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: Фоменко А.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 8.

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.

Код программы

parent.c

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

#define MAX_BUFFER 8192

int main() {
    int fd = shm_open("/my_shm", O_CREAT | O_RDWR, 0666);
    ftruncate(fd, MAX_BUFFER);
    char *shared_memory = mmap(0, MAX_BUFFER, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

    if (shared_memory == MAP_FAILED) {
        perror("mmap failed");
        exit(EXIT_FAILURE);
    }

    sem_t *sem = sem_open("/my_semaphore1", O_CREAT | O_EXCL, 0644, 0);
    if (sem == SEM_FAILED) {
        perror("sem_open failed 2");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid < 0) {
        perror("fork failed");
        return 1;
    }
```

```
    if (pid != 0) {
        sem_wait(sem);

        printf("The result from the child process:\n%s", shared_memory);

        munmap(shared_memory, MAX_BUFFER);
        shm_unlink("/my_shm");
        sem_close(sem);
        sem_destroy(sem);
        sem_unlink("/my_semaphore1");
        wait(NULL);
    } else {
        char filename[256];
        printf("Input file name: ");
        fgets(filename, sizeof(filename), stdin);
        filename[strcspn(filename, "\n")] = 0;

        char *args[] = {"./child", filename, NULL};
        execv(args[0], args);
        perror("execv failed");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

client.c

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

#define MAX_BUFFER 8192

int check_overflow(const char *str_number) {
    if (!str_number) return 1;

    char *endptr;
    errno = 0;
    long result = strtol(str_number, &endptr, 10);

    if (endptr == str_number) return 1;
    if ((result == LONG_MAX || result == LONG_MIN) && errno == ERANGE) return 1;
    if (*endptr != '\0' || result > INT_MAX || result < INT_MIN)  return 1;
```

```
    return 0;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Error: no file name\n");
        return 1;
    }

    int fd = shm_open("/my_shm", O_CREAT | O_RDWR, 0666);
    char *shared_memory = mmap(0, MAX_BUFFER, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

    shared_memory[0] = '\0';
    char buffer[128];
    char string[MAX_BUFFER];
    string[0] = '\0';

    sem_t *sem = sem_open("/my_semaphore1", 0);
    if (sem == SEM_FAILED) {
        perror("sem_open failed 3");
        exit(EXIT_FAILURE);
    }

    const char *filename = argv[1];
    FILE *file = fopen(filename, "r");

    if (file == NULL) {
        perror("Error: file can't be open");
        return 1;
    }

    char line[MAX_BUFFER];
    while (fgets(line, sizeof(line), file)) {
        int num1, num2;
        char *token = strtok(line, " ");

        if (token != NULL) {

            if (check_overflow(token)) {
                    snprintf(buffer, 128, "Error: int overflow in token = %s\n", token);
                    strcat(string, buffer);
            }
            else {
                num1 = atoi(token);

                while ((token = strtok(NULL, " ")) != NULL) {
                    if (check_overflow(token)) {
                        snprintf(buffer, 128, "Error: int overflow in token = %s\n", token);
                        strcat(string, buffer);
                    }
                    else {
                        num2 = atoi(token);

                        if (num2 == 0) {
                            snprintf(buffer, 128, "Error: division by zero, exit();\n");
```

```
                        strcat(string, buffer);
                        strcpy(shared_memory, string);
                        sem_post(sem);

                        munmap(shared_memory, MAX_BUFFER);
                        shm_unlink("/my_semaphore1");
                        sem_close(sem);
                        exit(EXIT_FAILURE);
                    }

                    int result = num1 / num2;
                    snprintf(buffer, 128, "%d / %d = %d\n", num1, num2, result);
                    strcat(string, buffer);

                }

            }
        }
    }

    strcpy(shared_memory, string);
    sem_post(sem);
    munmap(shared_memory, MAX_BUFFER);
    shm_unlink("/my_semaphore1");
    sem_close(sem);

    fclose(file);
    return 0;
}
```

Протокол работы программы

Тестирование:

```
fantastik@LAPTOP-PS2345T8:~/OS/OS/lab3$ ./a.out
Input file name: input.txt
The result from the child process:
8 / 4 = 2
8 / 2 = 4
8 / 1 = 8
8 / 3 = 2
Error: int overflow in token = 60000000000000
8 / 5 = 1
Error: division by zero, exit();
fantastik@LAPTOP-PS2345T8:~/OS/OS/lab3$
```

Strace:

```
fantastik@LAPTOP-PS2345T8:~/OS/OS/lab3$ strace ./a.out
execve("./a.out", ["./a.out"], 0x7ffcf80861e0 /* 37 vars */) = 0
brk(NULL)                               = 0x56119ea16000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd841aea000
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
fstat(3, {st_mode=S_IFREG|0644, st_size=19791, ...}) = 0
mmap(NULL, 19791, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd841ae5000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=14624, ...}) = 0
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd841ae0000
mmap(0x7fd841ae1000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000)
= 0x7fd841ae1000
mmap(0x7fd841ae2000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7fd841ae2000
mmap(0x7fd841ae3000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000)
= 0x7fd841ae3000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=14408, ...}) = 0
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd841adb000
mmap(0x7fd841adc000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000)
= 0x7fd841adc000
mmap(0x7fd841add000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7fd841add000
mmap(0x7fd841ade000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000)
= 0x7fd841ade000
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\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) = 0x7fd8418c9000
mmap(0x7fd8418f1000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7fd8418f1000
mmap(0x7fd841a79000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) =
0x7fd841a79000
mmap(0x7fd841ac8000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1fe000) = 0x7fd841ac8000
mmap(0x7fd841ace000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7fd841ace000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd8418c6000
arch_prctl(ARCH_SET_FS, 0x7fd8418c6740) = 0
set_tid_address(0x7fd8418c6a10)         = 401726
set_robust_list(0x7fd8418c6a20, 24)     = 0
rseq(0x7fd8418c7060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fd841ac8000, 16384, PROT_READ) = 0
mprotect(0x7fd841ade000, 4096, PROT_READ) = 0
mprotect(0x7fd841ae3000, 4096, PROT_READ) = 0
mprotect(0x56119e201000, 4096, PROT_READ) = 0
mprotect(0x7fd841b22000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fd841ae5000, 19791)           = 0
openat(AT_FDCWD, "/dev/shm/my_shm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 8192)                      = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fd841ae8000
getrandom("\x4b\xe7\xb5\x0a\x75\x94\x23\xf5", 8, GRND_NONBLOCK) = 8
getrandom("\x9e\xef\xaf\x14\x14\x9a\x62\x2a", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.817miW", 0x7ffd9c794e70, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
```

```
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.817miW", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0644) = 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) = 0x7fd841ae7000
link("/dev/shm/sem.817miW", "/dev/shm/sem.my_semaphore1") = 0
fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
getrandom("\xcb\xe1\x4f\x67\x66\x3f\x63\xfe", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x56119ea16000
brk(0x56119ea37000)                     = 0x56119ea37000
unlink("/dev/shm/sem.817miW")           = 0
close(4)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fd8418c6a10) = 401727
Input file name: futex(0x7fd841ae7000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANYinput.txt
) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=401727, si_uid=1000, si_status=1,
si_utime=0, si_stime=0} ---
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x7), ...}) = 0
write(1, "The result from the child proces"..., 35The result from the child process:
) = 35
write(1, "8 / 4 = 2\n", 108 / 4 = 2
)             = 10
write(1, "8 / 2 = 4\n", 108 / 2 = 4
)             = 10
write(1, "8 / 1 = 8\n", 108 / 1 = 8
)             = 10
write(1, "8 / 3 = 2\n", 108 / 3 = 2
)             = 10
write(1, "Error: int overflow in token = 6"..., 46Error: int overflow in token = 60000000000000
) = 46
write(1, "8 / 5 = 1\nError: division by zer"..., 438 / 5 = 1
Error: division by zero, exit();
) = 43
munmap(0x7fd841ae8000, 8192)            = 0
unlink("/dev/shm/my_shm")               = 0
munmap(0x7fd841ae7000, 32)              = 0
unlink("/dev/shm/sem.my_semaphore1")    = 0
wait4(-1, NULL, 0, NULL)                = 401727
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе написания данной лабораторной работы я научился работать с новыми системными вызовами в СИ, которые используются для работы с семафорами и shared memory. Научился передавать данные посредством shared memory и контролировать доступ через семафоры. Проблем во время написания лабораторной работы не возникло.