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: Путилин Д.Н..

Группа: М8О-207Б-21

Вариант: 13

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

[https://github.com/](https://github.com/Liguha/OS)putilin21dn/OC

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

1. Освоение принципов работы с файловыми системами
2. 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.

**Общие сведения о программе**

Программа родительского процесса компилируется из main.c, использует заголовочные файлы stdio.h, stdlib.h, unistd.h, sys/mman.h, fcntl.h, semaphore.h, string.h, errno.h. В программе используются следующие системные вызовы:

1. unlink() – удаление имени из файловой системы
2. fork() – создание дочернего процесса
3. open() – открытие файла
4. close() – закрытие файла
5. write() – запись последовательности байт
6. lseek() - установка смещения в файловом дескрипторе
7. mmap() - создание отражения файла в памяти
8. munmap() - удаление отражения файла в памяти

**Общий метод и алгоритм решения**

На вход подается файл, в котором находится текст. Дочерний процесс должен привести все буквы к нижнему регистру и заменить пробелы на нижние подчеркивание. Результат обратно передается в родительский процесс и выводится в терминал.

Синхронизация процессов достигается по средствам 2 семафоров, так после прочтения строки и записи её в образ файла родительский процесс открывает семафор 1 и начинает ждать открытия семафора 2. Открытие семафора 1 позволяет дочернему процессу обработать текст и записать результат в память. После чего в родительском процессе произойдет вывод результата в консоль.

**Исходный код**

|  |
| --- |
| **main.c** |
| #include "stdio.h"  #include "stdlib.h"  #include "unistd.h"  #include "fcntl.h"  #include "sys/mman.h"  #include "string.h"  #include "errno.h"  #include "semaphore.h"  #define CHECK\_ERROR(expr, message) \  do \  { \  int res = (expr); \  if (res == -1) \  { \  perror(message); \  return -1; \  } \  } while (0)  #define UNLINK\_ERROR(expr, message) \  do \  { \  int res = (expr); \  if (res == -1 && errno == EACCES) \  { \  perror(message); \  return -1; \  } \  } while (0)  const int MAX\_LENGTH = 10000;  const int SIZE = MAX\_LENGTH + sizeof(int);  const int zero = 0;  int main(){  UNLINK\_ERROR(unlink("file1"), "unlink error");  int file1 = open("file1", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);  if (file1 == -1 )  {  perror("open error");  return -1;  }  CHECK\_ERROR(lseek(file1, SIZE - 1, SEEK\_SET), "lseek error");  write(file1, &zero, 1);  sem\_t\* sem1 = sem\_open("semaphore1", O\_CREAT, S\_IRUSR | S\_IWUSR, 0);  sem\_t\* sem2 = sem\_open("semaphore2", O\_CREAT, S\_IRUSR | S\_IWUSR, 0);  if (sem1 == SEM\_FAILED || sem2 == SEM\_FAILED )  {  perror("sem\_open error");  return -1;  }  int id = -1;  CHECK\_ERROR(id = fork(), "fork error");  // child  if (id == 0)  {  void\* in = mmap(NULL, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file1, 0);  if (in == MAP\_FAILED )  {  perror("mmap error");  return -1;  }  char\* str = (char \*)calloc(MAX\_LENGTH, sizeof(char));  char\* out = (char \*)malloc(sizeof(char)\*MAX\_LENGTH);  if (str == NULL)  {  perror("calloc error");  return -1;  }  CHECK\_ERROR(sem\_wait(sem1), "sem\_wait error");  memcpy(str, in, sizeof(char)\*MAX\_LENGTH);  out[0]=0;  for(int i=1; i<str[0]+1;++i){  out[0]++;  if(str[i]>='A' & str[i]<='Z'){  out[out[0]] = str[i] - 'A' + 'a';  }  else{  out[out[0]] = str[i];  }  if(str[i]==' '){  out[out[0]] = '\_';  }  }  out[0]++;  memcpy(in, out, MAX\_LENGTH\*sizeof(char));  CHECK\_ERROR(sem\_post(sem2), "sem\_post error");  CHECK\_ERROR(munmap(in, SIZE), "munmap error");  free(str);  free(out);  }  // parent  if (id>0){  void\* out = mmap(NULL, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file1, 0);  if (out == MAP\_FAILED )  {  perror("mmap error");  return -1;  }  char \*in = (char \*)malloc(sizeof(char)\*MAX\_LENGTH);  in[0] = 0;  char c;  while ((c = getchar()) != EOF) {  in[0] += 1;  in[in[0]] = c;  }  in[in[0]] = '\0';  memcpy(out, in, MAX\_LENGTH\*sizeof(char));  CHECK\_ERROR(sem\_post(sem1), "sem\_post error");  CHECK\_ERROR(sem\_wait(sem2), "sem\_wait error");  char\* str = (char \*)calloc(MAX\_LENGTH, sizeof(char));  memcpy(str, out, sizeof(char)\*MAX\_LENGTH);  for (int i=1; i<str[0]+1;++i){  printf("%c",str[i]);  }  CHECK\_ERROR(munmap(out, SIZE), "munmap error");  sem\_close(sem1);  sem\_close(sem2);  printf("\n");  free(in);  free(str);  }  CHECK\_ERROR(close(file1), "close error");  unlink("file1");  } |

**Демонстрация работы программы**

dmitry@dmitry-VirtualBox:~/Рабочий стол/OC/lab4$ cat test

Hello

My name is Dima !

.. ... gg vp

nt

gh

dmitry@dmitry-VirtualBox:~/Рабочий стол/OC/lab4$ ./main.out < test

hello\_\_\_

my\_name\_is\_dima\_\_!

..\_...\_gg\_vp\_

nt\_

\_\_\_\_\_\_gh

**Выводы**

Составлена и отлажена программа на языке Си, осуществляющая работу и взаимодействие между процессами с использованием отображаемых файлов. Так, получены навыки в обеспечении обмена данных между процессами посредством технологии «File mapping».