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-211Б-23

Студент: Ласточкин М.В.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 10.12.24

Москва, 2024

**Постановка задачи**

**Вариант 17.**

**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 file1 или в file2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.**

**Правило фильтрации: строки длины больше 10 символов отправляются в file1, иначе в file2. Дочерние процессы удаляют все гласные из строк.**

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int pipe(int \*fd); – создает канал и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].
* 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 dup2(int \_\_fd, int \_\_fd2); – копирует FD в FD2, закрыв FD2 если это требуется.
* 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.
* 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 ftruncate(int fd, off\_t length); – устанавливают длину файла с файловым дескриптором fd в length байт.
* 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\_destroy( sem\_t \*sem ); - уничтожает безымянный семафор, расположенный по адресу sem

Для выполнения данной лабораторной работы я изучил указанные выше системные вызовы, а также пример выполнения подобного задания.

Программа parent.c получает на вход два аргумента – два имени файла для двух дочерних процессов. Далее создаются два файла для общей памяти, в которые будут записываться строки. Создаются два семафора для каждого дочернего процесса для синхронизации работы с общей памятью.

Для каждого процесса с помощью fork() создается новый процесс. После успешного создания, родитель запускает child.c, передавая ей параметры: имя файла, в который дочерний процесс будет записывать результат, и название общей памяти и семафоров, с которыми дочерний процесс будет работать.

Родитель считывает строки с консоли, если символов в строки меньше или равно 10, то отправляется в первую область общей памяти – file2, иначе – во вторую - file1.

В child.c получаются данные, открывается файл для записи, создается общая память для обмена строчками и подключаются семафоры. После получения строчки дочерний процесс удаляет из нее все гласны.

После окончания ввода закрывает общую память и семафоры, ждем завершения дочерних процессов с помощью wait()

**Код программы**

**client.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <semaphore.h>

#define SHM\_NAME "/shared\_memory"

#define BUFFER\_SIZE 4096

typedef struct {

char buffer1[BUFFER\_SIZE];

char buffer2[BUFFER\_SIZE];

sem\_t sem\_parent;

sem\_t sem\_child1;

sem\_t sem\_child2;

bool exit\_flag;

} SharedMemory;

void delete\_vowels(char \*str) {

if (!str) return;

const char \*vowels = "AEIOUYaeiouy";

size\_t j = 0;

for (size\_t i = 0; str[i] != '\0'; ++i) {

if (!strchr(vowels, str[i])) {

str[j++] = str[i];

}

}

str[j] = '\0';

}

int main(int argc, char \*\*argv) {

if (argc != 3) {

fprintf(stderr, "Usage: %s <client\_id> <output\_file>\n", argv[0]);

exit(EXIT\_FAILURE);

}

int shm\_fd = shm\_open(SHM\_NAME, O\_RDWR, 0600);

if (shm\_fd == -1) {

perror("shm\_open");

exit(EXIT\_FAILURE);

}

SharedMemory \*shm = mmap(NULL, sizeof(SharedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shm == MAP\_FAILED) {

perror("mmap");

exit(EXIT\_FAILURE);

}

int output\_file = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0600);

if (output\_file == -1) {

perror("open");

exit(EXIT\_FAILURE);

}

sem\_t \*my\_sem;

char \*my\_buffer;

if (strcmp(argv[1], "client1") == 0) {

my\_sem = &shm->sem\_child1;

my\_buffer = shm->buffer1;

} else {

my\_sem = &shm->sem\_child2;

my\_buffer = shm->buffer2;

}

while (true) {

sem\_wait(my\_sem);

if (shm->exit\_flag) break;

delete\_vowels(my\_buffer);

size\_t len = strlen(my\_buffer);

if (write(output\_file, my\_buffer, len) == -1) {

perror("write");

exit(EXIT\_FAILURE);

}

write(output\_file, "\n", 1);

sem\_post(&shm->sem\_parent);

}

close(output\_file);

munmap(shm, sizeof(SharedMemory));

return 0;

}

**server.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <semaphore.h>

#include <sys/wait.h>

#define SHM\_NAME "/shared\_memory"

#define BUFFER\_SIZE 4096

typedef struct {

char buffer1[BUFFER\_SIZE];

char buffer2[BUFFER\_SIZE];

sem\_t sem\_parent;

sem\_t sem\_child1;

sem\_t sem\_child2;

bool exit\_flag;

} SharedMemory;

int main(int argc, char \*\*argv) {

if (argc < 3) {

fprintf(stderr, "Usage: %s file1 file2\n", argv[0]);

exit(EXIT\_FAILURE);

