Отчёт по лабораторной работе №2 по курсу Операционные системы.

Студент группы 207Б Ляшун Д.С., № по списку 14

Контакты www, e-mail, icq, skype gabn37@gmail.com

Работа выполнена: «18» октября 2020 г.

Преподаватель: каф.806 Миронов Е.С

Входной контроль знаний с оценкой _____

Отчёт сдан « » _____20 __ г., итоговая оценка _____

Подпись преподавателя _____

1. **Тема:** Процессы. Каналы.

2. **Цель работы:** Приобретение практических навыков в: управлении процессами в ОС; обеспечении обмена данных между процессами посредством каналов.

3. **Задание** (*вариант №* **10**): Родительский процесс создает дочерний процесс. Пользователь в консоль родительского процесса вводит имя файла, который будет открыт на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода.
Задача: в файле записаны команды вида: «число». Д.п. проверяет эти числа на простоту. Если число составное, то д.п. пишет это число в parent, если же отрицательное или простое, то тогда д.п. и р.п. завершаются.

4. **Оборудование(лабораторное)**:
ЭВМ _____, процессор _____, имя узла сети_____ с ОП _____ Мб,
НМД_____Мб. Терминал_____адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel Core i3-5005u с ОП 6000 Мб, НМД 562000 Мб. Монитор Acer (ноутбук)
Другие устройства _____

5. **Программное обеспечение(лабораторное):**
Операционная система семейства _____, наименование _____ версия _____
интерпретатор команд _____ версия _____
Система программирования _____ версия _____
Редактор текстов _____ версия _____
Утилиты операционной системы_____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства UNIX, наименование Ubuntu версия 16.4
интерпретатор команд bash версия _____.
Система программирования GNU C++ версия _____
Редактор текстов emacs версия _____
Утилиты операционной системы_____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных на домашнем компьютере_____

6. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

При написании лабораторной работы были использованы следующие системные вызовы:
FILE fopen(const char* path, const char* mode)* – функция, открывающая файл с именем path в соответствующем режиме mode. В результате работы возвращается указатель на открытый файл.
int pipe(int pipefd[2]) – функция, создающая однонаправленный канал данных, который можно использовать для взаимодействия между процессами. Массив pipefd заполняется файловыми дескрипторами, указывающие на разные концы канала: pipefd[0] – для записи данных, pipefd[1] – для чтения данных. В результате работы возвращается 0, если функция отработала успешно, иначе -1.
int fork() – функция, создающая новый процесс (дочерний) посредством копирования процесса, в котором она была вызвана (родительский). В результате работы функции родительский процесс получает числовой pid созданного дочернего процесса, в то время как дочернему возвращается 0. Также при ошибке в создании процесса родительский процесс получит pid равный -1.
int getpid() – функция, возвращающая идентификатор вызвавшего процесса.
int close(int fd) – функция, закрывающая файловый дескриптор, который после этого не ссылается не на один файл и может быть использован повторно. При успешном выполнении возвращается 0, а в случае ошибки -1.
int dup2(int fd1, int fd2) – функция, связывающая файловые дескрипторы fd1 и fd2 так, что fd2 станет обозначать fd1. При успешном выполнении функции возвращается 0, а в случае ошибки -1.

7. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

int execv(const char path, char* const argv[])* – функция, заменяющая текущий образ процесса новым с именем path и набором аргументов argv. В результате работы возвращается значение -1 в случае возникновения ошибки.
ssize_t write(int fd, const void* buf, size_t count) - записывает в бинарном виде до count байт из буфера, на который указывает buf, в файл, на который указывает дескриптор fd. Функция возвращает количество успешно записанных байт.
ssize_t read(int fd, void buf, size_t count*) – пытается прочитать в бинарном виде count байт из файла с файловым дескриптором fd и записать в буфер, начинающийся по адресу buf. При успешном выполнении функция возвращает количество успешно прочитанных байт.
void perror(const char* s) – выдает сообщение о стандартной ошибке, обнаруженной во время вызова системной или библиотечной функции.

*Пункты 1-7 отчета составляются **строго до** начала лабораторной работы.*

Допущен к выполнению работы. Подпись преподавателя _____

Исходный код программ ЛР №2.

Содержимое файла main.c

```
#include "unistd.h"
#include "stdio.h"
int main()
{
     char input[256];
     printf("Enter file name: ");
     scanf("%s", input);
     FILE* fp;
     if ((fp = fopen(input, "r")) == NULL)
     {
          perror("Error! Input file isn't opened!");
          return -1;
     }
     int fd[2];
     if (pipe(fd) == -1)
     {
         perror("Error! Pipe isn't created!");
         return -2;
     }
     int id = fork();
     if (id == -1)
     {
         perror("Fork error!");
         return -3;
     }
     if (id == 0)
     {
          printf("[%d] It's child process\n", getpid());
          close(fd[0]);
          char* arg[] = {"child", NULL};
          if (dup2(fd[1], 1) == -1)
          {
              perror("Error creating duplicate file descriptor for pipe
output!\n");
              return -4;
          }
          if (dup2(fileno(fp), 0) == -1)
         {
              perror("Error creating duplicate file descriptor for input
file!\n");
              return -5;
         }
         if (execv("child", arg) == -1)
         {
              perror("Error when starting a child for execution!");
              return -6;
         }
     }
     else
     {
```

```
            printf("[%d] It's parent process of %d\n", getpid(), id);
            close(fd[1]);
            int res;
            while (read(fd[0], &res, sizeof(int)) != 0)
            {
                printf("%d is composit number\n", res);
            }
        }
        return 0;
}
```

