

# Практическая работа 5

## Процессы

### 1 Дополнительные теоретические сведения

#### 1.1 Демонстрация запуска процесса

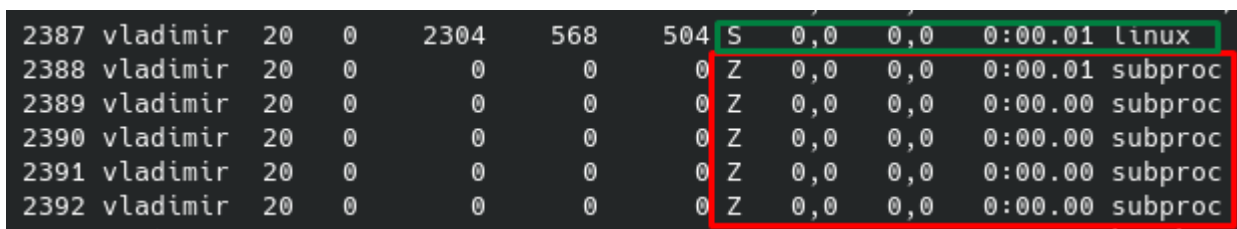
Чтобы продемонстрировать тот факт, что родительский процесс запустил нужное количество дочерних, следует воспользоваться следующими средствами:

- в Windows нужно открыть Диспетчер задач (Ctrl + Shift + Esc), перейти на вкладку «Подробности» (если такая отсутствует, необходимо развернуть окно, нажав на стрелку «Подробнее») и найти в списке процессов нужный (название будет совпадать с названием исполняемого файла);
- в Linux следует воспользоваться командой `top`, которую следует запустить из терминала и клавишами `↓ ↑` или `PageDown PageUp` прокрутить список процессов до нужного (для выхода нажмите `q`).

Однако решение задач из представленных вариантов не является ресурсоёмкой задачей, в связи с чем процессы будут завершать свою работу достаточно быстро, и увидеть их с использованием представленных выше утилит не представится возможным. Для решения этой проблемы предлагается использовать функцию «засыпания» процесса, т.е. его искусственной приостановки на некоторое время. В Windows такая функция называется **Sleep** и принимает количество миллисекунд, на которое нужно «усыпить» процесс. В Linux аналогом является функция **sleep**, но время она принимает в секундах. Таким образом, для демонстрации запуска нужного количества подпроцессов следует в коде для дочернего процесса в самом начале вызвать усыпляющую функцию, после чего запустить родительский процесс и воспользоваться средством просмотра.

Чтобы временно приостановить родительский процесс (например, чтобы продемонстрировать появление «зомби» в Linux), для него следует так же вызвать функцию усыпления, либо воспользоваться кроссплатформенной функцией **getchar()**, приостанавливающей процесс до того момента, пока пользователь не введёт Enter с клавиатуры.

Для первого примера будет продемонстрировано появление «зомби» в Linux. В данном случае нет смысла приостанавливать дочерние процессы, т.к. для превращения в «зомби» они должны завершиться, т.е. код дочерних процессов остаётся без изменения (в данном случае он состоит из единственной инструкции – **exit(EXIT\_SUCCESS);**). Чтобы породить «зомби», родительский процесс не должен вызывать **wait** для своих потомков, а значит, следует приостановить родительский процесс с помощью **getchar** перед вызовом **wait**, после чего проверить наличие нужного количества «зомби» с помощью **top** и далее продолжить выполнение процесса для корректного завершения. В результате можно будет увидеть следующую картину в выводе **top** (рисунок 1.1).



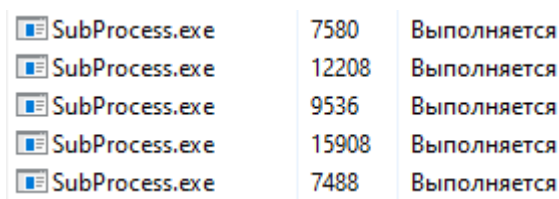
2387	vladimir	20	0	2304	568	504	S	0,0	0,0	0:00.01	linux
2388	vladimir	20	0	0	0	0	Z	0,0	0,0	0:00.01	subproc
2389	vladimir	20	0	0	0	0	Z	0,0	0,0	0:00.00	subproc
2390	vladimir	20	0	0	0	0	Z	0,0	0,0	0:00.00	subproc
2391	vladimir	20	0	0	0	0	Z	0,0	0,0	0:00.00	subproc
2392	vladimir	20	0	0	0	0	Z	0,0	0,0	0:00.00	subproc

Рисунок 1.1 – Порождение «зомби»

На рисунке 1.1 зелёной рамкой выделен родительский процесс, красной – дочерние процессы. Статус **Z** дочерних процессов означает, что они являются «зомби».

На втором примере будет продемонстрирован запуск нескольких дочерних процессов в Windows. Для этого первой инструкцией в коде дочернего процесса будет сделана строка **Sleep(30000)**, чтобы каждый из запущенных процессов был усыплен на 30 секунд, прежде чем выполнить

свою задачу. За это время будет возможно продемонстрировать данные процессы в диспетчере задач, как показано на рисунке 1.2.



SubProcess.exe	7580	Выполняется
SubProcess.exe	12208	Выполняется
SubProcess.exe	9536	Выполняется
SubProcess.exe	15908	Выполняется
SubProcess.exe	7488	Выполняется

Рисунок 1.2 – Дочерние процессы

## 1.2 Передача данных от родительского процесса дочерним и обратно

Ввиду того что для решения задач по варианту родительский процесс должен будет делегировать свою задачу дочерним, возникает необходимость каким-то образом сообщить дочерним процессам, какая именно часть задачи достанется каждому из них. К примеру, если задачей является расчёт суммы ряда чисел, и родительский процесс передаёт какую-то часть полученного ряда каждому из порождённых процессов, подпроцессам необходимо знать, какую именно часть им нужно суммировать. Лучшей практикой для передачи данных в таких целях являются средства межпроцессного взаимодействия (ИРС), однако, ввиду того что они ещё не были рассмотрены в рамках наших занятий, для этой цели предлагается использовать файлы. К примеру, в задаче, описанной выше, это можно сделать следующим образом:

- 1) Родительский процесс получает файл, в котором записан весь массив чисел для суммирования.
- 2) Родительский процесс считывает весь массив из файла.
- 3) Родительский процесс создаёт столько файлов, сколько он собирается породить дочерних процессов и в каждый из созданных файлов записывает ту часть считанного массива, которая должна быть просуммирована соответствующим дочерним процессом.
- 4) Родительский процесс запускает дочерний, передавая в качестве аргумента командной строки имя предназначенного для него файла.

5) Дочерний процесс получает имя предназначенного для него файла из аргумента командной строки, открывает его и считывает свою часть массива.

6) Дочерний процесс суммирует считанные числа.

7) Дочерний процесс создаёт файл с уникальным именем, таким, чтобы родительский процесс мог точно понять, какой из файлов каким процессом создан (создать такое уникальное имя можно, опираясь на PID процесса или на имя полученного файла с входными данными).

8) Дочерний процесс записывает результат в созданный файл и завершает свою работу.

9) Родительский процесс дожидается окончания работы всех дочерних.

10) Родительский процесс поочерёдно открывает файлы, созданные каждым из дочерних процессов и считывает из них результаты.

11) Родительский процесс суммирует считанные результаты и выдаёт полученную сумму как решение задачи.

Представленный подход имеет ряд недостатков, т.к. создаётся большое количество ненужных файлов и тратится время на осуществление файлового ввода-вывода, однако он может быть использован в рамках выполнения данной работы, т.к. позволяет легко отследить все взаимодействия между процессами, а также является простым в реализации.

### 1.3 Получение ID текущего и родительского процессов

В случае, если процессу необходимо получить собственный ID (например, чтобы задать названия для порождаемых файлов или вывести ID в консоль для отладки), можно воспользоваться следующими функциями:

Для Windows – **GetCurrentProcessID**

Для Linux – **getpid**

## 2 Задание

Для выполнения работы необходимо:

1) Повторно ознакомиться с теоретическим материалом из презентации по теме «Процессы», а также рассмотреть теоретическую часть из данного файла (раздел 1).

**Внимание! Все настройки, действия и манипуляции, проводимые для выполнения заданий из пункта 2, необходимо фиксировать в отчёте в виде снимков экрана или листингов вводимых команд.**

2) С помощью языка программирования Си или C++ необходимо написать **четыре** программы (по одной для родительского и дочернего процесса для двух ОС) с идентичным функционалом, решающих задачу из варианта, – одну для ОС Windows и одну для ОС Linux. Для выполнения задания может использоваться любая среда разработки. При выполнении необходимо учитывать следующие требования:

а) Задача считается решённой, если необходимое по варианту действие выполняется для всех допустимых входных данных.