}

int shm\_fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0600);

if (shm\_fd == -1) {

perror("shm\_open");

exit(EXIT\_FAILURE);

}

if (ftruncate(shm\_fd, sizeof(SharedMemory)) == -1) {

perror("ftruncate");

exit(EXIT\_FAILURE);

}

SharedMemory \*shm = mmap(NULL, sizeof(SharedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shm == MAP\_FAILED) {

perror("mmap");

exit(EXIT\_FAILURE);

}

sem\_init(&shm->sem\_parent, 1, 1);

sem\_init(&shm->sem\_child1, 1, 0);

sem\_init(&shm->sem\_child2, 1, 0);

shm->exit\_flag = false;

pid\_t child1 = fork();

if (child1 == 0) {

execlp("./client", "./client", "client1", argv[1], NULL);

perror("execlp");

exit(EXIT\_FAILURE);

}

pid\_t child2 = fork();

if (child2 == 0) {

execlp("./client", "./client", "client2", argv[2], NULL);

perror("execlp");

exit(EXIT\_FAILURE);

}

char input[BUFFER\_SIZE];

while (true) {

sem\_wait(&shm->sem\_parent);

printf("Input strings (press ENTER to exit): ");

if (fgets(input, sizeof(input), stdin) == NULL) {

perror("fgets");

continue;

}

size\_t len = strlen(input);

if (len > 0 && input[len - 1] == '\n') {

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

}

if (input[0] == '\0') {

shm->exit\_flag = true;

sem\_post(&shm->sem\_child1);

sem\_post(&shm->sem\_child2);

break;

}

if (strlen(input) > 10) {

strncpy(shm->buffer1, input, BUFFER\_SIZE - 1);

shm->buffer1[BUFFER\_SIZE - 1] = '\0';

sem\_post(&shm->sem\_child1);

} else {

strncpy(shm->buffer2, input, BUFFER\_SIZE - 1);

shm->buffer2[BUFFER\_SIZE - 1] = '\0';

sem\_post(&shm->sem\_child2);

}

}

wait(NULL);

wait(NULL);

sem\_destroy(&shm->sem\_parent);

sem\_destroy(&shm->sem\_child1);

sem\_destroy(&shm->sem\_child2);

munmap(shm, sizeof(SharedMemory));

shm\_unlink(SHM\_NAME);

return 0;

}

**Протокол работы программы**

max@DESKTOP-L04A0IM:/mnt/c/Users/lasto/CLionProjects/Osi/laba3$ gcc -o server server.c

max@DESKTOP-L04A0IM:/mnt/c/Users/lasto/CLionProjects/Osi/laba3$ gcc -o client client.c

max@DESKTOP-L04A0IM:/mnt/c/Users/lasto/CLionProjects/Osi/laba3$ ./server file1.txt

file2.txt

Input strings (press ENTER to exit): bababajfd

Input strings (press ENTER to exit): ituir423a

Input strings (press ENTER to exit): lerwrgd3

Input strings (press ENTER to exit): kjfskjfkdsjfksfjsd

Input strings (press ENTER to exit):

max@DESKTOP-L04A0IM:/mnt/c/Users/lasto/CLionProjects/Osi/laba3$ strace ./server

file1.txt file2.txt

execve("./server", ["./server", "file1.txt", "file2.txt"], 0x7fff6a000100 /\* 27 vars

\*/) = 0

brk(NULL) = 0x55a266c16000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe1a21ebd0) = -1 EINVAL (Invalid argument)

**mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) =**

**0x7f10ba80b000**

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=18383, ...}, AT\_EMPTY\_PATH) = 0

**mmap(NULL, 18383, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f10ba806000**

**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) = 0x7f10ba5dd000**

**mprotect(0x7f10ba605000, 2023424, PROT\_NONE) = 0**

**mmap(0x7f10ba605000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE,**

**3, 0x28000) = 0x7f10ba605000**

**mmap(0x7f10ba79a000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,**

**0x1bd000) = 0x7f10ba79a000**

**mmap(0x7f10ba7f3000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE,**

**3, 0x215000) = 0x7f10ba7f3000**

**mmap(0x7f10ba7f9000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS,**

**-1, 0) = 0x7f10ba7f9000**

**close(3)**  = 0

**mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) =**

**0x7f10ba5da000**

arch\_prctl(ARCH\_SET\_FS, 0x7f10ba5da740) = 0

set\_tid\_address(0x7f10ba5daa10) = 95199

set\_robust\_list(0x7f10ba5daa20, 24) = 0

rseq(0x7f10ba5db0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f10ba7f3000, 16384, PROT\_READ) = 0

mprotect(0x55a25f8d1000, 4096, PROT\_READ) = 0

mprotect(0x7f10ba845000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

**munmap(0x7f10ba806000, 18383) = 0**

**openat(AT\_FDCWD, "/dev/shm/shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0600) =**

**3**

**ftruncate(3, 8296) = 0**

**mmap(NULL, 8296, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f10ba808000**

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD,

child\_tidptr=0x7f10ba5daa10) = 95200

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD,

child\_tidptr=0x7f10ba5daa10) = 95201

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH)

