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-211Б-23

Студент: Бачурин Н.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 09.01.24

Москва, 2024

Постановка задачи

Вариант 8.

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством shared memory и memory mapping

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через shared memory. Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы. Для синхронизации чтения и записи из shared memory используется семафор.

В файле записаны команды вида: «число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- pid_t getpid(void); – возвращает ID вызывающего процесса.
- int open(const char *__file, int __oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
- ssize_t write(int __fd, const void *__buf, size_t __n); – Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int close(int __fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
- int execv(const char *__path, char *const *__argv); – заменяет образ текущего процесса на образ нового процесса, определённого в пути path.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – считывает указанное количество байт из файла(FD) в буфер(BUF).
- pid_t wait(int *__stat_loc); – используются для ожидания изменения состояния процесса-потомка вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.
- int shm_open(const char *name, int oflag, mode_t mode); – создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX.
- int shm_unlink(const char *name); – удаляется имя объекта разделяемой памяти и, как только все процессы завершили работу с объектом и отменили его распределение, очищают пространство и уничтожают связанную с ним область памяти.

- int ftruncate(int fd, off_t length); – устанавливают длину файла с файловым дескриптором fd в length байт.
- void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.
- int munmap(void *start, size_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти".
- sem_t *sem_open(const char *name, int oflag); ИЛИ sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value); – создаёт новый семафор или открывает уже существующий.
- int sem_wait(sem_t *sem); – уменьшает значение семафора на 1. Если семафор в данный момент имеет нулевое значение, то вызов блокируется до тех пор, пока либо не станет возможным выполнить уменьшение.
- int sem_post(sem_t *sem); – увеличивает значение семафора на 1.
- int sem_unlink(const char *name); – удаляет имя семафора из системы. После вызова этой функции другие процессы больше не смогут открыть этот семафор по имени.
- int sem_close(sem_t *sem); – закрывает указанный семафор, освобождая ресурсы, связанные с ним.

Работа программы:

Родительский процесс (parent.c):

Инициализация ресурсов:

- Создает разделяемую память с помощью shm_open и задает ее размер с помощью ftruncate.
- Отображает разделяемую память в адресное пространство процесса с помощью mmap.
- Создает именованный семафор с помощью sem_open, который будет использоваться для синхронизации между процессами.

Получение имени файла:

- Запрашивает у пользователя имя файла для обработки.
- Читает имя файла из стандартного ввода и записывает его в разделяемую память.
- Создание дочернего процесса:
- С помощью fork создает дочерний процесс.
- Если процесс успешно создан, родитель вызывает sem_post для разблокировки семафора, сигнализируя дочернему процессу, что можно начинать работу.

Ожидание завершения:

- Родительский процесс ожидает завершения дочернего процесса с помощью wait.
- После завершения работы дочернего процесса читает результаты обработки из разделяемой памяти (построчно) и выводит их в стандартный вывод.

Очистка ресурсов:

- Освобождает разделяемую память (munmap, shm_unlink) и закрывает семафор (sem_close, sem_unlink).

Дочерний процесс (child.c):

Подключение к разделяемой памяти и семафору:

- Подключается к существующей разделяемой памяти с помощью shm_open и mmap.
- Подключается к существующему семафору с помощью sem_open.

Ожидание сигнала:

- Ожидает разблокировки семафора (sem_wait), чтобы начать обработку.

Чтение имени файла:

- Считывает имя файла из разделяемой памяти и пытается открыть файл для чтения.
- Если файл не удается открыть, записывает сообщение об ошибке в разделяемую память и завершает выполнение.

Обработка файла:

- Читает содержимое файла блоками. Предполагается, что каждая строка файла содержит одно или несколько целых чисел, разделенных пробелами или табуляцией.

Для каждой строки:

- Первый прочитанный номер используется как делимое.
- Все последующие номера делятся на первый. Если встречается деление на ноль, программа записывает сообщение об ошибке в разделяемую память и завершает выполнение.
- Результаты вычислений формируются в строку, которая записывается в соответствующую строку разделяемой памяти.

Завершение:

- Закрывает файл.
- Освобождает семафор (sem_post) для уведомления родительского процесса о завершении обработки.
- Завершает выполнение.

Код программы

parent.c

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

#define SHM_NAME "/shared_memory"
#define SEM_NAME "/sync_semaphore"
#define BUFFER_SIZE 1024
#define NUM_LINES 100

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

int main() {
    int shm_fd;
    char *shared_mem;
    sem_t *semaphore;
    ssize_t bytesRead;

    shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        error_handler("ERROR: broken creation of shared memory");
    }

    if (ftruncate(shm_fd, BUFFER_SIZE * NUM_LINES) == -1) {
        error_handler("ERROR: broken size changing for shared memory");
    }

    shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        error_handler("ERROR: broken displaying for shared memory");
    }

    semaphore = sem_open(SEM_NAME, O_CREAT, 0666, 0);
    if (semaphore == SEM_FAILED) {
        error_handler("ERROR: broken semaphore creation");
    }

    char filename[BUFFER_SIZE];
    const char *prompt = "Put file name : ";
    write(STDOUT_FILENO, prompt, strlen(prompt));
    bytesRead = read(STDIN_FILENO, filename, sizeof(filename));
```

