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: Пивницкий Д.С.
Группа: М8о–206Б–19
Вариант: 11
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020.

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.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Вариант 11: Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Общие сведения о программе

Программа компилируется из файла lab4.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. **mmap** – создает отображение файла в память.
2. **munmap** – снимает отображение.
3. **open** – открывает файл.
4. **close –** закрывает файл.
5. **sem_init –** инициализация семафора.
6. **sem_wait** – ожидание доступа, если значение семафора отрицательное, то вызывающий поток блокируется до тех пор, пока один из потоков не вызовет sem_post.
7. **sem_post** – увеличивает значение семафора и разблокирует ожидающие потоки.
8. **sem_destroy** – уничтожает семафор.
9. **read** – чтение из файла в буфер.
10. **write** – запись из буфера в файл.
11. **sleep** – переход в режим ожидания на указанное количество секунд.
12. **exit** – завершение работы программы с некоторым статусом.

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).

Основные файлы программы

lab4.c:

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

void change_spaces(char* src, int size)
{
    char* res = malloc(size*sizeof(char));
    int j = 0;
    bool flag = true;
    for(int i = 0; i < size ; ++i) {
    if(src[i] == ' ')
    src[i] = '_';
    }
}

int main(int argc, char* argv[])
{
      if(argc != 3)
      {
            printf("INVALID COUNT OF ARGS\nUSAGE: %s <file>\n", argv[0]);
            exit(-1);
      }

      int fd_0 = -1;
      int fd_1 = -1;
      char* src;
```

```
struct stat statbuf;

if((fd_0 = open(argv[1], O_RDWR)) < 0)
{
      printf("OPEN ERROR\n");
      exit(-1);
}

if((fd_1 = open(argv[2], O_CREAT | O_RDWR, S_IRUSR | S_IWUSR)) < 0)
{
      printf("OPEN ERROR\n");
      exit(-1);
}

if(fstat(fd_0, &statbuf) < 0)
{
      printf("FSTAT ERROR\n");
      exit(-1);
}

if(ftruncate(fd_1, statbuf.st_size) < 0)
{
      printf("FTRUNCATE ERROR\n");
      exit(-1);
}

char buff[statbuf.st_size];
if(read(fd_0, buff, statbuf.st_size) != statbuf.st_size)
{
      printf("READ ERROR\n");
      exit(-1);
}
if(write(fd_1, buff, statbuf.st_size) != statbuf.st_size)
{
      printf("READ ERROR\n");
      exit(-1);
}
```

```
	int pid_0 = 0;
	int pid_1 = 0;
	int status_0 = 0;
	int status_1 = 0;

	sem_t semaphore;
	sem_init(&semaphore, 0, 1);

	if((pid_0 = fork()) > 0)
	{
		if((pid_1 = fork()) > 0)
		{
			sem_wait(&semaphore);
			sleep(2);
			waitpid(pid_1, &status_1, WNOHANG);
			waitpid(pid_0, &status_0, WNOHANG);
			src = (char*)mmap(0, statbuf.st_size, PROT_READ, MAP_SHARED,
fd_1, 0);
			if(src == MAP_FAILED)
			{
				printf("MMAP ERROR\n");
				exit(-1);
			}
			for(int i = 0; i < statbuf.st_size; ++i) { printf("%c",
src[i]); }
			printf("\n");
			if(munmap(src, statbuf.st_size) != 0)
			{
				printf("MUNMAP ERROR\n");
				exit(-1);
			}
			sleep(2);
			sem_post(&semaphore);
		}
		else if(pid_1 == 0)
		{
			sem_wait(&semaphore);
```

```
                sleep(1);
                src = (char*)mmap(0, statbuf.st_size, PROT_READ |
PROT_WRITE, MAP_SHARED, fd_1, 0);
                if(src == MAP_FAILED)
                {
                      printf("MMAP ERROR\n");
                      exit(-1);
                }
                change_spaces(src, statbuf.st_size);
                if(munmap(src, statbuf.st_size) != 0)
                {
                      printf("MUNMAP ERROR\n");
                      exit(-1);
                }
                sleep(1);
                sem_post(&semaphore);
          }
          else
          {
                printf("FORK ERROR 1\n");
                exit(-1);
          }
    }
    else if (pid_0 == 0)
    {
          sem_wait(&semaphore);
          sleep(1);
          src = (char*)mmap(0, statbuf.st_size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd_1, 0);
          if(src == MAP_FAILED)
          {
                printf("MMAP ERROR\n");
                exit(-1);
          }
          for(int i = 0; i < statbuf.st_size; ++i) { src[i] =
toupper(src[i]); }
          if(munmap(src, statbuf.st_size) != 0)
          {
                printf("MUNMAP ERROR\n");
```

```
                exit(-1);
            }
            sleep(1);
            sem_post(&semaphore);
        }
        else
        {
            printf("FORK ERROR 2\n");
            exit(-1);
        }

        sem_destroy(&semaphore);
        close(fd_0);
        close(fd_1);
        return 0;
}
```

Пример работы

daniel@daniel-Ideapad-Z570: ~ cd os/oslab4

daniel@daniel-Ideapad-Z570: ~ oslab4 cat test.txt

  Hello   wOrld!

Bye   Bye

  hot   chilli

reS   grand

daniel@daniel-Ideapad-Z570: ~ oslab4 gcc lab4.c -o out -pthread

daniel@daniel-Ideapad-Z570: ~ oslab4 ./out test.txt res.txt

___HELLO_____WORLD!

BYE_____BYE

___HOT_____CHILLI

RES____GRAND

Вывод

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.