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: Малеев В.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 17.12.24

Москва, 2024

Постановка задачи

Вариант 19.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. 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 с вероятностью 80% 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 <child_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], "child1") == 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("./child", "./child", "child1", argv[1], NULL);
        perror("execlp");
        exit(EXIT_FAILURE);
```

```
}

pid_t child2 = fork();
if (child2 == 0) {
    execlp("./child", "./child", "child2", 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 (rand() % 100 > 80) {
        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

```
      vladislavmaleev@192 src % sudo dtruss ./parent test1.txt test2.txt
execve("./parent", ["./parent", "file1.txt", "file2.txt"], 0x7fff405a3770 /* 26 vars */)
= 0
brk(NULL)                                       = 0x557d45d0e000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff43815210) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125e8de000
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=25563, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 25563, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f125e8d7000
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\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\0\0"..., 784, 64)
= 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f125e6ae000
mprotect(0x7f125e6d6000, 2023424, PROT_NONE) = 0
mmap(0x7f125e6d6000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f125e6d6000
mmap(0x7f125e86b000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7f125e86b000
mmap(0x7f125e8c4000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7f125e8c4000
mmap(0x7f125e8ca000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f125e8ca000
close(3)                                        = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125e6ab000
arch_prctl(ARCH_SET_FS, 0x7f125e6ab740) = 0
set_tid_address(0x7f125e6aba10)         = 42132
set_robust_list(0x7f125e6aba20, 24)     = 0
rseq(0x7f125e6ac0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f125e8c4000, 16384, PROT_READ) = 0
mprotect(0x557d3d328000, 4096, PROT_READ) = 0
mprotect(0x7f125e918000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f125e8d7000, 25563)           = 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) = 0x7f125e8db000
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f125e6aba10) = 42133
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f125e6aba10) = 42134
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...},
AT_EMPTY_PATH) = 0
getrandom("\xc1\xec\x6d\x34\xc8\xd5\xa6\xa5", 8, GRND_NONBLOCK) = 8
brk(NULL)                                       = 0x557d45d0e000
brk(0x557d45d2f000)                             = 0x557d45d2f000
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...},
AT_EMPTY_PATH) = 0
write(1, "Input strings (press ENTER to ex"..., 37) = 37
read(0, "skarghurghkshdg\n", 1024)      = 16
futex(0x7f125e8dd020, FUTEX_WAKE, 1)    = 1
futex(0x7f125e8dd000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
```

```
FUTEX_BITSET_MATCH_ANY) = 0
write(1, "Input strings (press ENTER to ex"..., 37) = 37
read(0, "sdhsghfhcshs\n", 1024)         = 13
futex(0x7f125e8dd020, FUTEX_WAKE, 1)    = 1
futex(0x7f125e8dd000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Resource temporarily unavailable)
write(1, "Input strings (press ENTER to ex"..., 37) = 37
read(0, "hsdchshsdh\n", 1024)           = 11
futex(0x7f125e8dd040, FUTEX_WAKE, 1)    = 1
futex(0x7f125e8dd000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Resource temporarily unavailable)
write(1, "Input strings (press ENTER to ex"..., 37) = 37
read(0, "scdchfdsfhcdschcdshc\n", 1024) = 21
futex(0x7f125e8dd040, FUTEX_WAKE, 1)    = 1
futex(0x7f125e8dd000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Resource temporarily unavailable)
write(1, "Input strings (press ENTER to ex"..., 37) = 37
read(0, "s\n", 1024)                    = 2
futex(0x7f125e8dd020, FUTEX_WAKE, 1)    = 1
futex(0x7f125e8dd000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Resource temporarily unavailable)
write(1, "Input strings (press ENTER to ex"..., 37) = 37
read(0, "\n", 1024)                     = 1
futex(0x7f125e8dd020, FUTEX_WAKE, 1)    = 1
futex(0x7f125e8dd040, FUTEX_WAKE, 1)    = 1
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=42133, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=42134, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
wait4(-1, NULL, 0, NULL)                = 42133
wait4(-1, NULL, 0, NULL)                = 42134
munmap(0x7f125e8db000, 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.