Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 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

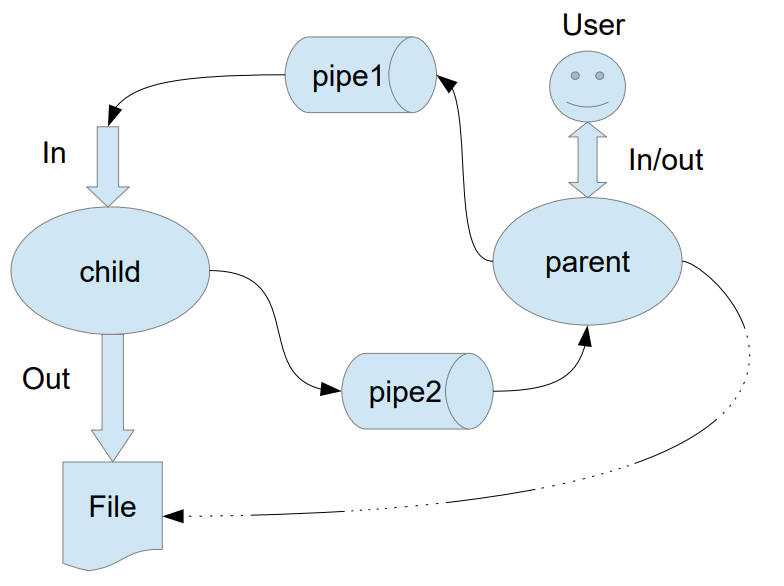
**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

 Управление процессами в ОС  
 Обеспечение обмена данных между процессами посредством каналов

**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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода. Правило проверки: строка должна оканчиваться на «.» или «;».

**Общие сведения о программе**

Программа родительского процесса компилируется из parent.c, использует заголовочные файлы stdio.h, unistd.h, sys/stat.h, fcntl.h. Программа дочернего процесса компилируется из child.c, использует заголовочные файлы stdio.h, unistd.h, sys/stat.h, fcntl.h. В программах используются следующие системные вызовы:

1. mkfifo() – создание именованного канала
2. unlink() – удаление имени из файловой системы
3. fork() – создание дочернего процесса
4. open() – открытие файла
5. close() – закрытие файла
6. write() – запись последовательности байт
7. read() – чтение последовательности байт
8. execl() – замена образа памяти процесса
9. dup2() – переназначение файлового дескриптора

**Общий метод и алгоритм решения**

Родительский процесс получает имя файла, после чего создаётся дочерний процесс, при вызове execl() полученное имя файла передаётся в дочерний процесс в качестве аргументов командной строки. После того как оба процесса открыли каналы, они входят в циклы, условие выхода из которых – конец ввода. Родительский процесс передаёт введённую строку в дочерний, после чего ждёт ответа от дочернего, первый байт в последовательности ответа – результат проверки (0 – строка не подходит, 1 – строка подходит), в случае получения 0 читается ещё 19 байт – сообщение об ошибке.

**Исходный код**

|  |
| --- |
| **parent.c** |
| #include "unistd.h"  #include "stdio.h"  #include "sys/stat.h"  #include "fcntl.h"  int main(int argc, char\* argv[])  {  unlink("pipe1");  unlink("pipe2");  if (mkfifo("pipe1", S\_IREAD | S\_IWRITE) == -1 || mkfifo("pipe2", S\_IREAD | S\_IWRITE) == -1)  {  perror("Parent: pipe create error");  return -1;  }  char\* fout;  size\_t k = 0;  if (getline(&fout, &k, stdin) <= 0)  {  perror("Parent: file name error");  return -1;  }  int id = fork();  if (id == -1)  {  perror("Parent: fork error");  return -1;  }  if (id == 0)  {  int p1 = open("pipe1", O\_WRONLY);  int p2 = open("pipe2", O\_RDONLY);  if (p1 == -1 || p2 == -1)  {  perror("Parent: pipe open error");  return -1;  }  char\* str;  size\_t n = 0;  int s = getline(&str, &n, stdin);  while (s > 0)  {  if (write(p1, str, s) == -1)  {  perror("Parent: write error");  return -1;  }  char ok;  if (read(p2, &ok, 1) <= 0)  {  perror("Parent: read error");  return -1;  }  if (ok == '0')  {  char ans[19];  if (read(p2, ans, 19) <= 0)  {  perror("Parent: read error");  return -1;  }  printf("%s\n", ans);  }  s = getline(&str, &n, stdin);  }  close(p1);  close(p2);  unlink("pipe1");  unlink("pipe2");  }  else  {  if (execl("child.out", fout, NULL) == -1)  {  perror("Child: exec error");  return -1;  }  }  } |

|  |
| --- |
| **child.c** |
| #include "unistd.h"  #include "stdio.h"  #include "sys/stat.h"  #include "fcntl.h"  int main(int argc, char\* argv[])  {  int p1 = open("pipe1", O\_RDONLY);  int p2 = open("pipe2", O\_WRONLY);  if (p1 == -1 || p2 == -1)  {  perror("Child: pipe open error");  return -1;  }  unlink(argv[0]);  int fout = open(argv[0], O\_CREAT | O\_WRONLY, S\_IREAD);  if (fout == -1)  {  perror("Child: file error");  return -1;  }  if (dup2(p1, 0) == -1 || dup2(fout, 1) == -1)  {  perror("Child: dup error");  return -1;  }  char\* str;  size\_t n = 0;  int s = getline(&str, &n, stdin);  char err[19] = "Last symbol is \'.\'";  char ok[3] = "01";  while (s > 0)  {  if (str[s - 2] == '.' || str[s - 2] == ';')  {  printf("%s", str);  if (write(p2, &ok[1], 1) == -1)  {  perror("Child: write error");  return -1;  }  }  else  {  err[16] = str[s - 2];  if (write(p2, &ok[0], 1) == -1 || write(p2, err, 19) == -1)  {  perror("Child: write error");  return -1;  }  }  s = getline(&str, &n, stdin);  }  close(p1);  close(p2);  } |

**Демонстрация работы программы**

liguha@Laptop:~/OS/LR1/build$ ls

child.out CMakeFiles Makefile

CMakeCache.txt cmake\_install.cmake parent.out

liguha@Laptop:~/OS/LR1/build$ ./parent.out

out\_file.txt

first string

Last symbol is 'g'

Second string.

string number three;

another string without '.' or ';' in the end

Last symbol is 'd'

string with point.

last string in the test

Last symbol is 't'

liguha@Laptop:~/OS/LR1/build$ ls

child.out CMakeFiles Makefile parent.out

CMakeCache.txt cmake\_install.cmake out\_file.txt

liguha@Laptop:~/OS/LR1/build$ cat out\_file.txt

Second string.

string number three;

string with point.

**Выводы**

Составлена и отлажена программа на языке Си, осуществляющая работу с процессами. Тем самым, приобретены навыки в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов.