Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу

«Операционные системы»

Основы взаимодействия с операционными системами

Студент: Кузьмичев Александр Николаевич
Группа: М80 – 306Б-18
Вариант: 15
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020

Постановка задачи

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 решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы, события или через отображаемые файлы

Вариант задания: 15. Родительский процесс считывает стандартный входной поток, отдает его дочернему процессу, который удаляет задвоенные пробелы и выводит его в файл(имя файла также передается от родительского процесса).

Общие сведения о программе

Программа состоит из одного файла **main.c**

В программе используются заголовочные файлы stdlib.h, string.h, stdio.h, sys/mman.h, unistd.h, fcntl.h, semaphore.h, wait.h, sys/stat.h, stdbool.h.

Используются следующие системные вызовы

1. **mkstemp** — создает временный файл.
2. **mmap —** позволяет выполнить отображение файла или устройства в память.
3. **fork —** создает дочерний процесс.
4. **wait** — дожидается завершения дочернего процесса.
5. **sem_open —** инициализирует и открывает именованный семафор.
6. **sem_unlink —** удаляет именованный семафор.
7. **sem_post —** разблокирует семафор, инкрементируя значение, ассоциированное с ним.
8. **sem_wait** — декрементирует значение, ассоциированное с семафором, при этом блокируя его, если это значение равно 0.
9. **sem_close** — закрывает именованный семафор.
10. **read** – для чтения данных из входного потока.
11. **write** – для записи данных в файл или выходной поток.

Общий метод и алгоритм решения

- Произвести проверки корректности входных данных.
- Создать временный файл для последующего маппинга, заполнить его нужным количеством нулевых символов, произвести маппинг.
- Создать два семафора, для синхронизации работы с файлом, отображенным в память.
- Записать в отображенный фал имя файла для вывода, переданное программе в качестве аргумента. Создать дочерний процесс с помощью **fork**.
- Из родительского процесса: Считать все символы, переданные во входной поток и записать их в отображенный файл, для того, чтобы дочерний процесс мог получить к ним доступ. Считывается по 100 символов за раз, а так как **read** возвращает количество считанных символов, то когда символы во входном потоке закончатся, будет нетрудно это понять.
- Из дочернего процесса: считать из отображенного в память файла имя того файла, в который надо записать выходные данные, открыть его с помощью **open**(при неудаче завершиться с кодом выхода 1). Далее, каждый раз, когда в отображенный файл записываются очередные 100 символов, обрабатывать их функцией **parse_string**. Далее полученная строка(без лишних пробелов) записывается в выходной файл с помощью функции **write**. Когда символы, переданные из родительского процесса закончатся, дочерний процесс закроет файл с помощью функции **close** и завершится с кодом выхода 0.
- После передачи символов входного потока в дочерний процесс, родительский процесс ожидает его завершения посредством функции **wait**.

Код программы

main.c:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <wait.h>
#include <sys/stat.h>
#include <stdbool.h>
#define BUFFER_SIZE 100
int parse_string(char buf[], int size, bool* prev) {
  int temp_size = 0;
  char copy[size];
  bool space = *prev;
  for (int i = 0; i < size; ++i) {
    if (buf[i] != ' ' || !space) {
      if (buf[i] == ' ') {
        space = true;
      } else {
        space = false;
      }
      copy[temp_size] = buf[i];
      temp_size++;
    }
  }
  for (int i = 0; i < temp_size; ++i) {
    buf[i] = copy[i];
  }
  *prev = space;
  return temp_size;
}
void throw_error(const char* error) {
  printf("%s\n", error);
  exit(1);
}
int main(int argc, char** argv) {
  if (argc < 2) {
    throw_error("No file");
  }
  if (strlen(argv[1]) > 100) {
    throw_error("Filename is too long");
  }
  //создание временного файла для маппинга
  char* tmp_name = strdup("/tmp/tmp_file.XXXXXX");
  int tmp_fd = mkstemp(tmp_name);
  if (tmp_fd == -1) {
    throw_error("Cannot create temp file to map");
  }
  free(tmp_name);
  int file_size = BUFFER_SIZE + 1;
```

