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: Путилин Дмитрий

Группа: М80-207Б-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/putilin21dn/OC

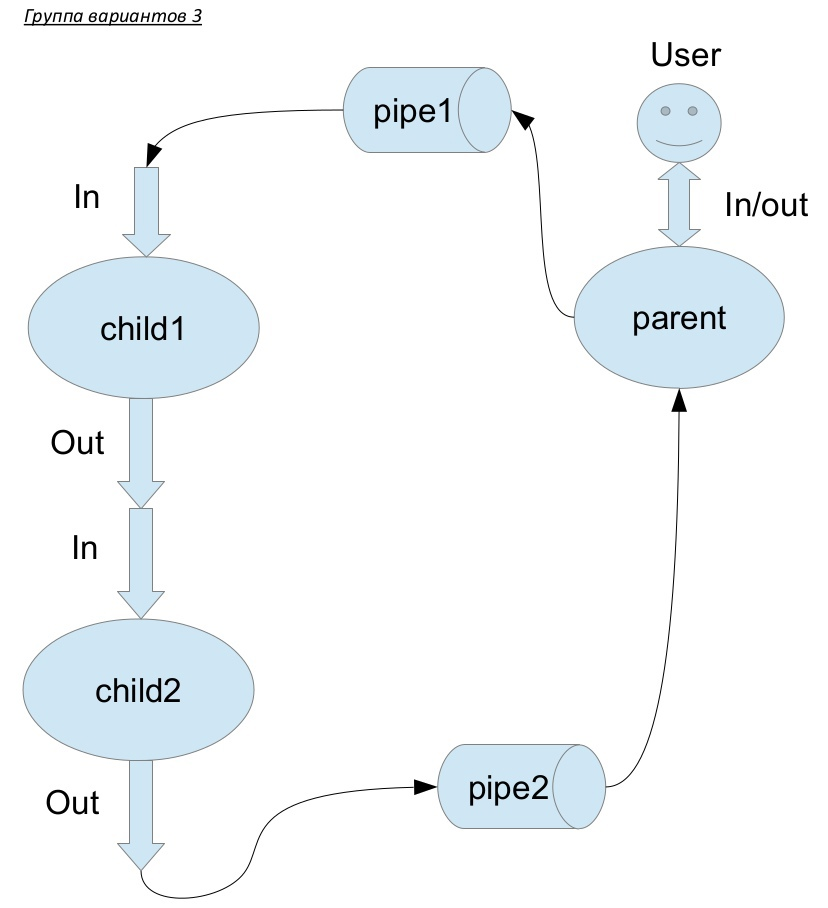
**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

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

**Задание**



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.

Child1 переводит строки в нижний регистр. Child2 превращает все пробельные

символы в символ «\_».

**Общие сведения о программе**

Программа компилируется из файла parent.cpp. Также используется заголовочные файлы: iostream, unistd.h, stdio.h, stdlib.h, string.h. В программе используются следующие системные вызовы:

1. pipe() - создание однонаправленного канала
2. fork() – создание дочернего процесса
3. dup2() – переназначение файлового дескриптора
4. execl() – замена образа памяти процесса
5. write() – запись последовательности байт
6. read() – чтение последовательности байт
7. close() - закрытие канала

**Общий метод и алгоритм решения**

Родительский процесс получает строки, которые считываются посимвольно с помощью getchar и записываются в массив символов. Создаются два дочерних процесса ( для удобства объединены в одну программу). При вызове execl() данные с названием дочерней программы и созданные fd- стандартные потоки ввода-вывода передаются в дочерний процесс, как аргументы, причем каждый из fd[i] переводятся предварительно в строку. После этого в дочернем процессе выполняются изменения над строками, после которых происходит запись в нужный fd. После выполнения работы дочернего процесса действие возвращается в родительский и там происходит вывод результата, при этом при каждом записи или считывании происходит проверка.

**Исходный код**

|  |
| --- |
| **parent.c** |
| #include <iostream>  #include "unistd.h"  #include "stdio.h"  #include "stdlib.h"  #include "string.h"  int main(){  int error = 0;  int fd[2];  int fd1[2];  int fd2[2];  pipe(fd); // pipe between child process  pipe(fd1); // pipe between child1 process and parent process  pipe(fd2); // pipe between child2 process and parent process    int pid1 = fork();    if(pid1==-1){  printf("FORK ERROR!\n");  return -1;  }  if(pid1==0){  // CHILD    //printf("%d %d %d %d %d %d\n", fd[0],fd[1], fd1[0],fd1[1], fd2[0],fd2[1]);    error = dup2(fd1[0],1);  if(error==-1){  printf("Child: dup error\n");  return -1;  }  std :: string str10 = std :: to\_string(fd1[0]);  std :: string str11 = std :: to\_string(fd1[1]);  std :: string str20 = std :: to\_string(fd2[0]);  std :: string str21 = std :: to\_string(fd2[1]);  std :: string str0 = std :: to\_string(fd[0]);  std :: string str1 = std :: to\_string(fd[1]);  execl("child.out", str10.c\_str(),str11.c\_str(),str20.c\_str(),str21.c\_str(), str0.c\_str(), str1.c\_str(), NULL);    }  else if(pid1>0){    // PARENT  printf("[%d] It's parent. Child id: %d\n", getpid(), pid1);  char \*in = (char\*)malloc(sizeof(char)\*2);  in[0] = 0;  char c;    while((c=getchar())!=EOF){    in[0] += 1;  in[in[0]] = c;  in = (char\*)realloc(in, (in[0]+2)\*sizeof(char));  }  in[in[0]] = '\0';  error = write(fd1[1], in, (in[0]+2)\*sizeof(char));  if (error==-1){  printf("Not string! \n");  return -1;  }  char \*out = (char\*)malloc(sizeof(char));  error = read(fd2[0], &out[0],sizeof(char));  if(error==-1){  printf("WARNING! NOT READ TEXT!\n");  return -1;  }  out = (char\*)realloc(out, (out[0]+2)\*sizeof(char));  for(int i=1; i<out[0]+1; ++i){  error = read(fd2[0],&out[i], sizeof(char));  if(error==-1){  printf("WARNING! Not read liter!\n");  return -1;  }  else{  printf("%c",out[i]);  }  }  printf("\n");    close(fd1[1]);  close(fd2[0]);    free(in);  free(out);    }    } |

