Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени
Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»
КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №1

по дисциплине «Разработка программных систем»

Студент:	Турунов Дмитрий Николаевич
Группа:	РК6-63Б
Тип задания:	Лабораторная работа
Тема:	Многопроцессное программирование
Вариант:	4

Студент _____ (подпись, дата) Турунов Д.Н. (Фамилия, И.О.)

Преподаватель _____ (подпись, дата) _____ (Фамилия, И.О.)

Москва, 2024

Содержание

Задание

Составить программу поиска в архиве (библиотеке) имен файлов, содержащих заданный прототип. Имя архива и прототипа должны передаваться программе как параметры. Для просмотра содержания архива использовать команду `ar` в сочетании со средствами перенаправления стандартного ввода-вывода через программный канал. Дополнительно предложить вариант решения данной задачи средствами командного процессора `sh` или `csh`.

Содержание отчета

- Текст задания на лаб. работу
- Описание структуры программы и реализованных способов взаимодействия процессов (с рисунком)
- Описание основных используемых структур данных
- Блок-схема программы согласно ГОСТ и пояснения к ней
- Примеры результатов работы программы
- Текст программы с исчерпывающими комментариями

Описание структуры программы и использованных структур данных

- `pipe()` создает однонаправленный коммуникационный канал, который может быть использован для передачи данных между процессами.

- `fork()` порождает новый процесс, который является копией исходного (родительского) процесса.

- `dup2()` используется для копирования файловых дескрипторов, что позволяет перенаправить ввод и вывод между процессами.

- `execvp()` выполняет программу, указанную в качестве аргумента, заменяя текущий процесс-потомок новым процессом.

- `close()` закрывает файловый дескриптор, освобождая его для других процессов.

- `wait()` заставляет родительский процесс ожидать завершения выполнения его дочерних процессов, позволяя избежать "зомби"процессов.

- `fprintf()` и `perror()` используются для вывода ошибок в случае возникновения исключительных ситуаций.

Процесс работы программы:

1. Программа начинает свою работу с проверки наличия необходимых аргументов командной строки. Она требует указания имени архива и шаблона для поиска в архиве.

2. Создается канал (`pipe`) для передачи данных между двумя процессами. Канал состоит из двух файловых дескрипторов: один для чтения и один для записи.

3. С помощью `fork()` создаются два процесса. Первый процесс вызывает команду `ar` с аргументом `t` для просмотра списка файлов в архиве, второй процесс использует `grep` для фильтрации списка по заданному шаблону.

4. В каждом из созданных процессов перенаправление стандартного ввода и вывода осуществляется с помощью `dup2()`, чтобы обеспечить передачу данных через каналы между процессами. После перенаправления неиспользуемые файловые дескрипторы закрываются.

5. `execvp()` используется в каждом дочернем процессе для запуска соответствующих команд (`ar` и `grep`), при этом текущий дочерний процесс заменяется новым процессом, выполненным командой.

6. Родительский процесс ожидает завершения всех дочерних процессов, используя `wait()`, чтобы корректно завершить работу программы и освободить все ресурсы.

Визуально взаимодействие между процессами можно представить следующим образом (рис. 1):

Рис. 1. Взаимодействие процессов

Блок-схема

На рисунке 2 представлена блок-схема программы.