```
  char file_filler[file_size];
  for (int i = 0; i < file_size; ++i) {
    file_filler[i] = '\0';
  }
  write(tmp_fd, file_filler, file_size);
  //маппинг файла
  unsigned char* map = (unsigned char*)mmap(NULL, file_size, PROT_WRITE |
PROT_READ, MAP_SHARED, tmp_fd, 0);
  if (map == NULL) {
    throw_error("Cant map file");
  }
  //создание семафоров для синхронизации работы
  const char* in_sem_name = "/input_semaphor";
  const char* out_sem_name = "/output_semaphor";
  sem_unlink(in_sem_name);
  sem_unlink(out_sem_name);
  sem_t* in_sem = sem_open(in_sem_name, O_CREAT, 777, 0);
  sem_t* out_sem = sem_open(out_sem_name, O_CREAT, 777, 0);
  if (in_sem == SEM_FAILED || out_sem == SEM_FAILED) {
    throw_error("Cannot create semaphor");
  }
  strcpy(map, argv[1]);
  map[BUFFER_SIZE] = strlen(argv[1]);
  int pid = fork();
  if (pid == -1) {
    throw_error("Fork failure");
  } else if (pid == 0) { //child
    int output_file = open(argv[1], O_RDWR | O_TRUNC | O_CREAT, S_IREAD |
S_IWRITE);
    if (output_file == -1) {
      map[BUFFER_SIZE] = 101;
      sem_post(out_sem);
      throw_error("Cannot create output file");
    }
    bool space = false;
    sem_post(out_sem);
    while (true) {
      sem_wait(in_sem);
      int new_size = parse_string(map, map[BUFFER_SIZE], &space);
      write(output_file, map, new_size);
      if (map[BUFFER_SIZE] < BUFFER_SIZE){
        sem_post(out_sem);
        break;
      }
      sem_post(out_sem);
    }
    close(output_file);
    exit(0);
  } else { //parent
    sem_wait(out_sem);
    if (map[BUFFER_SIZE] != 101) {
      int read_count = read(STDIN_FILENO, map, BUFFER_SIZE);
      map[BUFFER_SIZE] = read_count;
      sem_post(in_sem);
      while (read_count == BUFFER_SIZE) {
```

```
        sem_wait(out_sem);
        read_count = read(STDIN_FILENO, map, BUFFER_SIZE);
        map[BUFFER_SIZE] = read_count;
        sem_post(in_sem);
      }
      int stat_lock;
      wait(&stat_lock);
      if (stat_lock != 0) {
        printf("%s\n", "Child failure");
      }
    } else {
      int stat_lock;
      wait(&stat_lock);
      if (stat_lock != 0) {
        printf("%s\n", "Child failure");
      }
    }
    sem_close(in_sem);
    sem_close(out_sem);
  }
}
```

Демонстрация работы программы

```
alex@alex-lenovo:~/CLionProjects/os_lab_04/src/cmake-build-debug$ ls
CMakeCache.txt  cmake_install.cmake  input_file  os_lab_04
CMakeFiles      empty                Makefile    os_lab_04.cbp
alex@alex-lenovo:~/CLionProjects/os_lab_04/src/cmake-build-debug$ cat input_file
Lorem ipsum dolor sit amet consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore
et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore
ue fugiat nulla pariatur.
alex@alex-lenovo:~/CLionProjects/os_lab_04/src/cmake-build-debug$ strace ./os_lab_04 output < input_file
execve("./os_lab_04", ["./os_lab_04", "output"], 0x7ffd38655438 /* 52 vars */) = 0
brk(NULL)                               = 0x5569edacc000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
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=146577, ...}) = 0
mmap(NULL, 146577, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f226b020000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
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\0000b\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=144976, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f226b01e000
```

```
mmap(NULL, 2221184, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f226abfe000

mprotect(0x7f226ac18000, 2093056, PROT_NONE) = 0

mmap(0x7f226ae17000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19000) =
0x7f226ae17000

mmap(0x7f226ae19000, 13440, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f226ae19000

close(3)                        = 0

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)

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\260\34\2\0\0\0\0\0"...,
832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0

mmap(NULL, 4131552, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f226a80d000

mprotect(0x7f226a9f4000, 2097152, PROT_NONE) = 0

mmap(0x7f226abf4000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) =
0x7f226abf4000

mmap(0x7f226abfa000, 15072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f226abfa000

close(3)                        = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f226b01b000