```
    if (bytesRead <= 0) {
        error_handler("ERROR: error of reading filename");
    }

    size_t len = strlen(filename);
    if (len > 0 && filename[len - 1] == '\n') {
        filename[len - 1] = '\0';
    }

    strncpy(shared_mem, filename, BUFFER_SIZE);

    pid_t child_pid = fork();
    if (child_pid == -1) {
        error_handler("ERROR: error of creating child process");
    }

    if (child_pid == 0) {
        execl("./child", "./child", NULL);
        error_handler("ERROR: error in child process");
    } else {
        sem_post(semaphore);

        wait(NULL);

        for (int i = 0; i < NUM_LINES; i++) {
            if (strlen(shared_mem + i * BUFFER_SIZE) > 0) {
                write(
                        STDOUT_FILENO, shared_mem + i * BUFFER_SIZE, strlen(
                                shared_mem + i * BUFFER_SIZE
                                )
                        );
            }
        }

        munmap(shared_mem, BUFFER_SIZE * NUM_LINES);
        shm_unlink(SHM_NAME);
        sem_close(semaphore);
        sem_unlink(SEM_NAME);

        exit(EXIT_SUCCESS);
    }
}
```

child.c

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

```c
#define SHM_NAME "/shared_memory"
#define SEM_NAME "/sync_semaphore"
#define BUFFER_SIZE 1024
#define NUM_LINES 100

void HandleError(const char *message) {
    write(STDERR_FILENO, message, strlen(message));
    exit(EXIT_FAILURE);
}

int my_strtol(const char *str, char **endptr, char *error_flag) {
    long value = strtol(str, endptr, 10);
    if (value > INT_MAX || value < INT_MIN) {
        *error_flag = 1;
    }
    return (int)value;
}

int main() {
    int shm_fd;
    char *shared_mem;
    sem_t *semaphore;

    shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        HandleError("ERROR: wrong connection to shared memory.\n");
    }

    shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        HandleError("ERROR: wrong showing shared memory\n");
    }

    semaphore = sem_open(SEM_NAME, 0);
    if (semaphore == SEM_FAILED) {
        HandleError("ERROR: broken connection to semaphore\n");
    }

    sem_wait(semaphore);

    char filename[BUFFER_SIZE];
    strncpy(filename, shared_mem, BUFFER_SIZE);

    int file = open(filename, O_RDONLY);
    if (file == -1) {
        strncpy(shared_mem, "ERROR: broken file opening\n", BUFFER_SIZE);
        sem_post(semaphore);
        exit(EXIT_FAILURE);
    }

    char buffer[BUFFER_SIZE];
    ssize_t bytesRead;
    int first_number, next_number;
    char *current;
```

```
    int line_number = 0;

    while ((bytesRead = read(file, buffer, BUFFER_SIZE)) > 0) {
        current = buffer;

        while (current < buffer + bytesRead) {
            while (*current == ' ' || *current == '\t') current++;
            if (*current == '\n') {
                current++;
                continue;
            }

            char *endptr;
            char error_flag = 0;
            first_number = my_strtol(current, &endptr, &error_flag);
            current = endptr;

            char result[BUFFER_SIZE];
            int result_len;
            if (error_flag) {
                result_len = snprintf(result, BUFFER_SIZE, "Division result: overflow
error");
            } else {
                result_len = snprintf(result, BUFFER_SIZE, "Division result: %d",
first_number);
            }

            while (current < buffer + bytesRead && *current != '\n') {
                while (*current == ' ' || *current == '\t') current++;
                if (*current == '\n') break;

                next_number = my_strtol(current, &endptr, &error_flag);
                if (error_flag) {
                    result_len += snprintf(result + result_len, BUFFER_SIZE - result_len, ",
overflow error");
                } else if (next_number == 0) {
                    result_len += snprintf(result + result_len, BUFFER_SIZE - result_len, ",
null division error");
                } else {
                    result_len += snprintf(result + result_len, BUFFER_SIZE - result_len, ",
%d / %d = %d", first_number, next_number, first_number / next_number);
                }
                current = endptr;
            }

            result[result_len++] = '\n';
            strncpy(shared_mem + line_number * BUFFER_SIZE, result, result_len);
            line_number++;

            if (line_number >= NUM_LINES) break;
        }
    }

    if (bytesRead == -1) {
        strncpy(shared_mem, "ERROR: broken reading for file\n", BUFFER_SIZE);
```