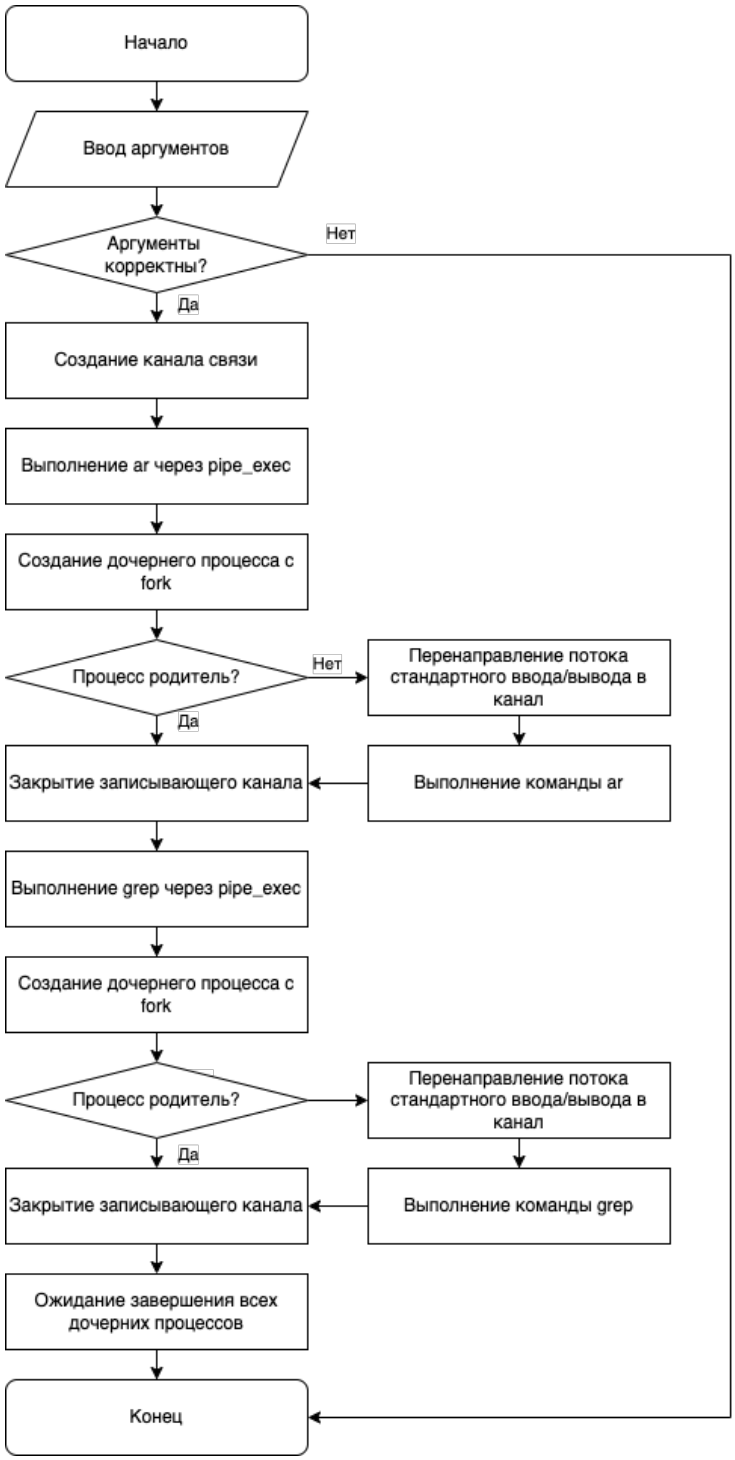

Рис. 2. Блок-схема алгоритма работы программы

Примеры работы программы

На рисунке 3 представлен пример работы программы, когда в нее поданы корректные аргументы. Программа выведет все файлы, содержащие `.txt` в архиве.

```
~/multiproc$ gcc main.c
~/multiproc$ ./a.out example_archive.a .txt
example1.txt
testfile2.txt
sample3.txt
sample4.txt
~/multiproc$ ar -t example_archive.a | grep -e ".txt"
example1.txt
testfile2.txt
sample3.txt
sample4.txt
```

Рис. 3. Пример работы программы и сравнение с средствами командного процессора sh

На рисунке 4 представлен пример работы программы, когда в нее поданы некорректное количество аргументов.

```
~/multiproc$ ./a.out example_archive.a
Usage: ./a.out <archive> "<pattern>"
~/multiproc$ ./a.out
Usage: ./a.out <archive> "<pattern>"
```

Рис. 4. Пример вывода ошибки

Текст программы

На листинге 1 представлен код программы.

```c
#include <stdio.h> // Include standard input and output functions
#include <stdlib.h> // Include standard library for various functions including exit()
#include <sys/wait.h> // Include for wait() system call
#include <unistd.h> // Include for POSIX operating system API, e.g., fork(), pipe()

// Function to execute a command using a pipe for input and/or output
void pipe_exec(char *cmd[], int input_fd, int output_fd) {
  // Create a child process
  if (fork() == 0) {
    // If the input file descriptor is not standard input, duplicate it to standard input and close the original
    if (input_fd != STDIN_FILENO) {
      dup2(input_fd, STDIN_FILENO);
      close(input_fd);
    }
    // If the output file descriptor is not standard output, duplicate it to standard output and close the original
    if (output_fd != STDOUT_FILENO) {
      dup2(output_fd, STDOUT_FILENO);
      close(output_fd);
    }
    // Replace the current process image with a new process image specified by cmd
    execvp(cmd[0], cmd);
    // If execvp returns, it must have failed. Print an error message and exit
    fprintf(stderr, "Error starting %s\n", cmd[0]);
    exit(EXIT_FAILURE);
  }
}

int main(int argc, char *argv[]) {
  // Check if the number of command-line arguments is exactly 3
  if (argc != 3) {
    // Print usage message if not and return with error
    fprintf(stderr, "Usage: %s <archive> \"<pattern>\"\n", argv[0]);
    return 1;
  }

  int pipe1[2]; // Declare an array to hold two file descriptors for one pipe

  // Create a pipe and check for failure
  if (pipe(pipe1) == -1) {
    perror("pipe() failed");
    exit(EXIT_FAILURE);
  }

  // Prepare the command to list contents of an archive file using 'ar'
  char *ar_cmd[] = {"ar", "t", argv[1], NULL};
```

```
46    // Prepare the grep command with a pattern
47    char *grep_cmd[] = {"grep", "-e", argv[2], NULL};
48
49    // Execute the 'ar' command with standard input and output redirected to the
        pipe
50    pipe_exec(ar_cmd, STDIN_FILENO, pipe1[1]);
51    // Close the write-end of the pipe in the parent, as it's no longer needed
52    close(pipe1[1]);
53    // Execute the 'grep' command, taking input from the pipe
54    pipe_exec(grep_cmd, pipe1[0], STDOUT_FILENO);
55    // Close the read-end of the pipe in the parent, as it's no longer needed
56    close(pipe1[0]);
57
58    // Wait for all child processes to terminate
59    while (wait(NULL) > 0);
60
61    // Return 0 to indicate success
62    return 0;
63 }
```

Листинг 1. Программный код с использованием pipe и exec