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

Лабораторная работа №4 по курсу

«Операционные системы»

Студент: Ядров Артем Леонидович
Группа: М8О-208Б-20
Вариант: 12
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

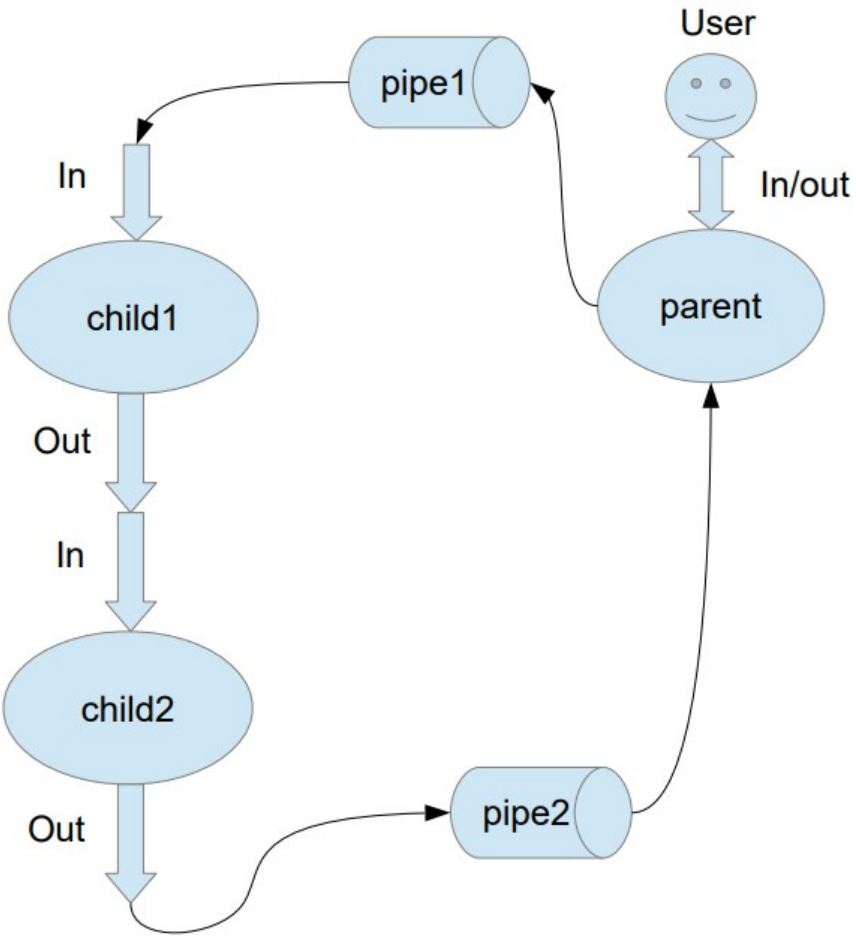
https://github.com/Yadroff/OS/tree/master/2_lab

Постановка задачи

Цель работы
Приобретение практических навыков в:
· Освоение принципов работы с файловыми системами
· Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание
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 системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.



12 вариант) Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, fcntl.h, errno.h, sys/mman.h, sys/stat.h, string.h, stdbool.h, ctype.h, sys/wait.h, semaphore.h. В программе используются следующие системные вызовы:

1. shm_open - создаёт/открывает объекты общей памяти POSIX.
2. sem_open - инициализирует и открывает именованный семафор.
3. ftruncate - обрезает файл до заданного размера.
4. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
5. memset - заполнение памяти значением определённого байта.
6. sem_getvalue - возвращает значение семафора.
7. close - закрывает файловый дескриптор.
8. sem_close - закрывает именованный семафор.
9. execl - запуск файла на исполнение.
10. sem_getvalue - возвращает значение семафора.
11. sem_wait - блокирует семафор.
12. sem_post - разблокирует семафор.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить работу с отображением файла в память(mmap и munmap).
2. Изучить работу с процессами(fork).
3. Создать 2 дочерних и 1 родительский процесс.

4. В каждом процессе отобразить файл в память, преобразовать в соответствии с вариантом и снять отображение(mmap, munmap).

Исходный код

main.c

```
#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <semaphore.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>

#include "shrmem.h"

int main() {
  size_t map_size = 0;
  char *in = (char *)calloc(1, sizeof(char));
  char c;
  while ((c = getchar()) != EOF) {
        in[map_size] = c;
```

```
        ++map_size;
        in = (char *)realloc(in, (map_size + 1) * sizeof(char));
    }
    in[map_size] = '\0';
    //read string stream
    int fd = shm_open(BackingFile, O_RDWR | O_CREAT, AccessPerms);
    if (fd == -1) {
        perror("OPEN");
        exit(EXIT_FAILURE);
    }
    sem_t *semptr = sem_open(SemaphoreName, O_CREAT, AccessPerms, 3);
    if (semptr == SEM_FAILED) {
        perror("SEM_OPEN");
        exit(EXIT_FAILURE);
    }
    int val;

    ftruncate(fd, map_size);
    caddr_t memptr = mmap(
            NULL,
            map_size,
            PROT_READ | PROT_WRITE,
            MAP_SHARED,
            fd,
            0);
    if (memptr == MAP_FAILED) {
        perror("MMAP");
        exit(EXIT_FAILURE);
    }
```

```
memset(memptr, '\0', map_size);
sprintf(memptr, "%s", in);
free(in);
if (sem_getvalue(semptr, &val) != 0) {
        perror("SEM_GETVALUE");
        exit(EXIT_FAILURE);
}
while (val++ < 2) {// if semaphore already was created, just fill it up to 2
        sem_post(semptr);
}
int pid_0 = 0;
int pid_1 = 0;
if ((pid_0 = fork()) == 0) {
        munmap(memptr, map_size);
        close(fd);
        sem_close(semptr);
        char str[10];
        sprintf(str, "%lu", map_size);
        execl("4_lab_child_0", "4_lab_child_0", str, NULL);
        perror("EXECL");
        exit(EXIT_FAILURE);
}
while (true) {
        if (sem_getvalue(semptr, &val) != 0) {
          perror("SEM_GETVALUE");
          exit(EXIT_FAILURE);
        }
        if (val == 0) {
          if (sem_wait(semptr) == -1) {
```

```
                perror("SEM_POST");
                exit(EXIT_FAILURE);
            }
            char *string = (char *)malloc(strlen(memptr) * sizeof(char));
            strcpy(string, memptr);
            printf("%s", string);
            free(string);
            return EXIT_SUCCESS;
        }
    }
}
```

child_0.c

```
//
// Created by Temi4 on 17.10.2021.
//
#include <assert.h>
#include <ctype.h>
#include <fcntl.h>
#include <semaphore.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>

#include "shrmem.h"
```

```
int main(int argc, char **argv) {
  assert(argc == 2);
  const size_t map_size = atoi(argv[1]);
  int map_fd = shm_open(BackingFile, O_RDWR, AccessPerms);
  if (map_fd < 0) {
        perror("SHM_OPEN");
        exit(EXIT_FAILURE);
  }
  caddr_t memptr = mmap(
          NULL,
          map_size,
          PROT_READ | PROT_WRITE,
          MAP_SHARED,
          map_fd,
          0);
  if (memptr == MAP_FAILED) {
        perror("MMAP");
        exit(EXIT_FAILURE);
  }
  sem_t *semptr = sem_open(SemaphoreName, O_CREAT, AccessPerms, 2);
  if (semptr == SEM_FAILED) {
        perror("SEM_OPEN");
        exit(EXIT_FAILURE);
  }
  if (sem_wait(semptr) != 0) {
        perror("SEM_WAIT");
        exit(EXIT_FAILURE);
  }
  char *string = (char *)malloc(strlen(memptr) * sizeof(char));
```

```
    strcpy(string, memptr);
    for (int i = 0; i < map_size; ++i) {// преобразование
            string[i] = toupper(string[i]);
    }
    memset(memptr, '\0', map_size);
    sprintf(memptr, "%s", string);
    free(string);
    pid_t pid = fork();
    if (pid == 0) {
            munmap(memptr, map_size);
            close(map_fd);
            sem_close(semptr);
            char str[10];
            sprintf(str, "%lu", map_size);
            execl("4_lab_child_1", "4_lab_child_1", str, NULL);
            perror("EXECL");
            exit(EXIT_FAILURE);
    } else if (pid == -1) {
            perror("FORK");
            exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```

child_1.c

```
//
// Created by Temi4 on 17.10.2021.
//
#include <assert.h>
```

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

#include "shrmem.h"

int main(int argc, char **argv) {
  assert(argc == 2);
  const size_t map_size = atoi(argv[1]);
  int map_fd = shm_open(BackingFile, O_RDWR, AccessPerms);
  if (map_fd < 0) {
        perror("SHM_OPEN");
        exit(EXIT_FAILURE);
  }
  caddr_t memptr = mmap(
          NULL,
          map_size,
          PROT_READ | PROT_WRITE,
          MAP_SHARED,
          map_fd,
          0);
  if (memptr == MAP_FAILED) {
        perror("MMAP");
```

```
        exit(EXIT_FAILURE);
    }
    sem_t *semptr = sem_open(SemaphoreName, O_CREAT, AccessPerms, 2);
    if (semptr == SEM_FAILED) {
        perror("SEM_OPEN");
        exit(EXIT_FAILURE);
    }
    if (sem_wait(semptr) != 0) {
        perror("SEM_WAIT");
        exit(EXIT_FAILURE);
    }
    char *string = (char *)calloc(map_size, sizeof(char));
    char *out = (char *)calloc(1, sizeof(char));
    size_t m_size = 0;
    strcpy(string, memptr);
    for (int i = 0; i < map_size; ++i) {// преобразование
        if (string[i] == ' ' && string[i + 1] == ' ') {
          ++i;
          continue;//
        }
        out[m_size] = string[i];
        ++m_size;
        out = (char *)realloc(out, (m_size + 1) * sizeof(out));
    }
    out[m_size] = '\0';
    memset(memptr, '\0', m_size);
    sprintf(memptr, "%s", out);
    out = (char*)calloc(m_size, sizeof(char));
    strcpy(out, memptr);
```

```
  close(map_fd);
  sem_post(semptr);
  sem_close(semptr);
  return EXIT_SUCCESS;
}
```

Демонстрация работы программы

```
[Temi4@localhost src]$ cat test.txt
Hello   World!
I  am    learning OS
I love C++
And   you)))
[Temi4@localhost src]$ gcc -pthread -lrt main.c
[Temi4@localhost src]$ gcc -pthread -lrt child_0.c -o 4_lab_child_0
[Temi4@localhost src]$ gcc -pthread -lrt child_1.c -o 4_lab_child_1
[Temi4@localhost src]$ ./a.out < test.txt
HELLO WORLD!
IAMLEARNING OS
I LOVE C++
AND YOU)))
```

Выводы

В Си помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.