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: Смирнов А.В.

Группа: М8О-207Б-21

Вариант: 16

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/Liguha/OS>

**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 file\_mapping.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**

Родительский процесс крутиться в бесконечном цикле, пока не получит на вход пустую строку — знак завершения работы. Аналогично в цикле находится и дочерний процесс — обработчик строк. 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 строку, записать результат в образ второго файла, открыть семафор 2 и закрыть семафор 1. Тем самым продолжается работа родительского процесса, который считывает результат из образа второго файла и выводит ошибку, если она была.

**Исходный код**

|  |
| --- |
| **file\_mapping.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");  UNLINK\_ERROR(unlink("file2"), "unlink error");  int file1 = open("file1", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);  int file2 = open("file2", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);  if (file1 == -1 || file2 == -1)  {  perror("open error");  return -1;  }  CHECK\_ERROR(lseek(file1, SIZE - 1, SEEK\_SET), "lseek error");  CHECK\_ERROR(lseek(file2, SIZE - 1, SEEK\_SET), "lseek error");  CHECK\_ERROR(write(file1, &zero, 1), "write error");  CHECK\_ERROR(write(file2, &zero, 1), "write error");  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, MAP\_SHARED, file1, 0);  void\* ans = mmap(NULL, SIZE, PROT\_WRITE, MAP\_SHARED, file2, 0);  if (in == MAP\_FAILED || ans == MAP\_FAILED)  {  perror("mmap error");  return -1;  }  UNLINK\_ERROR(unlink("result.txt"), "unlink error");  int fout = open("result.txt", O\_CREAT | O\_WRONLY, S\_IRUSR);  if (fout == -1)  {  perror("open error");  return -1;  }  char\* str = calloc(MAX\_LENGTH, sizeof(char));  if (str == NULL)  {  perror("calloc error");  return -1;  }  while (1)  {  CHECK\_ERROR(sem\_wait(sem1), "sem\_wait error");  int n = 0;  memcpy(&n, in, sizeof(int));  if (n == 0)  break;  memcpy(str, in + sizeof(int), n);  int p = (n - 2 > 0) ? n - 2 : 0;  if (str[p] != '.' && str[p] != ';')  {  char err[] = "Last symbol is \'.\'\n";  int k = strlen(err);  err[k - 3] = str[p];  memcpy(ans, &k, sizeof(int));  memcpy(ans + sizeof(int), err, k);  }  else  {  memcpy(ans, &zero, sizeof(int));  CHECK\_ERROR(write(fout, in + sizeof(int), n), "write error");  }  CHECK\_ERROR(sem\_post(sem2), "sem\_post error");  }  CHECK\_ERROR(munmap(in, SIZE), "munmap error");  CHECK\_ERROR(munmap(ans, SIZE), "munmap error");  CHECK\_ERROR(close(fout), "close error");  free(str);  }  // parent  else  {  void\* out = mmap(NULL, sizeof(int), PROT\_WRITE, MAP\_SHARED, file1, 0);  void\* ans = mmap(NULL, sizeof(int), PROT\_READ, MAP\_SHARED, file2, 0);  if (out == MAP\_FAILED || ans == MAP\_FAILED)  {  perror("mmap error");  return -1;  }  char\* err = calloc(MAX\_LENGTH, sizeof(char));  if (err == NULL)  {  perror("calloc error");  return -1;  }  char\* str;  size\_t s = 0;  int n = getline(&str, &s, stdin);  while (n > 0)  {  memcpy(out, &n, sizeof(int));  memcpy(out + sizeof(int), str, n);  CHECK\_ERROR(sem\_post(sem1), "sem\_post error");  CHECK\_ERROR(sem\_wait(sem2), "sem\_wait error");  int k;  memcpy(&k, ans, sizeof(int));  if (k != 0)  {  memcpy(err, ans + sizeof(int), k);  printf("%s", err);  }  n = getline(&str, &s, stdin);  }  memcpy(out, &n, sizeof(int));  CHECK\_ERROR(sem\_post(sem1), "sem\_post error");  CHECK\_ERROR(munmap(out, SIZE), "munmap error");  CHECK\_ERROR(munmap(ans, SIZE), "munmap error");  free(err);  }  CHECK\_ERROR(sem\_close(sem1), "sem\_close error");  CHECK\_ERROR(sem\_close(sem2), "sem\_close error");  CHECK\_ERROR(close(file1), "close error");  CHECK\_ERROR(close(file2), "close error");  CHECK\_ERROR(unlink("file1"), "unlink error");  CHECK\_ERROR(unlink("file2"), "unlink error");  } |

**Демонстрация работы программы**

liguha@Laptop:~/OS/LR4/build$ cat result.txt

cat: result.txt: Нет такого файла или каталога

liguha@Laptop:~/OS/LR4/build$ ./a.out

abcde

Last symbol is 'e'

acb def g

Last symbol is 'g'

abcd.

123 456;

.

;

Last symbol is ' '

Last symbol is '

'

liguha@Laptop:~/OS/LR4/build$ cat result.txt

abcd.

123 456;

.

;

**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».