ИДЗ ПО ОС 1

метаданные

Бабушкин В. А. 237

номер варианта 27

задания в файле task01-core_(0).pdf

Разработать программу, которая определяет количество целых чисел в ASCII–строке.Числа состоят из цифр от 0 до 9.Разделителями являются все другие символы. Вывести не только количество таких чисел, но и сами числа, разделяя их запятыми.

тесты

в каждой папке для удобства запуска тестов лежит bash-script "./run-tests":

```
#!/bin/bash

./a.out ../tests/test1 1
./a.out ../tests/test2 2
./a.out ../tests/test3 3
./a.out ../tests/test4 4
./a.out ../tests/test5 5
```

4 балла

использование

`Usage: ./a.out <from> <to>`

Схема решения

В этом решении я написал

```
int* find_all_numbers(char* string, ssize_t* ans_count)
```

работает примерно как `map(int, input().split())` в питоне. То есть парсит строку на массив чисел. Причем `ans_count` длина получившегося массива. Она занимает так многоо строчек, потому что динамически меняет длину массива чисел, а так же длину последнего считанного числа(позже я понял что это зря, потому что все числа длиннее 12 символов и так упали бы на моменте перевода их в int).

```
char* make_string_from_array(int* arr, ssize_t size)
```

```
```

создает строку формата: "there are 2 numbers: 123, 123"

всего в программе три процесса:

- первый считывает содержимое файла и направляет в неименованный канал, дескриптор которого лежит в переменной `read_chan`
- второй считывает эти данные из канала `read_chan`, обрабатывает благодаря описанным выше функциям и перенаправляет результат в неименованный канал, дескриптор которого лежит в переменной `write_chan`
- третий считывает эти данные из канала `write_chan` и выводит в файл.

тестирование

1.

```
there are 2 numbers: 123, 123
```

2.

```
there are 0 numbers:
```

3.

```
there are 4 numbers: 12345, 4556, 5, 2
```

4. (The Beatles - 8 Days a Week)

```
there are 11 numbers: 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 8
```

5. (Holy Bible)

Этот файл больше 5000 символов, но согласно логике программы он считает первые 5000 символов

```
there are 72 numbers: 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1,
9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19,
1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1,
30, 1, 31, 2, 1, 2, 2, 2, 3, 2, 4, 2, 5
```

5 баллов

использование

```
Usage: ./a.out <from> <to> <chan1> <chan2>
```

Схема решения

В этом решении я также использую

```
int* find_all_numbers(char* string, ssize_t* ans_count)
```

работает примерно как `map(int, input().split())` в питоне. То есть парсит строку на массив чисел. Причем `ans_count` длина получившегося массива. Она занимает так многоо строчек, потому что динамически меняет длину массива чисел, а так же длину последнего считанного числа(позже я понял что это зря, потому что все числа длиннее 12 символов и так упали бы на моменте перевода их в int).

```
char* make_string_from_array(int* arr, ssize_t size)
```

создает строку формата: "there are 2 numbers: 123, 123"

всего в программе три процесса:

- первый считывает содержимое файла и направляет в именованный канал `readchan`
- второй считывает эти данные из канала `readchan`, обрабатывает благодаря описанным выше функциям и перенаправляет результат в именованный канал `writechan`
- третий считывает эти данные из канала `writechan` и выводит в файл.

тестирование

1.

```
there are 2 numbers: 123, 123
```

2.

```
there are 0 numbers:
```

3.

```
there are 4 numbers: 12345, 4556, 5, 2
```

4. (The Beatles - 8 Days a Week)

```
there are 11 numbers: 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 8
```

5. (Holy Bible)

Этот файл больше 5000 символов, но согласно логике программы он считает первые 5000 символов

```
there are 72 numbers: 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1,
9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19,
1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1,
30, 1, 31, 2, 1, 2, 2, 2, 3, 2, 4, 2, 5
```

6 баллов

использование

```
Usage: ./a.out <from> <to>
```

Схема решения

В этом решении я также использую

```
int* find_all_numbers(char* string, ssize_t* ans_count)
```

работает примерно как `map(int, input().split())` в питоне. То есть парсит строку на массив чисел. Причем `ans_count` длина получившегося массива. Она занимает так многоо строчек, потому что динамически меняет длину массива чисел, а так же длину последнего считанного числа(позже я понял что это зря, потому что все числа длиннее 12 символов и так упали бы на моменте перевода их в int).

```
char* make_string_from_array(int* arr, ssize_t size)
```

создает строку формата: "there are 2 numbers: 123, 123"

всего в программе два процесса:

- первый считывает содержимое файла и направляет в неименованный канал, дескриптор которого лежит в переменной `read_chan`, после чего ждет выполнение своего сына через системный вызов `wait(NULL)`. После завершения работы сына процесс считываеь данные из неименованного канала, дескриптор которого лежит в переменной `write_chan`
- второй считывает эти данные из канала `read_chan`, обрабатывает благодаря описанным выше функциям и перенаправляет результат в неименованный канал, дескриптор которого лежит в переменной `write_chan`

тестирование

1.

```
there are 2 numbers: 123, 123
```

2.

```
there are 0 numbers:
```

3.

```
there are 4 numbers: 12345, 4556, 5, 2
```

4. (The Beatles - 8 Days a Week)

```
there are 11 numbers: 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 8
```

5. (Holy Bible)

Этот файл больше 5000 символов, но согласно логике программы он считает первые 5000 символов

```
there are 72 numbers: 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1,
9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19,
1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1,
30, 1, 31, 2, 1, 2, 2, 2, 3, 2, 4, 2, 5
```

7 баллов

использование

```
Usage: ./a.out <from> <to> <chan1> <chan2>
```

Схема решения

В этом решении я также использую

```
int* find_all_numbers(char* string, ssize_t* ans_count)
```

работает примерно как `map(int, input().split())` в питоне. То есть парсит строку на массив чисел. Причем `ans_count` длина получившегося массива. Она занимает так многоо строчек, потому что динамически меняет длину массива чисел, а так же длину последнего считанного числа(позже я понял что это зря, потому что все числа длиннее 12 символов и так упали бы на моменте перевода их в int).

```
char* make_string_from_array(int* arr, ssize_t size)
```

создает строку формата: "there are 2 numbers: 123, 123"

всего в программе два процесса:

- первый считывает содержимое файла и направляет в именованный канал `readchan`, после чего ждет завершения дочернего процесса, считывает данные из канала `writechan` и выводит в файл.
- второй считывает эти данные из канала `readchan`, обрабатывает благодаря описанным выше функциям и перенаправляет результат в именованный канал `writechan`

тестирование

1.

```
there are 2 numbers: 123, 123
```

2.

```
there are 0 numbers:
```

3.

```
there are 4 numbers: 12345, 4556, 5, 2
```

4. (The Beatles - 8 Days a Week)

```
there are 11 numbers: 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 8
```

5. (Holy Bible)

Этот файл больше 5000 символов, но согласно логике программы он считает первые 5000 символов

```

```

```
there are 72 numbers: 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1,
9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19,
1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1,
30, 1, 31, 2, 1, 2, 2, 2, 3, 2, 4, 2, 5
```

возникшие проблемы

дочерний процесс завершался раньше вызова системного вызова `wait`, поэтому пришлось написать

```
waitpid(pid1, NULL, WNOHANG);
```

8 баллов

использование

- p1:

```
Usage: ./a.out <from> <to> <chan1> <chan2>
```

- p2:

```
Usage: ./a.out <chan1> <chan2>
```

Схема решения

В этом решении я также использую

```
int* find_all_numbers(char* string, ssize_t* ans_count)
```

работает примерно как `map(int, input().split())` в питоне. То есть парсит строку на массив чисел. Причем `ans_count` длина получившегося массива. Она занимает так многоо строчек, потому что динамически меняет длину массива чисел, а так же длину последнего считанного числа(позже я понял что это зря, потому что все числа длиннее 12 символов и так упали бы на моменте перевода их в int).

```
char* make_string_from_array(int* arr, ssize_t size)
```

создает строку формата: "there are 2 numbers: 123, 123"

всего в программе два процесса в отдельных программах:

- первый считывает содержимое файла и направляет в именованный канал `readchan`, после чего считывает данные из канала `writechan` и выводит в файл.

- второй считывает эти данные из канала `readchan`, обрабатывает благодаря описанным выше функциям и перенаправляет результат в именованный канал `writechan`

тестирование

модифицировал `run-tests` вот так:

```
#!/bin/bash

echo "Running test 1..."
./p1 ../tests/test1 1 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 1 completed."

echo "Running test 2..."
./p1 ../tests/test2 2 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 2 completed."

echo "Running test 2..."
./p1 ../tests/test3 3 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 3 completed."

echo "Running test 2..."
./p1 ../tests/test4 4 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 4 completed."

echo "Running test 2..."
./p1 ../tests/test5 5 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 5 completed."
```

1.

```
there are 2 numbers: 123, 123
```

2.

```
there are 0 numbers:
```

3.

```
there are 4 numbers: 12345, 4556, 5, 2
```

4. (The Beatles - 8 Days a Week)

```
there are 11 numbers: 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 8
```

5. (Holy Bible)

Этот файл больше 5000 символов, но согласно логике программы он считает первые 5000 символов

```
there are 72 numbers: 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1,
9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1, 19,
1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1, 27, 1, 28, 1, 29, 1,
30, 1, 31, 2, 1, 2, 2, 2, 3, 2, 4, 2, 5
```

9 баллов

использование

- p1:

```
Usage: ./a.out <from> <to> <chan1> <chan2>
```

- p2:

```
Usage: ./a.out <chan1> <chan2>
```

Схема решения

В этом решении я также использую

```
int* find_all_numbers(char* string, ssize_t* ans_count)
```

работает примерно как `map(int, input().split())` в питоне. То есть парсит строку на массив чисел. Причем `ans_count` длина получившегося массива. Она занимает так многоо строчек, потому что динамически меняет длину массива чисел, а так же длину последнего считанного числа(позже я понял что это зря, потому что все числа длиннее 12 символов и так упали бы на моменте перевода их в int).

```
char* make_string_from_array(int* arr, ssize_t size)
```

создает строку формата: "there are 2 numbers: 123, 123"

всего в программе два процесса в отдельных программах:

- первый считывает содержимое файла буффером в 128 байтов и направляет в именованный канал `readchan`, после чего считывает данные буффером в 128 байтов из канала `writechan` и выводит в файл.
- второй считывает эти данные буффером в 128 байтов из канала `readchan`, обрабатывает благодаря описанным выше функциям и перенаправляет результат буффером в 128 байтов в именованный канал `writechan`

небольшая проблема

Мой вариант требует чтобы я вывел вначале количество встреченных чисел, а после каждое из них. При таком подходе не получится прямо во время считывания ограниченным буффером строить итоговую строку, потому что в моменте не понятно какова будет итоговая длина. Поэтому в `p2.c` я считываю 'чанками' по 128 байтов из канала и добавляю их к `data`, то есть массив одновременно содержащий все данные и записываю после обработки тоже из этого массива чанками по 128 символов.

тестирование

модифицировал `run-tests` вот так:

```
#!/bin/bash

echo "Running test 1..."
./p1 ../tests/test1 1 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 1 completed."

echo "Running test 2..."
./p1 ../tests/test2 2 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 2 completed."

echo "Running test 2..."
./p1 ../tests/test3 3 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 3 completed."
```

```
echo "Running test 2..."
./p1 ../tests/test4 4 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 4 completed."

echo "Running test 2..."
./p1 ../tests/test5 5 chan1 chan2 &
sleep 1
./p2 chan1 chan2 &
wait
echo "Test 5 completed."
```

1.

```
there are 2 numbers: 123, 123
```

2.

```
there are 0 numbers:
```

3.

```
there are 4 numbers: 12345, 4556, 5, 2
```

4. (The Beatles - 8 Days a Week)

```
there are 11 numbers: 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 8
```

5. (Holy Bible)

Этот файл очень большой, (все таки все числа в библии!) поэтому его можно посмотреть здесь

```
there are 68338 numbers: 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8,
1, 9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1,
19, 1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1...
```

10 баллов

использование

- p1:

```
Usage: ./a.out <from> <to> <chan1> <chan2>
```

- p2:

```
Usage: ./a.out <chan1> <chan2>
```

Схема решения

В этом решении я также использую

```
int* find_all_numbers(char* string, ssize_t* ans_count)
```

работает примерно как `map(int, input().split())` в питоне. То есть парсит строку на массив чисел. Причем `ans_count` длина получившегося массива. Она занимает так многоо строчек, потому что динамически меняет длину массива чисел, а так же длину последнего считанного числа(позже я понял что это зря, потому что все числа длиннее 12 символов и так упали бы на моменте перевода их в int).

```
char* make_string_from_array(int* arr, ssize_t size)
```

создает строку формата: "there are 2 numbers: 123, 123"

всего в программе два процесса в отдельных программах:

- первый считывает содержимое файла буффером в 128 байтов и направляет в первую очередь сообщений, после чего считывает данные буффером в 128 байтов из второй очереди сообщений и выводит в файл.
- второй считывает эти данные буффером в 128 байтов из первой очереди сообщений, обрабатывает благодаря описанным выше функциям и перенаправляет результат буффером в 128 байтов во вторую очередь сообщений.

небольшая проблема

Мой вариант требует чтобы я вывел вначале количество встреченных чисел, а после каждое из них. При таком подходе не получится прямо во время считывания ограниченным буффером строить итоговую строку, потому что в моменте не понятно какова будет итоговая длина. Поэтому в `p2.c` я считываю 'чанками' по 128 байтов из канала и добавляю их к `data`, то есть массив одновременно содержащий все данные и записываю после обработки тоже из этого массива чанками по 128 символов.

тестирование

модифицировал `run-tests` вот так:

```
#!/bin/bash

touch msgq1 msgq2

run_test() {
    local test_num=$1
    local input_file="../tests/test$test_num"
    local output_file="$test_num"

    echo "Running test $test_num..."

    ./p1 "$input_file" "$output_file" msgq1 msgq2 &
    sleep 1
    ./p2 msgq1 msgq2 &

    wait

    echo "Test $test_num completed."
}

for i in {1..5}; do
    run_test $i
done

rm msgq1 msgq2
```

1.

```
there are 2 numbers: 123, 123
```

2.

```
there are 0 numbers:
```

3.

```
there are 4 numbers: 12345, 4556, 5, 2
```

4. (The Beatles - 8 Days a Week)

```
there are 11 numbers: 8, 8, 8, 8, 8, 8, 8, 8, 1, 8, 8
```

5. (Holy Bible)

Этот файл очень большой, (все таки все числа в библии!) поэтому его можно посмотреть здесь

```
there are 68338 numbers: 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8,
1, 9, 1, 10, 1, 11, 1, 12, 1, 13, 1, 14, 1, 15, 1, 16, 1, 17, 1, 18, 1,
19, 1, 20, 1, 21, 1, 22, 1, 23, 1, 24, 1, 25, 1, 26, 1...
```