= 0

getrandom("\xf4\xf5\xbc\x0f\xd0\x94\x40\x76", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55a266c16000

brk(0x55a266c37000) = 0x55a266c37000

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH)

= 0

write(1, "Input strings (press ENTER to ex"..., 37Input strings (press ENTER to exit):

) = 37

read(0, fsdhfjhjrhwejrhj234h32j4h

"fsdhfjhjrhwejrhj234h32j4h\n", 1024) = 26

**futex(0x7f10ba80a020, FUTEX\_WAKE, 1) = 1**

**futex(0x7f10ba80a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL,**

**FUTEX\_BITSET\_MATCH\_ANY) = 0**

write(1, "Input strings (press ENTER to ex"..., 37Input strings (press ENTER to exit):

) = 37

read(0, dhajahjhjhsjhjhfjfhjhj

"dhajahjhjhsjhjhfjfhjhj\n", 1024) = 23

**futex(0x7f10ba80a020, FUTEX\_WAKE, 1) = 1**

**futex(0x7f10ba80a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL,**

**FUTEX\_BITSET\_MATCH\_ANY) = 0**

write(1, "Input strings (press ENTER to ex"..., 37Input strings (press ENTER to exit):

) = 37

read(0, assdasdasdsdasdasdad

"assdasdasdsdasdasdad\n", 1024) = 21

**futex(0x7f10ba80a020, FUTEX\_WAKE, 1) = 1**

**futex(0x7f10ba80a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL,**

**FUTEX\_BITSET\_MATCH\_ANY) = 0**

write(1, "Input strings (press ENTER to ex"..., 37Input strings (press ENTER to exit):

) = 37

read(0, aaaaaaad

"aaaaaaad\n", 1024) = 9

**futex(0x7f10ba80a040, FUTEX\_WAKE, 1) = 1**

**futex(0x7f10ba80a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL,**

**FUTEX\_BITSET\_MATCH\_ANY) = 0**

write(1, "Input strings (press ENTER to ex"..., 37Input strings (press ENTER to exit):

) = 37

read(0, 4i324u34832ujfhjdjfkdjfk4j34

"4i324u34832ujfhjdjfkdjfk4j34\n", 1024) = 29

**futex(0x7f10ba80a020, FUTEX\_WAKE, 1) = 1**

**futex(0x7f10ba80a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL,**

**FUTEX\_BITSET\_MATCH\_ANY) = 0**

write(1, "Input strings (press ENTER to ex"..., 37Input strings (press ENTER to exit):

) = 37

read(0, jrekgjrkgjrkgjrkgjrkgjrhigkrhgkrghrkgjrng

"jrekgjrkgjrkgjrkgjrkgjrhigkrhgkr"..., 1024) = 42

**futex(0x7f10ba80a020, FUTEX\_WAKE, 1) = 1**

**futex(0x7f10ba80a000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL,**

**FUTEX\_BITSET\_MATCH\_ANY) = 0**

write(1, "Input strings (press ENTER to ex"..., 37Input strings (press ENTER to exit):

) = 37

read(0,

"\n", 1024) = 1

**futex(0x7f10ba80a020, FUTEX\_WAKE, 1) = 1**

**futex(0x7f10ba80a040, FUTEX\_WAKE, 1) = 1**

wait4(-1, NULL, 0, NULL) = 95200

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=95200, si\_uid=1000,

si\_status=0, si\_utime=0, si\_stime=0} ---

wait4(-1, NULL, 0, NULL) = 95201

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=95201, si\_uid=1000,

si\_status=0, si\_utime=0, si\_stime=0} ---

**munmap(0x7f10ba808000, 8296) = 0**

**unlink("/dev/shm/shared\_memory") = 0**

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В процессе выполнения данной лабораторной работы я изучил новые системные вызовы на языке Си, которые позволяют эффективно работать с разделяемой памятью и семафорами. Освоил передачу данных между процессами через shared memory и управление доступом с использованием семафоров. Затруднений в ходе выполнения лабораторной работы не возникло, все задачи удалось успешно реализовать.