|  |
| --- |
| **child.c** |
| #include "unistd.h"  #include "stdio.h"  #include "stdlib.h"  #include "string.h"  int main(int argc, char \*argv[])  {  int error = 0;  printf("[%d] It's child\n", getpid());  int fd1[2];  int fd2[2];  int fd[2];    fd1[0] = atoi(argv[0]);  fd1[1] = atoi(argv[1]);  fd2[0] = atoi(argv[2]);  fd2[1] = atoi(argv[3]);  fd[0] = atoi(argv[4]);  fd[1] = atoi(argv[5]);  // CHILD1  //printf("%d %d\n",fd1[0],fd1[1]);  char\* in = (char\*)malloc(sizeof(char)\*2);  error = read(fd1[0], &in[0], sizeof(char));  //printf("%d\n",in[0]);  if(error==-1){  printf("WARNING! NOT READ STRING!\n");  return -1;  }  in = (char\*)realloc(in, (in[0]+2)\*sizeof(char));  for(int i=1; i<in[0]+1; ++i){  error = read(fd1[0],&in[i], sizeof(char));  if(error==-1){  printf("WARNING! Not read liter!\n");  return -1;  }  }  char \*out = (char\*)malloc(sizeof(char)\*2);  out[0] = 0;  for(int i=1; i<in[0]+1;++i){  if(in[i]>='A' && in[i]<='Z'){  out[0]+=1;  out[out[0]] = in[i] + 32;  }  else{  out[0]+=1;  out[out[0]] = in[i];  }  out = (char\*)realloc(out, (out[0]+2)\*sizeof(char));  }  out[0]++;  out[out[0]] = '\0';  write(fd[1],out,sizeof(char)\*(out[0]+2));    free(in);  free(out);\  error = dup2(fd[1],0);  if(error==-1){  perror("Child: dup error\n");  return -1;  }  error = dup2(fd[0],1);  if(error==-1){  perror("Child: dup error\n");  return -1;  }  close(fd1[0]);  close(fd[1]);  // CHILD2  in = (char\*)malloc(sizeof(char)\*2);  error = read(fd[0], &in[0], sizeof(char));    if(error==-1){  printf("WARNING! NOT READ STRING!\n");  return -1;  }    in = (char\*)realloc(in, (in[0]+2)\*sizeof(char));  for(int i=1; i<in[0]+1; ++i){  error = read(fd[0],&in[i], sizeof(char));  if(error==-1){  printf("WARNING! Not read liter!\n");  return -1;  }  }  out = (char\*)malloc(sizeof(char)\*2);  out[0] = 0;  for(int i=1; i<in[0]+1;++i){  if(in[i]==' '){  out[0]+=1;  out[out[0]] = '\_';  }  else{  out[0]+=1;  out[out[0]] = in[i];  }  out = (char\*)realloc(out, (out[0]+2)\*sizeof(char));  }  out[0]++;  out[out[0]] = '\0';  error = write(fd2[1], out, (out[0]+2)\*sizeof(char));  if(error==-1){  printf("WARRING! Problem write!\n");  }  free(in);  free(out);      error = dup2(fd2[1],0);  if(error==-1){  perror("Child: dup error\n");  return -1;  }  close(fd2[1]);  close(fd[0]);  } |

**Демонстрация работы программы**

dmitry@dmitry-VirtualBox:~/Рабочий стол/OC/lab2/build$ ls

child.out CMakeFiles Makefile test1

CMakeCache.txt cmake\_install.cmake parent.out test2

dmitry@dmitry-VirtualBox:~/Рабочий стол/OC/lab2/build$ ./parent.out < test1

[31575] It's parent. Child id: 31576

my\_name\_is\_dima!

i\_am\_student\_\_\_\_!

i\_a\_m\_1\_9\_a\_g\_e\_!

dmitry@dmitry-VirtualBox:~/Рабочий стол/OC/lab2/build$ ./parent.out < test2

[31633] It's parent. Child id: 31634

it's\_my\_first\_laba\_

in\_this\_study\_year!

on\_process\_!\_!\_!\_!\_!

**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.