Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Memory mapped files**

Студент: Гаврилов Максим Сергеевич

Группа: М8О–206Б–20

Вариант: 7

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Освоении принципов работы с файловыми системами
* Обеспечение обмена данными 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 решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или отображаемые в память файлы.

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

**Общие сведения о программе**

Программа компилируется из файлов main.c, child\_proc.c, tramsformres.h. Используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, string.h, fcntl.h, pthread.h, time.h sys/types.h sys/mman.h sys/wait.h. В программе используются следующие системные вызовы:

1. **Mmap –** производит отображение файла в память. После этого к нему можно обращаться по адресу в *heap*
2. **Open –** преобразует путь к файлу в файловый дескриптор.
3. **Close –** закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно.
4. **Fork –** порождает новый процесс (процесс-потомок), который почти идентичен порождающему процессу-родителю.

Синхронизация работы производится с помощью системных сигналов (sigusr1, sigusr2). В каждой программе есть переменная-маркер, которая инкрементируется, если был получен тот или иной сигнал. Таким образом, прежде чем процесс запишет что-либо в отображенный файл (если к этому файлу подключено сразу несколько процессов), он дожидается, пока будет получен сигнал, означающий, что можно писать.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. **Найти способ синхронизации потоков, без использования пайпов.**  
   Для синхронизации я решил использовать сигналы, так как отображаемые файлы я уже применял для пересылки данных
2. **Найти оптимальный способ обработки сигналов.**  
   Так как мои процессы постоянно перекидываются сигналами, использовать обычный *signal* было нельзя: если во время обработки придет еще один сигнал, то он не будет обработан. Таким образом, я пришел к использованию *sigaction*.
3. **Работа с memory mapped files** осуществляется довольно просто, так что можно считать, что основные трудности закончились на сигналах.

**Основные файлы программы**

**Child\_proc.c**

|  |
| --- |
| #include <string.h>  #include "unistd.h"  #include "stdio.h"  #include "stdlib.h"  #include "fcntl.h"  #include <sys/types.h>  #include "tramsformres.h"  #include <sys/mman.h>  #include <signal.h>  //#include "thread.h"  //#include "synch.h"  #define MAP\_FILE\_SIZE 1000  #define TIMEOUT\_TIMER 1000000000  char syncTag **=** 0**;**  int upCount **=** 0**;**  void taghdl**(**int sig**)**  **{**  **if(**sig **==** SIGUSR1**)**  **{**  syncTag **=** 1**;**  **}**  **else**  **if(**sig **==** SIGUSR2**)**  **{**  upCount**++;**  **}**  **}**  int main**(**int argc**,**char**\*** argv**[])**  **{**  struct sigaction act\_sync**;**  memset**(&**act\_sync**,** 0**,** **sizeof(**act\_sync**));**  act\_sync**.**sa\_handler **=** taghdl**;**  sigset\_t set\_sync**;**  sigemptyset**(&**set\_sync**);**  sigaddset**(&**set\_sync**,** SIGUSR1**);**  sigaddset**(&**set\_sync**,** SIGUSR2**);**  act\_sync**.**sa\_mask **=** set\_sync**;**  sigaction**(**SIGUSR1**,** **&**act\_sync**,** **NULL);**  sigaction**(**SIGUSR2**,** **&**act\_sync**,** **NULL);**    //int fd1[1];fd1[0] = argv[1][0];  int l **=** strlen**(**argv**[**0**]);**  char filename1**[**l**];** strcpy**(**filename1**,**argv**[**0**]);**  printf**(**"%d\n | "**,**l**);**  int id **=** back\_transform**(**argv**[**1**]);**  printf**(**"child [%d] parent id is %d\n"**,**getpid**(),**id**);**  int f1**,**ftmp**;**  char**\*** adr**;**  //код дочернего процесса  printf**(**"[%d] It's child\n"**,**getpid**());**  char str**[**100**];**  f1 **=** open**(**filename1**,**O\_CREAT **|** O\_WRONLY **|** O\_APPEND**,** 00660**);**//открываем все нужные файлы, проводим отображение  **if(**f1 **==** **-**1**)** **{**  printf**(**"child: ERROR: unable to open output file\n"**);**  **return** 0**;**  **}**  ftmp **=** open**(**"twpfile"**,** O\_RDWR**,** 00660**);**  **if(**f1 **==** **-**1**)** **{**  printf**(**"child: ERROR: unable to open exchange file\n"**);**  **return** 0**;**  **}**  //отображение файла  adr **=** mmap**(NULL,**MAP\_FILE\_SIZE**,**PROT\_READ **|** PROT\_WRITE**,** MAP\_SHARED**,**ftmp**,**0**);**  **if(**adr **==** MAP\_FAILED**)**  **{**  printf**(**"failed to map\n"**);**  **return** 0**;**  **}**  kill **(**id**,**SIGUSR1**);** //отсылаем родителю сигнал о готовности читать сообщения  printf**(**"child: ready to read\n"**);**  **for(;;)**  **{**  int i**=**0**;**  **while(**syncTag **==** 0**){**//ждем, пока нам разрешат прочесть  **++**i**;**  **if(**i**==**TIMEOUT\_TIMER**)** **{**  **if(**upCount **==** 0**)**  **{**  printf**(**"chid: ERROR: syncroniser timed out\n"**);**  **return** 0**;**  **}**  **else**  **{**  upCount**--;**  i**=**0**;**  **}**  **}**  **}**  syncTag **=** 0**;**  strcpy**(**str**,**adr**);**  printf**(**"[%d] read: %s\n"**,**getpid**(),**str**);**  write**(**f1**,**str**,**strlen**(**str**));**  write**(**f1**,**"\n"**,**1**);**  kill**(**id**,**SIGUSR1**);**//сигнализируем о прочтении  **}**  munmap**(**adr**,**MAP\_FILE\_SIZE**);**  close**(**ftmp**);**  close**(**f1**);**  **return** 0**;**  **}** |

**Main.c**

|  |
| --- |
| //Гаврилов М.С. М8О-206Б-19 | Лр#4, вар. 21  // 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.  #include <string.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include "sys/types.h"  #include <sys/mman.h>  #include <signal.h>  #include "sys/wait.h"  #include "tramsformres.h"  //#include "thread.h"  //#include "synch.h"  #define MAP\_FILE\_SIZE 1000  #define TIMEOUT\_TIMER 1000000000  int id1 **=** **-**1**,**id2 **=** **-**1**;**  int a**;** //дескриптор файла  char**\*** adr **=** **NULL;** //адрес отображаемого куска  char syncTag **=** 0**;**  char resSync **=** 0**;**  void hdl**(**int sig**)**  **{**  **if(**sig **==** SIGUSR1**)**  **{**  syncTag**++;**  **}**  **else**  **if(**sig **==** SIGCHLD**)**  **{**  kill**(**id1**,**SIGKILL**);**  kill**(**id2**,**SIGKILL**);**  kill**(**getpid**(),**SIGKILL**);**  **}**  **}**  char**\*** bitescan **(**char**\*** str**)** //принимает неинициализированный указатель, маллочит для него память и считывает строку  **{**  int counter **=** 0**,**size **=** 2**;**  str **=** **(**char**\*)**realloc**(**str**,**size**);**  char buf**;**  int mark**=**0**;**  **while(**read**(**0**,&**buf**,**1**)!=**0**)** **{**  **if(**buf **==** ' '**||**buf **==** '\n'**||**buf **==**'\t'**)** **{**  **if(**mark **==** 0**)** **{**  **continue;**  **}**  **else** **{**  **break;**  **}**  **}**  mark**=**1**;**  **if(**counter**+**1 **==** size**)** **{**  size**\*=**2**;**  str **=** **(**char**\*)**realloc**(**str**,**size**);**  **}**  str**[**counter**]** **=** buf**;**  **++**counter**;**  **}**  **if(**counter **==** 0**)** **{** //если ничего не считали  free**(**str**);**  str **=** **NULL;**  **return** str**;**  **}**  //str = (char\*)realloc(str,counter); //если раскомментить, то почему-то ругается валгринд  str**[**counter**]** **=** 0**;**  **return** str**;**  **}**  void terminate**()** **{**  munmap**(**adr**,**MAP\_FILE\_SIZE**);**  close**(**a**);**  kill**(**id1**,**SIGKILL**);**  kill**(**id2**,**SIGKILL**);**  **}**  int main**()**  **{**  printf**(**"Input two filenames\nthen input strings to be inverted\n"**);**  char**\*** filename1 **=** **NULL;**  char**\*** filename2 **=** **NULL;**  filename1 **=** bitescan**(**filename1**);**  **if(**filename1 **==** **NULL)** **{**  **return** 0**;**  **}**  **for(;;)** **{**  filename2 **=** bitescan**(**filename2**);**  **if(**filename2 **==** **NULL)** **{**  **return** 0**;**  **}**  **if(**strcmp**(**filename1**,**filename2**)** **!=** 0**)** **{**  **break;**  **}**  **else** **{**  printf**(**"please input string differ to name one\n"**);**  **}**  **}**  a **=** open**(**filename1**,**O\_CREAT **|** O\_WRONLY **|** O\_TRUNC**,** 00660**);** //создание первого файла с результатами  write**(**a**,**"This is file one\n(even)\n\n"**,**strlen**(**"This is file one\n(even)\n\n"**));**  close**(**a**);**  a **=** open**(**filename2**,**O\_CREAT **|** O\_WRONLY **|** O\_TRUNC**,** 00660**);** //создаие второго файла с результатами  write**(**a**,**"This is file two\n(odd)\n\n"**,**strlen**(**"This is file two\n(odd)\n\n"**));**  close**(**a**);**    a **=** open**(**"twpfile"**,**O\_CREAT **|** O\_WRONLY **|** O\_TRUNC**,** 00660**);** //создание и инициализация отображаемого файла  **for(**int i**=**0**;**i**<**MAP\_FILE\_SIZE**;++**i**)** **{**  write**(**a**,**"0"**,**1**);** //я хотел проинициализировать его нулями, но он не хочет в текстовый файл писать настоящие нули (  **}**  close**(**a**);**  a **=** open**(**"twpfile"**,** O\_RDWR**,** 00660**);**  printf**(**"initiated\n"**);**  //отображение файла    adr **=** mmap**(NULL,**MAP\_FILE\_SIZE**,**PROT\_READ **|** PROT\_WRITE**,** MAP\_SHARED**,**a**,**0**);**  **if(**adr **==** MAP\_FAILED**)**  **{**  printf**(**"failed to map\n"**);**  **return** 0**;**  **}**  munmap**(**adr**,**MAP\_FILE\_SIZE**);**  close**(**a**);**//конец фрагмента с отображением  int pid **=** getpid**();**  printf**(**"initiating process separation\n"**);**  id1 **=** fork**();** //разделение процесса, в основном процессе id будет соответствовать pid порожденного, в порожденном id = 0  **if(**id1 **==** **-**1**)** //ошибка  **{**  write**(**1**,**"error\n"**,**strlen**(**"error\n"**));**  **return** **-**1**;**  **}**  **else**  **if(**id1 **==** 0**)**  **{**  //char arg1[1]; arg1[0] = fd1[0];  execl**(**"chid\_proc\_one.out"**,**filename1**,**transform**(**pid**),NULL);**  **}**  **else**  **{**  //код основного процесса  id2 **=** fork**();**  **if(**id2 **==** **-**1**)**  **{**  write**(**1**,**"error in genetating str 2\n"**,**strlen**(**"error in genetating str 2\n"**));**  **return** **-**1**;**  **}**  **else**  **if(**id2 **==** 0**)**  **{**  execl**(**"chid\_proc\_one.out"**,**filename2**,**transform**(**pid**),NULL);**  **}**  **else**  **{**  struct sigaction act**;**  memset**(&**act**,** 0**,** **sizeof(**act**));**  act**.**sa\_handler **=** hdl**;**  sigset\_t set**;**  sigemptyset**(&**set**);**  sigaddset**(&**set**,** SIGUSR1**);**  sigaddset**(&**set**,** SIGUSR2**);**  act**.**sa\_mask **=** set**;**  sigaction**(**SIGUSR1**,** **&**act**,** 0**);**  sigaction**(**SIGUSR2**,** **&**act**,** 0**);**  //основной процесс  printf**(**"[%d] It's parent, child id: [%d] [%d]\n"**,**getpid**(),**id1**,**id2**);**  /\*void (\*fptr)(int) = NULL; //установка обработчика сигнала  fptr = hdl;  signal(SIGUSR1,fptr);\*/  //отображение файла  a **=** open**(**"twpfile"**,** O\_RDWR**,** 00660**);**  adr **=** mmap**(NULL,**MAP\_FILE\_SIZE**,**PROT\_READ **|** PROT\_WRITE**,** MAP\_SHARED**,**a**,**0**);**  unsigned long long i**=**0**;**  **while(**syncTag **<** 2**)** **{**  **++**i**;**  **if(**i**==**TIMEOUT\_TIMER**)** **{**  printf**(**"ERROR: syncroniser timed out. Child didn't start correctly\n"**);**  munmap**(**adr**,**MAP\_FILE\_SIZE**);**  close**(**a**);**  kill**(**id1**,**SIGKILL**);**  kill**(**id2**,**SIGKILL**);**  **return** 0**;**  **}**  **}**  syncTag **=** 0**;**//поймали сигнал от первого ребентак  printf**(**"got\_signal\_one\n"**);**  printf**(**"got\_signal\_two\n"**);**//поймали сигнал от второго ребенка  //дети готовы принимать сообщения  char**\*** inp **=** **NULL;**  int l**=**0**;**  **if(**adr **==** MAP\_FAILED**)**  **{**  printf**(**"failed to map\n"**);**  **return** 0**;**  **}**  **while(**0**==**0**)**  **{**  inp **=** bitescan**(**inp**);**  **if(**inp **==** **NULL)** **{**  **break;**  **}**  printf**(**"[%d] writing: %s\n"**,**getpid**(),**inp**);**  l **=** strlen**(**inp**);**  strcpy**(**adr**,**inp**);**  **if(**l **%** 2 **==** 0**)**  **{**  syncTag **=** 0**;**  kill**(**id1**,**SIGUSR1**);**//посылаем ребенку сигнал разрешающий считать сообщение  kill**(**id2**,**SIGUSR2**);**  **while(**syncTag **==** 0**)** **{**//ждем ответа от ребенка (что он считал)  **++**i**;**  **if(**i**==**TIMEOUT\_TIMER**)** **{**  printf**(**"ERROR: syncroniser timed out\n"**);**  terminate**();**  **return** 0**;**  **}**  **}**  **}**  **else**  **{**  syncTag **=** 0**;**  kill**(**id2**,**SIGUSR1**);**  kill**(**id1**,**SIGUSR2**);**  **while(**syncTag **==** 0**)** **{**  **++**i**;**  **if(**i**==**TIMEOUT\_TIMER**)** **{**  printf**(**"ERROR: syncroniser timed out\n"**);**  terminate**();**  **return** 0**;**  **}**  **}**  **}**  **}**  terminate**();**  printf**(**"[%d] parent: stop\n"**,**getpid**());**  **return** 0**;**  **}**  **}**  //общий код  **return** 0**;**  **}** |