arch_prctl(ARCH_SET_FS, 0x7f226b01b740) = 0

mprotect(0x7f226abf4000, 16384, PROT_READ) = 0

mprotect(0x7f226ae17000, 4096, PROT_READ) = 0

mprotect(0x5569ed6ca000, 4096, PROT_READ) = 0

mprotect(0x7f226b044000, 4096, PROT_READ) = 0

munmap(0x7f226b020000, 146577)          = 0
```

set_tid_address(0x7f226b01ba10)   = 13154

set_robust_list(0x7f226b01ba20, 24)   = 0

rt_sigaction(SIGRTMIN, {sa_handler=0x7f226ac03cb0, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7f226ac10890}, NULL, 8) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f226ac03d50, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f226ac10890}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

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

brk(NULL)   = 0x5569edacc000

brk(0x5569edaed000)   = 0x5569edaed000

getpid()   = 13154

openat(AT_FDCWD, "/tmp/tmp_file.pkOSsu", O_RDWR|O_CREAT|O_EXCL, 0600) = 3

write(3, "\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\0\0\0\0\0\0"..., 101) = 101

mmap(NULL, 101, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f226b043000

statfs("/dev/shm/", {f_type=TMPFS_MAGIC, f_bsize=4096, f_blocks=992841, f_bfree=947907, f_bavail=947907, f_files=992841, f_ffree=992323, f_fsid={val=[0, 0]}, f_namelen=255, f_frsize=4096, f_flags=ST_VALID|ST_NOSUID|ST_NODEV}) = 0

futex(0x7f226ae1c370, FUTEX_WAKE_PRIVATE, 2147483647) = 0

unlink("/dev/shm/sem.input_semaphor")   = 0

unlink("/dev/shm/sem.output_semaphor") = 0

openat(AT_FDCWD, "/dev/shm/sem.input_semaphor", O_RDWR|O_NOFOLLOW) = -1 ENOENT (No such file or directory)

getpid()   = 13154

lstat("/dev/shm/ZDgVKM", 0x7ffc72e11800) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/ZDgVKM", O_RDWR|O_CREAT|O_EXCL, 01411) = 4

write(4, "\0\0\0\0\0\0\0\200\0\0\0\"\177\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) = 0x7f226b042000

link("/dev/shm/ZDgVKM", "/dev/shm/sem.input_semaphor") = 0

fstat(4, {st_mode=S_IFREG|S_ISVTX|0411, st_size=32, ...}) = 0

unlink("/dev/shm/ZDgVKM") = 0

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.output_semaphor", O_RDWR|O_NOFOLLOW) = -1 ENOENT (No such file or directory)

getpid() = 13154

lstat("/dev/shm/nnX024", 0x7ffc72e11800) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/nnX024", O_RDWR|O_CREAT|O_EXCL, 01411) = 4

write(4, "\0\0\0\0\0\0\0\200\0\0\0\"\177\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) = 0x7f226b041000

link("/dev/shm/nnX024", "/dev/shm/sem.output_semaphor") = 0

fstat(4, {st_mode=S_IFREG|S_ISVTX|0411, st_size=32, ...}) = 0

unlink("/dev/shm/nnX024") = 0

close(4) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f226b01ba10) = 13155

futex(0x7f226b041000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, 0xffffffff) = 0

read(0, "Lorem ipsum "..., 100) = 100

futex(0x7f226b042000, FUTEX_WAKE, 1) = 1

read(0, ", sed do eiusmod tempor incididu"..., 100) = 100

futex(0x7f226b042000, FUTEX_WAKE, 1)   = 1

read(0, " minim veniam, quis nostrud\nexer"..., 100) = 100

futex(0x7f226b042000, FUTEX_WAKE, 1)   = 1

read(0, "   "..., 100) = 100

futex(0x7f226b042000, FUTEX_WAKE, 1)   = 1

read(0, "   ut aliquip ex ea commod"..., 100) = 100

futex(0x7f226b042000, FUTEX_WAKE, 1)   = 1

read(0, "tate velit esse cillum dolore\nue"..., 100) = 75

futex(0x7f226b042000, FUTEX_WAKE, 1)   = 1

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 13155

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13155, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

munmap(0x7f226b042000, 32)   = 0

munmap(0x7f226b041000, 32)   = 0

exit_group(0)   = ?

+++ exited with 0 +++

alex@alex-lenovo:~/CLionProjects/os_lab_04/src/cmake-build-debug$ cat output

Lorem ipsum dolor sit amet consectetur

adipiscing elit, sed do eiusmod tempor incididunt ut labore

et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud

exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore

ue fugiat nulla pariatur.

alex@alex-lenovo:~/CLionProjects/os_lab_04/src/cmake-build-debug$./os_lab_04 output < empty

alex@alex-lenovo:~/CLionProjects/os_lab_04/src/cmake-build-debug$ cat output

Вывод

В результате данной лабораторной работы мной был изучен механизм отображения файлов в виртуальное адресное пространство. Были приобретены навыки отладки программ, имеющих больше одного процесса с помощью GDB. Так же я научился синхронизировать работу процессов, используя семафоры.