Содержимое файла child.c

```
#include "stdio.h"
#include "unistd.h"
int main()
{
        int num;
        while (scanf("%d", &num) > 0)
        {
                int is_prime = 1;
                if (num < 0) break;
                for (int i = 2; i * i <= num; ++i)
                {
                        if (num % i == 0)
                        {
                                is_prime = 0;
                                break;
                        }
                }
                if (is_prime == 0) write(1, &num, sizeof(int));
                else break;
        }
}
```

8. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

```
dmitry@dmitry-VirtualBox:~$ cat head.txt
**********************************
▌      Лабораторная работа №2      ▐
▌    по теме Процессы. Каналы      ▐
▌    выполнил студент группы 207Б  ▐
▌           Ляшун Дмитрий          ▐
**********************************
dmitry@dmitry-VirtualBox:~$ cd Work_place/OS_labs/Lab2
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ ls
child  child.c  head.txt  main  main.c  tests1.txt tests2.txt
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ cat main.c
#include "unistd.h"
#include "stdio.h"
int main()
{
     char input[256];
     printf("Enter file name: ");
     scanf("%s", input);
     FILE* fp;
     if ((fp = fopen(input, "r")) == NULL)
     {
       perror("Error! Input file isn't opened!");
       return -1;
     }
     int fd[2];
     if (pipe(fd) == -1)
     {
         perror("Error! Pipe isn't created!");
         return -2;
     }
     int id = fork();
     if (id == -1)
     {
         perror("Fork error!");
         return -3;
     }
     if (id == 0)
     {
       printf("[%d] It's child process\n", getpid());
       close(fd[0]);
       char* arg[] = {"child", NULL};
       if (dup2(fd[1], 1) == -1)
          {
              perror("Error creating duplicate file descriptor for pipe
output!\n");
              return -4;
          }
          if (dup2(fileno(fp), 0) == -1)
       {
           perror("Error creating duplicate file descriptor for input
file!\n");
           return -5;
       }
       if (execv("child", arg) == -1)
```

```
       {
          perror("Error when starting a child for execution!");
          return -6;
       }
      }
      else
      {
          printf("[%d] It's parent process of %d\n", getpid(), id);
          close(fd[1]);
          int res;
          while (read(fd[0], &res, sizeof(int)) != 0)
          {
              printf("%d is composit number\n", res);
          }
      }
      return 0;
}

dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ cat child.c
 #include "stdio.h"
 #include "unistd.h"
 int main()
 {
    int num;
    while (scanf("%d", &num) > 0)
    {
      int is_prime = 1;
      if (num < 0) break;
      for (int i = 2; i * i <= num; ++i)
      {
           if (num % i == 0)
           {
              is_prime = 0;
              break;
           }
      }
        if (is_prime == 0) write(1, &num, sizeof(int));
      else break;
    }
 }
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ cat tests1.txt
10
20
30
40
33
8
10
20
30
40
33
8
10
20
```

```
30
40
33
5
2
8
-10
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ cat tests2.txt
18
20
22
100000
200000
300000
400000
121
234
333
222
91312
1112
11
222
333
-10
25
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ gcc main.c -o main
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ gcc child.c -o child
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ ./main
Enter file name: tests1.txt
[4177] It's parent process of 4178
[4178] It's child process
10 is composit number
20 is composit number
30 is composit number
40 is composit number
33 is composit number
8 is composit number
10 is composit number
20 is composit number
30 is composit number
40 is composit number
33 is composit number
8 is composit number
10 is composit number
20 is composit number
30 is composit number
40 is composit number
33 is composit number
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$ ./main
Enter file name: tests2.txt
[4180] It's parent process of 4181
[4181] It's child process
18 is composit number
20 is composit number
```

```
22 is composit number
100000 is composit number
200000 is composit number
300000 is composit number
400000 is composit number
121 is composit number
234 is composit number
333 is composit number
222 is composit number
91312 is composit number
1112 is composit number
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab2$
```

9. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

№	Лаб. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание
1.	Дом.	15.10	19:20	В результате работы main не происходит вывод составных чисел.	В дочернем процессе с помощью dup2 необходимо переназначить стандартный поток вывода и канал на ввод, чтобы родительский процесс мог считать составные числа.	

10. **Замечания автора** по существу работы

11. **Выводы**

В результате выполнения лабораторной работы я разобрался с использованием каналов, помогающих наладить межпроцессорное взаимодействие, научился переопределять потоки ввода/вывода, работать с файловыми дескрипторами и системными вызовами для управления ими.
В ходе выполнения лабораторной мне пришлось столкнуться с проблемой отладки многопроцессорных программ, в частности налаживания правильной связи с помощью каналов, корректного чтения и записи (я узнал, что функции read и write работают только с данными в двоичном виде, чего не было явно указано в переводе man на Linux, и поэтому послужило источником проблемы при выполнении программы).
Как мне показалось, в случае работы с реальными многопроцессорными приложениями их отладка станет довольно сложным и трудоемким процессом.

Недочёты при выполнении задания могут быть устранены следующим образом:

Подпись студента Ляшун Д.С.