**Пример работы**

**max@max-Swift:~/Рабочий стол/OS/lab4$ make**

*gcc --std=c99 lab4\_try.c -o exetry.out*

*gcc --std=c99 chid\_proc.c -o chid\_proc.out*

**max@max-Swift:~/Рабочий стол/OS/lab4$ ./exetry.out**

*Input two filenames*

*then input strings to be inverted*

file\_1

file\_2

*initiating process separation*

*parent: It's parent*

*It's child one*

*It's child two*

string\_one

*one: string got*

*one: result written*

longer\_string

*two: string is got*

*two: result written*

qwerty

*one: string got*

*one: result written*

qwert

*two: string is got*

*two: result written*

qwer

*one: string got*

*one: result written*

qwe

*two: string is got*

*two: result written*

qw

*one: string got*

*one: result written*

q

*two: string is got*

*two: result written*

*one: stop*

*parent: stop*

*two: stop*

**max@max-Swift:~/Рабочий стол/OS/lab4$ cat file\_1**

*This is file one*

*(even)*

*eno\_gnirts*

*ytrewq*

*rewq*

*wq*

**max@max-Swift:~/Рабочий стол/OS/lab4$ cat file\_2**

*This is file two*

*(odd)*

*gnirts\_regnol*

*trewq*

*ewq*

*q*

**max@max-Swift:~/Рабочий стол/OS/lab4$**

**Вывод**

Писать отчет через два месяца после завершения работы над лабораторной – не очень продуктивное занятие. Пожалуй, это единственный содержательный вывод, который я могу сформулировать в данный момент, ибо вспоминать, что мне пришло в голову в ходе создания этих программ мне удается с трудом. Как минимум, я научился обрабатывать сигналы, что, безусловно, крайне важно: почти ни одна моя последующая работа в рамках курса ОС не обходилась без их использования. Навык работы с отображаемыми файлами мне вроде пока не пригождался, но я думаю, все еще впереди.