

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-210Б-23

Студент: Григорян А.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 22.02.25

Москва, 2025

Постановка задачи

Вариант 6.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в `pipe1`. Родительский процесс читает из `pipe1` и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип `int`. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void)`; – создает дочерний процесс.
- `ssize_t write(int fd, const void buf[.count], size_t count)`; - пишет `count` байтов из буфера в файл, на который ссылается файловый дескриптор `fd`
- `ssize_t read(int fd, void buf[.count], size_t count)` – пытается прочесть `count` байтов из файлового дескриптора `fd` в буфер `buff`
- `pid_t getpid(void)`; - получить `pid` текущего или родительского процесса
- `int open(const char *pathname, int flags, mode_t mode)`; - открыть файл с указанными флагами или создать, если указаны специальные флаги, возвращает файловый дескриптор
- `int close(int fd)`; - закрывает файловый дескриптор
- `int execv(const char *pathname, char *const argv[])`; - заменяет образ текущего процесса на новый образ, создается новый стек, кучу и сегменты данных
- `pid_t waitpid(pid_t pid, int *stat_loc, int options)`; - ждет пока процесс с `pid` завершится и получается код выхода
- `getpid()`: Возвращает идентификатор текущего процесса.
- `shm_open()`: Открывает или создает объект разделяемой памяти.
- `ftruncate()`: Изменяет размер файла.
- `mmap()`: Отображает файл или устройство в память.
- `munmap()`: Удаляет отображение памяти.
- `sem_open()`: Открывает или создает именованный семафор.
- `sem_close()`: Закрывает именованный семафор.
- `sem_wait()`: Уменьшает значение семафора, блокируя, если значение равно нулю.
- `sem_post()`: Увеличивает значение семафора.
- `unlink()`: Удаляет имя файла.
- `waitpid()`: Ожидает завершения дочернего процесса.

В данной лабораторной работе реализуется межпроцессное взаимодействие с использованием разделяемой памяти и семафоров. Программа parent.c выполняет чтение строк из входного текстового файла, передает их в разделяемую память, а программа child.c получает эти данные, выполняет обработку и возвращает результат обратно.

Основной принцип работы заключается в том, что parent.c считывает строки из файла, передает их в разделяемую память и уведомляет child.c о готовности данных. В свою очередь, child.c ожидает данные, обрабатывает их, вычисляя сумму чисел в строке, и записывает результат обратно в разделяемую память. Процессы взаимодействуют между собой с помощью семафоров, что обеспечивает синхронизацию и предотвращает состояние гонки.

Работа parent.c начинается с чтения имени файла из стандартного ввода. Затем программа открывает указанный файл и создает объект разделяемой памяти с именем /shared_memory_lab, а также два семафора: SEM_PARENT и SEM_CHILD, которые управляют доступом к памяти. Далее с помощью вызова fork() создается дочерний процесс, который выполняет программу child.c посредством execl. В основном цикле parent.c считывает содержимое файла, разбивает его на строки, копирует их в разделяемую память и отправляет сигнал child.c о наличии новых данных. Дождавшись обработки строки, parent.c получает результат из разделяемой памяти и выводит его на экран. По завершении работы программа освобождает ресурсы, закрывает файлы, удаляет семафоры и освобождает разделяемую память.

Программа child.c подключается к уже существующему объекту разделяемой памяти и семафорам. В бесконечном цикле она ожидает, когда parent.c отправит новую строку. Получив данные, child.c выполняет их обработку: разбивает строку на числа, вычисляет их сумму и записывает результат обратно в разделяемую память.

После этого программа сигнализирует parent.c о завершении обработки. Если child.c обнаруживает пустую строку в памяти, это означает окончание работы, после чего программа завершает выполнение и освобождает используемые ресурсы.

Код программы

parent.c

```
#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <semaphore.h>

#include <stdlib.h>
```

```
#include <string.h>

#define SHM_NAME "/shared_memory_lab"

#define SEM_PARENT "/sem_parent"

#define SEM_CHILD "/sem_child"

#define SHM_SIZE 1024


int main() {

    char filename[256];

    int len = read(STDIN_FILENO, filename, sizeof(filename));

    if (len <= 0) _exit(1);

    filename[len - 1] = '\0';


    int file_fd = open(filename, O_RDONLY);

    if (file_fd == -1) _exit(1);


    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);

    ftruncate(shm_fd, SHM_SIZE);

    void *shm_ptr = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);


    sem_t *sem_parent = sem_open(SEM_PARENT, O_CREAT, 0666, 0);

    sem_t *sem_child = sem_open(SEM_CHILD, O_CREAT, 0666, 0);


    if (fork() == 0) {

        execl("./child", "child", NULL);

        _exit(1);

    }


    char buffer[SHM_SIZE];

    while (1) {
```

```

int bytes_read = read(file_fd, buffer, sizeof(buffer) - 1);

if (bytes_read <= 0) break;

buffer[bytes_read] = '\0';


char *line = strtok(buffer, "\n");

while (line) {

    memcpy(shm_ptr, line, strlen(line) + 1);

    sem_post(sem_child);

    sem_wait(sem_parent);


    write(STDOUT_FILENO, shm_ptr, strlen((char*)shm_ptr));

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


    line = strtok(NULL, "\n");

}

}


memset(shm_ptr, 0, SHM_SIZE);

sem_post(sem_child);


close(file_fd);

wait(NULL);

munmap(shm_ptr, SHM_SIZE);

shm_unlink(SHM_NAME);

sem_close(sem_parent);

sem_close(sem_child);

sem_unlink(SEM_PARENT);

sem_unlink(SEM_CHILD);

return 0;

}

```

child.c

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

```
#include <sys/mman.h>
```

```
#include <sys/stat.h>
```

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

```
#include <semaphore.h>
```

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

```
#include <string.h>
```

```
#define SHM_NAME "/shared_memory_lab"
```

```
#define SEM_PARENT "/sem_parent"
```

```
#define SEM_CHILD "/sem_child"
```

```
#define SHM_SIZE 1024
```

```
void int_to_str(int num, char *str);
```

```
int main() {
```

```
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
```

```
    void *shm_ptr = mmap(0, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,  
shm_fd, 0);
```

```
    sem_t *sem_parent = sem_open(SEM_PARENT, 0);
```

```
    sem_t *sem_child = sem_open(SEM_CHILD, 0);
```

```
    char buffer[SHM_SIZE];
```

```
while (1) {  
  
    sem_wait(sem_child);  
  
    memcpy(buffer, shm_ptr, SHM_SIZE);  
  
  
  
    if (strlen(buffer) == 0) break;  
  
  
  
  
    write(STDERR_FILENO, "Child received: ", 16);  
    write(STDERR_FILENO, buffer, strlen(buffer));  
    write(STDERR_FILENO, "\n", 1);  
  
  
  
  
    int sum = 0, num;  
    char *token = strtok(buffer, " \n");  
  
  
  
  
    while (token) {  
  
        num = atoi(token);  
  
        sum += num;  
  
  
  
  
        write(STDERR_FILENO, "Parsed number: ", 15);  
        write(STDERR_FILENO, token, strlen(token));  
        write(STDERR_FILENO, "\n", 1);  
  
  
  
  
        token = strtok(NULL, " \n");  
    }  
}
```

```
char result[32] = {0};  
int_to_str(sum, result);
```

```
write(STDERR_FILENO, "Child computed: ", 16);  
write(STDERR_FILENO, result, strlen(result));  
write(STDERR_FILENO, "\n", 1);
```

```
memcpy(shm_ptr, result, strlen(result) + 1);  
sem_post(sem_parent);  
}
```

```
munmap(shm_ptr, SHM_SIZE);  
sem_close(sem_parent);  
sem_close(sem_child);
```

```
return 0;  
}
```

```
void int_to_str(int num, char *str) {  
    int i = 0, is_negative = 0;  
  
    if (num == 0) {  
        str[i++] = '0';
```



```
} else {  
  
    if (num < 0) {  
  
        is_negative = 1;  
  
        num = -num;  
  
    }  
  
  
  
    char temp[32];  
  
    int j = 0;  
  
  
  
    while (num > 0) {  
  
        temp[j++] = (num % 10) + '0';  
  
        num /= 10;  
  
    }  
  
  
  
    if (is_negative) {  
  
        str[i++] = '-';  
  
    }  
  
  
  
    while (j > 0) {  
  
        str[i++] = temp[--j];  
  
    }  
  
}  
  
  
  
str[i] = '\0'; // Обязательно добавляем ноль в конец строки  
  
}
```


Протокол работы программы

```
[arcsenius@ars-nbdewxx9 src]$ strace ./parent
```

```
execve("./parent", [ "./parent" ], 0x7ffd854e1520 /* 66 vars */) = 0
```

```
brk(NULL) = 0x63101ca85000
```

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

```
fstat(3, {st_mode=S_IFREG|0644, st_size=241615, ...}) = 0
```

```
mmap(NULL, 241615, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a975d337000
```

```
close(3) = 0
```

```
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

```
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340_\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
```

```
fstat(3, {st_mode=S_IFREG|0755, st_size=2014520, ...}) = 0
```

```
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a975d335000
```

```
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, 2034616, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a975d144000
```

```
mmap(0x7a975d168000, 1511424, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7a975d168000
```

```
mmap(0x7a975d2d9000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x195000) = 0x7a975d2d9000
```

```
mmap(0x7a975d327000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e3000) = 0x7a975d327000
```

```
mmap(0x7a975d32d000, 31672, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a975d32d000
```

```
close(3) = 0
```

```
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a975d141000
```

```
arch_prctl(ARCH_SET_FS, 0x7a975d141740) = 0
```

```
set_tid_address(0x7a975d141a10) = 15277
```

```
set_robust_list(0x7a975d141a20, 24) = 0
```

```
rseq(0x7a975d142060, 0x20, 0, 0x53053053) = 0
```

```
mprotect(0x7a975d327000, 16384, PROT_READ) = 0
```

```
mprotect(0x6310092d8000, 4096, PROT_READ) = 0
```

```
mprotect(0x7a975d3ac000, 8192, PROT_READ) = 0
```

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

```
munmap(0x7a975d337000, 241615)    = 0
```

```
read(0, input.txt
```

```
"input.txt\n", 256)    = 10
```

```
openat(AT_FDCWD, "input.txt", O_RDONLY) = 3
```

```
openat(AT_FDCWD, "/dev/shm/shared_memory_lab",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
```

```
ftruncate(4, 1024)      = 0
```

```
mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7a975d371000
```

```
openat(AT_FDCWD, "/dev/shm/sem.yn6cXJ",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
```

[illegible]

```
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7a975d370000
```

```
link("/dev/shm/sem.yn6cXJ", "/dev/shm/sem.sem_parent") = 0
```

```
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
```

```
getrandom("\x06\xc6\xbd\xd3\x19\x9a\x29\xf0", 8, GRND_NONBLOCK) = 8
```

```
brk(NULL) = 0x63101ca85000
```

```
brk(0x63101caa6000) = 0x63101caa6000
```

```
unlink("/dev/shm/sem.vn6cXJ")      = 0
```

$$\text{close}(5) = 0$$

```
getrandom("\x5a\x75\xc8\xe4\xd5\xf5\xa7\xd6", 8, GRND_NONBLOCK) = 8
```

```
newfstatat(AT_FDCWD, "/dev/shm/sem.EQe0YO", 0x7ffdf9319700, AT_SYMLINK_NOFOLLOW) =  
-1 ENOENT (Нет такого файла или каталога)
```

```
openat(AT_FDCWD, "/dev/shm/sem.EQe0YO",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
```

[illegible]

```
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7a975d36f000
```

```
link("/dev/shm/sem.EOe0YO", "/dev/shm/sem.sem child") = 0
```

```
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
```

```
unlink("/dev/shm/sem.EQe0YO") = 0
```

```
close(5) = 0
```

```
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,  
child_tidptr=0x7a975d141a10) = 15279
```

```
read(3, "10 20 30\n5 15\n100 200 300\n", 1023) = 26
```

```
futex(0x7a975d370000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,  
FUTEX_BITSET_MATCH_ANYChild received: 10 20 30
```

```
Parsed number: 10
```

```
Parsed number: 20
```

```
Parsed number: 30
```

```
Child computed: 60
```

```
) = 0
```

```
write(1, "60", 260) = 2
```

```
write(1, "\n", 1
```

```
) = 1
```

```
futex(0x7a975d36f000, FUTEX_WAKE, 1Child received: 5 15) = 1
```

```
Parsed number: 5
```

```
Parsed number: 15
```

```
Child computed: 20
```

```
futex(0x7a975d370000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,  
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Ресурс временно недоступен)
```

```
write(1, "20", 220) = 2
```

```
write(1, "\n", 1
```

```
) = 1
```

```
futex(0x7a975d36f000, FUTEX_WAKE, 1Child received: 100 200 300
```

```
Parsed number: 100) = 1
```

```
Parsed number: 200
```

```
Parsed number: 300
```

```
Child computed: 600
```

```
futex(0x7a975d370000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,  
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Ресурс временно недоступен)
```

```
write(1, "600", 3600) = 3
```

```
write(1, "\n", 1
```

```
) = 1
```

```
read(3, "", 1023) = 0
```

```
futex(0x7a975d36f000, FUTEX_WAKE, 1) = 1
```

```
close(3) = 0
```

```
wait4(-1, NULL, 0, NULL) = 15279
```

```
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=15279, si_uid=1001, si_status=0,  
si_utime=0, si_stime=0} ---
```

```
munmap(0x7a975d371000, 1024) = 0
```

```
unlink("/dev/shm/shared_memory_lab") = 0
```

```
munmap(0x7a975d370000, 32) = 0
```

```
munmap(0x7a975d36f000, 32) = 0
```

```
unlink("/dev/shm/sem.sem_parent") = 0
```

```
unlink("/dev/shm/sem.sem_child") = 0
```

```
exit_group(0) = ?
```

```
+++ exited with 0 +++
```

Вывод

В ходе данной работы я научился создавать процессы, налаживать общение между ними с помощью shared memory. Столкнулся с проблемами при синхронизации с помощью семафоров.