```
        sem_post(semaphore);
        exit(EXIT_FAILURE);
    }

    close(file);
    sem_post(semaphore);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Тестирование:

lab3 > file1

```
2 1 1
5 2 1
4 2 1
49 7 1
52 52 1
52 3 4
-52 52 52
2 0 0
-8 2 2
```

```
lausniko@DESKTOP-MATHSNO:~/os/lab3$ make run
Running parent program...
./parent file1
Put file name : file1
Division result: 2, 2 / 1 = 2, 2 / 1 = 2
Division result: 5, 5 / 2 = 2, 5 / 1 = 5
Division result: 4, 4 / 2 = 2, 4 / 1 = 4
Division result: 49, 49 / 7 = 7, 49 / 1 = 49
Division result: 52, 52 / 52 = 1, 52 / 1 = 52
Division result: 52, 52 / 3 = 17, 52 / 4 = 13
Division result: -52, -52 / 52 = -1, -52 / 52 = -1
ERROR: null division
lausniko@DESKTOP-MATHSNO:~/os/lab3$
```

Strace:

```
lausniko@DESKTOP-MATHSNO:~/os/lab3$ make run
gcc -Wall -Wextra  -o parent parent.c -lm
gcc -Wall -Wextra  -o child child.c -lm
Running parent program...
strace ./parent file1
execve("./parent", ["./parent", "file1"], 0x7ffdf43d91a8 /* 40 vars */) = 0
brk(NULL)                               = 0x559c40677000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffde6247360) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f079d376000
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=37327, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 37327, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f079d36c000
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\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) = 0x7f079d143000
mprotect(0x7f079d16b000, 2023424, PROT_NONE) = 0
mmap(0x7f079d16b000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7f079d16b000
mmap(0x7f079d300000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7f079d300000
mmap(0x7f079d359000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7f079d359000
mmap(0x7f079d35f000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7f079d35f000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f079d140000
arch_prctl(ARCH_SET_FS, 0x7f079d140740) = 0
set_tid_address(0x7f079d140a10)         = 78053
set_robust_list(0x7f079d140a20, 24)     = 0
rseq(0x7f079d1410e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f079d359000, 16384, PROT_READ) = 0
mprotect(0x559c3c185000, 4096, PROT_READ) = 0
mprotect(0x7f079d3b0000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f079d36c000, 37327)           = 0
openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 102400)                    = 0
mmap(NULL, 102400, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f079d127000
openat(AT_FDCWD, "/dev/shm/sem.sync_semaphore", O_RDWR|O_NOFOLLOW) = -1 ENOENT (No such file or
directory)
getrandom("\xbf\xd2\xc4\x7e\x53\x86\x5d\xd4", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.xAZ8LJ", 0x7ffde6246c60, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.xAZ8LJ", O_RDWR|O_CREAT|O_EXCL, 0666) = 4
write(4, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f079d3af000
link("/dev/shm/sem.xAZ8LJ", "/dev/shm/sem.sync_semaphore") = 0
newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0
getrandom("\xb1\x98\xa3\x4c\x69\x87\x9a\x4a", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x559c40677000
brk(0x559c40698000)                     = 0x559c40698000
unlink("/dev/shm/sem.xAZ8LJ")           = 0
close(4)                                = 0
write(1, "Put file name : ", 16Put file name : )        = 16
read(0, file1
"file1\n", 1024)                = 6
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
```

```
child_tidptr=0x7f079d140a10) = 78103
wait4(-1, NULL, 0, NULL)                = 78103
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=78103, si_uid=1000, si_status=1,
si_utime=0, si_stime=0} ---
write(1, "Division result: 2, 2 / 1 = 2, 2"..., 41Division result: 2, 2 / 1 = 2, 2 / 1 = 2
) = 41
write(1, "Division result: 5, 5 / 2 = 2, 5"..., 41Division result: 5, 5 / 2 = 2, 5 / 1 = 5
) = 41
write(1, "Division result: 4, 4 / 2 = 2, 4"..., 41Division result: 4, 4 / 2 = 2, 4 / 1 = 4
) = 41
write(1, "Division result: 49, 49 / 7 = 7,"..., 45Division result: 49, 49 / 7 = 7, 49 / 1 = 49
) = 45
write(1, "Division result: 52, 52 / 52 = 1"..., 46Division result: 52, 52 / 52 = 1, 52 / 1 = 52
) = 46
write(1, "Division result: 52, 52 / 3 = 17"..., 46Division result: 52, 52 / 3 = 17, 52 / 4 = 13
) = 46
write(1, "Division result: -52, -52 / 52 ="..., 51Division result: -52, -52 / 52 = -1, -52 / 52
= -1
) = 51
write(1, "ERROR: null division \n", 22ERROR: null division
) = 22
munmap(0x7f079d127000, 102400)          = 0
unlink("/dev/shm/shared_memory")        = 0
munmap(0x7f079d3af000, 32)              = 0
unlink("/dev/shm/sem.sync_semaphore")   = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе написания данной лабораторной работы я научился работать с системными вызовами, которые используются для работы с семафорами и shared memory. Я научился передавать данные посредством shared memory и контролировать доступ к ним через семафоры.