i] == ' ')
            shared_memory[i] = '_';
    }

    if (shmdt(shared_memory) == -1) handle_error("shmdt");

    return 0;
}
```

Протокол работы программы

Тестирование:

./final

Введите строку (или пустую строку для выхода): Hello Why

Результат обработки: hello___why

Введите строку (или пустую строку для выхода): I am MARK

Результат обработки: i_am_mark

Введите строку (или пустую строку для выхода): Nope

Результат обработки: nope

Введите строку (или пустую строку для выхода):

Strace

execve("./final", ["./final"], 0x7ffe702942d0 /* 49 vars */) = 0

brk(NULL)   = 0x561561ba1000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc7e8ff5e0) = -1 EINVAL (Недопустимый аргумент)

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=70284, ...}) = 0

mmap(NULL, 70284, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fac632e2000

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\300A\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\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fac632e0000

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\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fac630ee000

mmap(0x7fac63110000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fac63110000

mmap(0x7fac63288000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7fac63288000

mmap(0x7fac632d6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fac632d6000

mmap(0x7fac632dc000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fac632dc000

close(3) = 0

arch_prctl(ARCH_SET_FS, 0x7fac632e1540) = 0

mprotect(0x7fac632d6000, 16384, PROT_READ) = 0

mprotect(0x56156172e000, 4096, PROT_READ) = 0

mprotect(0x7fac63321000, 4096, PROT_READ) = 0

munmap(0x7fac632e2000, 70284) = 0

openat(AT_FDCWD, "shared_memory", O_RDWR|O_CREAT, 0666) = 3

close(3) = 0

stat("shared_memory", {st_mode=S_IFREG|0664, st_size=0, ...}) = 0

shmget(0x41053223, 1024, IPC_CREAT|0666) = 32824

shmat(32824, NULL, 0) = 0x7fac63320000

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"..., 84Введите строку (или пустую строку для выхода):) = 84

read(0, Hello Why

"Hello Why\n", 1023) = 10

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fac632e1810) = 18945

wait4(-1, NULL, 0, NULL) = 18945

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=18945, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fac632e1810) = 18946

wait4(-1, NULL, 0, NULL) = 18946

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=18946, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202\320\276\320\261\321\200\320\260\320\261\320\276\321"..., 39Результат обработки:) = 39

write(1, "hello_why", 9hello_why) = 9

write(1, "\n\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"..., 85

Введите строку (или пустую строку для выхода):) = 85

read(0, I am MARK

"I am MARK\n", 1023) = 10

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fac632e1810) = 18947

wait4(-1, NULL, 0, NULL) = 18947

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=18947, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fac632e1810) = 18948

wait4(-1, NULL, 0, NULL) = 18948

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=18948, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202\320\276\320\261\321\200\320\260\320\261\320\276\321"..., 39Результат обработки:) = 39

write(1, "i_am_mark", 9i_am_mark) = 9

write(1, "\n\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"..., 85

Введите строку (или пустую строку для выхода):) = 85

read(0, Nope

"Nope\n", 1023) = 5

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fac632e1810) = 18949

wait4(-1, NULL, 0, NULL) = 18949

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=18949, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fac632e1810) = 18950

wait4(-1, NULL, 0, NULL) = 18950

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=18950, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202\320\276\320\261\321\200\320\260\320\261\320\276\321"..., 39Результат обработки:) = 39

write(1, "nope", 4nope) = 4

write(1, "\n\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"..., 85

Введите строку (или пустую строку для выхода):) = 85

read(0,

"\n", 1023) = 1

shmdt(0x7fac63320000) = 0

shmctl(32824, IPC_RMID, NULL) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе лабораторной работы была реализована программа, использующая общую память для межпроцессного взаимодействия. Родительский процесс записывает строку в общую память, а два дочерних процесса последовательно изменяют её, преобразуя символы в нижний регистр и заменяя пробелы на подчеркивания. Работа с системными вызовами, такими как `shmget`, `shmat`, и `shmdt`, а также синхронизация процессов через `wait`, позволили продемонстрировать основы взаимодействия между процессами. Лабораторная работа улучшила понимание механизмов работы с памятью и процессами в Linux.