Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Операционные системы»

Студент:	Марков А.Н.
Группа:	М80-208Б-18
Преподаватель:	Миронов Е.С.
Оценка:	
Дата:	

Москва
2019

Содержание

1. Постановка задачи.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними. Родительский процесс отвечает за ввод и вывод. Дочерний процесс осуществляет поиск образца в строке.

2. Общие сведения о программе.

Исходный код хранится в файле bm.c. В данном файле используются заголовочные файлы stdio.h, stdlib.h, string.h, sys/types.h, sys/wait.h, unistd.h, limits.h. В программе используются следующие системные вызовы:

1. pipe — создание канала для обмена данными между процессами. Системный вызов возвращает два дескриптора. Один для записи в канал, другой для чтения из канала.
2. fork — создание дочернего процесса.
3. read — чтение из потока в буфер некоторого количества байт.
4. write — запись в поток из буфера некоторого количества байт.
5. wait — ожидание завершения дочернего процесса.
6. close — закрытие потока.

3. Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Используя системный вызов pipe, создать два канала для общения между процессами.
2. С помощью системного вызова fork создать дочерний процесс.
3. Посимвольно считывать из стандартного потока ввода строку, в которой будет происходить поиск образца, с помощью системного вызова read. Записать результат в первый канал с помощью системного вызова write.Одновременно со считыванием нужно считать количество считанных символов, а затем это число записать во второй канал.
4. Повторить пункт 3 только для считывания образца.
5. Затем родительский процесс вступает в состояние ожидания завершения дочернего процесса.
6. Как только родительский процесс записал данные в первый канал дочерний процесс считывает их, производит поиск образца в строке и записывает результат во первый канал.
7. Как только дочерний процесс завершился, родительский считывает результат из первого канала и выводит в стандартный поток вывода.

4. Основные файлы программы.

Файл src.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <limits.h>

int bmh(char *str, int str_len, char *pattern, int pat_len) {
   int table[CHAR_MAX + 1];

   if (str_len < pat_len || pat_len <= 0 || !str || !pattern) {
      return -1;
   }

   for (register int i = 0; i < CHAR_MAX + 1; ++i) {
      table[i] = pat_len;
   }

   for (register int i = 1; i < pat_len; ++i) {
      if (table[(int) pattern[pat_len - i - 1]] != pat_len) {
         continue;
      }
      else {
         table[(int) pattern[pat_len - i - 1]] = i;
      }
   }
   for (register int i = 0; i < str_len; ++i) {
      int match = 0;
      for (register int j = pat_len - 1; j >= 0; --j) {
         if (str[i + j] != pattern[j] && !match) {
            i += table[(int) str[i + j]] - 1;
            break;
         }
         else if(str[i + j] != pattern[j] && match) {
            i += table[(int) pattern[pat_len - 1]] - 1;
            match = 0;
            break;
         }
         else {
```

```
            match = 1;
        }
      }
      if (match) {
        while (i != 0 && str[i - 1] != ' ') {
          i--;
        }
        return i;
      }
    }
    return -1;
}

int main() {
    char temp_ch;
    int fd1[2], fd2[2], temp_fork, cnt = 0;
    int read_size, write_size;

    if (pipe(fd1) == -1 || pipe(fd2) == -1) {
        printf("Can\'t create pipe");
        exit(-1);
    }

    temp_fork = fork();
    if (temp_fork == -1) {
        printf("Can\'t create child");
        exit(-1);
    }
    else if (temp_fork > 0) {
        close(fd1[0]);
        int out;
        while ((out = read(2, &temp_ch, sizeof(char))) != 0) {
            if (out == -1)
            {
                exit(1);
            }
            write_size = write(fd1[1], &temp_ch, sizeof(char));
            if (write_size != sizeof(char)) {
                printf("Can\'t write char\n");
                exit(-1);
            }
            cnt++;
        }
```

```
    write_size = write(fd2[1], &cnt, sizeof(int));
    if (write_size != sizeof(int)) {
        printf("Can\'t write size of string\n");
        exit(-1);
    }
    cnt = read_size = write_size = 0;

    while ((out = read(2, &temp_ch, sizeof(char))) != 0 ) {
        if (out == -1)
        {
            exit(1);
        }
        write_size = write(fd1[1], &temp_ch, sizeof(char));
        if (write_size != sizeof(char)) {
            printf("Can\'t write char\n");
            exit(-1);
        }
        cnt++;
    }
    write_size = write(fd2[1], &cnt, sizeof(int));
    if (write_size != sizeof(int)) {
        printf("Can\'t write size of pattern\n");
        exit(-1);
    }
    cnt = read_size = 0;

    wait(NULL);

    read_size = read(fd2[0], &cnt, sizeof(int));
    if (read_size != sizeof(int)) {
        printf("Can\'t read result");
        exit(-1);
    }
    printf("%d\n", cnt);
    close(fd1[1]);
    close(fd2[0]);
    close(fd2[1]);
}
else {
    close(fd1[1]);
    int str_size, pat_size, result;

    read_size = read(fd2[0], &str_size, sizeof(int));
    if (read_size != sizeof(int)) {
```

```
            printf("Can\'t read size of string");
            exit(-1);
        }
        char *str = (char *) malloc(sizeof(char *) * str_size);
        read_size = read(fd1[0], str, sizeof(char) * str_size);
        if (read_size != sizeof(char) * str_size) {
            printf("Can\'t read string\n");
            exit(-1);
        }

        read_size = read(fd2[0], &pat_size, sizeof(int));
        if (read_size != sizeof(int)) {
            printf("Can\'t read size of pattern");
            exit(-1);
        }
        char *pattern = (char *) malloc(sizeof(char *) * pat_size);
        read_size = read(fd1[0], pattern, sizeof(char) * pat_size);
        if (read_size != sizeof(char) * pat_size) {
            printf("Can\'t read pattern\n");
            exit(-1);
        }

        result = bmh(str, str_size - 1, pattern, pat_size - 1);
        write_size = write(fd2[1], &result, sizeof(int));
        if (write_size != sizeof(int)) {
            printf("Can\'t write result\n");
            exit(-1);
        }
        close(fd1[0]);
        close(fd2[0]);
        close(fd2[1]);
    }

    return 0;
}
```

5. Демонстрация работы программы.

```
oem@Alex-PC:~/Documents/OS/lab2$ ./bm
kol kolokol
kolok
4
oem@Alex-PC:~/Documents/OS/lab2$ ./bm
kol kolokol
kol
0
oem@Alex-PC:~/Documents/OS/lab2$ ./bm
kasha gerkules
kul
6
oem@Alex-PC:~/Documents/OS/lab2$ ./bm
kasha gerkules
ha
0
```

6. Вывод.

Процессы — это одна из самых старых и наиболее важных абстракций, присущих операционной системе. Они поддерживают возможность осуществления (псевдо) параллельных операций даже при наличии всего одного процессора. Они превращают один центральный процессор в несколько виртуальных. Без абстракции процессоров современные вычисления просто не могут существовать. Межпроцессное взаимодействие можно осуществлять с помощью канала. В системах UNIX канал создается с помощью системного вызова pipe. Я считаю, что такой подход к общению процессов удобен, поскольку при использовании блокирующих вызовов read и write процессы блокируются, если им нечего считывать или буфер для записи полный. Также одним из плюсов такого способа общения процессов является то, что каналом могут пользоваться только родственные процессы.