б) Для работы с процессами (запуска, ожидания, завершения) следует использовать только предназначенные для этого функции из API соответствующих ОС. **Внимание! Не используйте `return` для завершения процесса в данной работе.** Однако для файлового и консольного ввода-вывода **можно** воспользоваться любыми доступными средствами.

в) Не следует использовать коды завершения процессов и аргументы командной строки для непосредственной передачи обрабатываемых данных. Через аргументы следует передавать только названия файлов, содержащих данные или другую вспомогательную информацию. Коды возврата следует использовать только для сообщения об успешном или ошибочном завершении работы процесса.

г) Программа должна корректно завершаться, не вызывая аварийный останов.

д) Программа должна брать входные данные из аргументов, переданных при запуске в консоли. В случае, если количество переданных аргументов не равно ожидаемому, программа должна вывести подсказку для пользователя, поясняющую правила её (программы) использования.

е) Возвращаемые значения **всех** вызываемых функций должны проверяться на предмет возникновения ошибок. В случае возникновения ошибки необходимо вывести сообщение, оповещающее пользователя о произошедшем, содержащее в обязательном порядке код ошибки и её текстовое описание. В случае, если в результате возникшей ошибки программа должна быть завершена, перед завершением необходимо освободить все занятые ресурсы (очистить выделенную память, закрыть открытые файлы).

ж) Программа должна работать с любым количеством данных без необходимости внесения изменений в код и перекомпиляции. Если количество процессов, которые необходимо породить, больше количества входных данных, необходимо запустить кол-во процессов, вдвое меньшее кол-ва входных данных и вывести соответствующее предупреждение.

3) Оформить отчёт в соответствии с шаблоном из файла «Шаблон и требования к оформлению отчёта по практической работе». Не забыть прикрепить к отчёту **исходные коды** обеих программ **в виде приложения (не приложить отдельными файлами, а именно вставить в сам файл с отчётом под заголовками «Приложение А», «Приложение Б» и т.д.)**. В отчёте необходимо отразить, как минимум:

а) Результаты работы программы в виде снимков экрана, демонстрирующих работу программы для 2-3 вариантов верных и для

каждого типа неверных входных данных (чтобы продемонстрировать все предусмотренные сообщения об ошибках).

б) Процесс сборки и запуска программы.

в) Изменения ключевых значений (в переменных) в ходе работы программы.

г) Описание реализованного алгоритма, содержащее листинги соответствующих отрезков кода программы. В описание следует включать только ключевые моменты работы программы (т.е. описывать обработку ошибок и освобождение памяти не требуется).

д) Для программы на Linux необходимо продемонстрировать все порождённые процессы в нормальном состоянии, а также в состоянии «зомби». Для программы на Windows необходимо продемонстрировать всё порождённые процессы.

4) Защитить работу на занятии. Во время защиты необходимо:

а) Продемонстрировать работу программы.

б) При необходимости изменить входные данные и продемонстрировать, что программа отработывает верно на новых данных.

в) При необходимости внести изменения в программный код для демонстрации порождённых процессов или «зомби» с помощью Диспетчера задач или утилиты top.

г) Ответить на вопросы по логике работы программы (почему и зачем была написана та или иная строка, каково её назначение и вклад в решение данной задачи).

д) Ответить на вопросы по теоретической части (**в том числе по лекциям**).

5) Дополнительное задание: дополнительно реализовать третью кроссплатформенную программу, решающую задачу по варианту на языке программирования, отличном от C/C++ и Python. Даёт два дополнительных балла. **Не обязательно к выполнению!**

### 3 Варианты заданий

Общее требование для всех заданий: если количество входных данных (чисел или символов) не делится нацело на количество дочерних процессов, числа должны распределяться между дочерними процессами следующим образом:

Пусть кол-во входных данных равно  $M$ , кол-во дочерних процессов равно  $N$ , тогда первым  $N - 1$  процессам должны быть назначены фрагменты входных данных длины:

$$n = \left\lfloor \frac{M}{N} \right\rfloor,$$

а оставшемуся  $N$ -ному процессу:

$$n_N = M - (N - 1) \left\lfloor \frac{M}{N} \right\rfloor,$$

где  $\lfloor x \rfloor$  – округление вещественного числа  $x$  в меньшую сторону. Например, если дано 10 чисел и их необходимо распределить между 4 процессами, получим:

$$n = \left\lfloor \frac{10}{4} \right\rfloor = \lfloor 2,5 \rfloor = 2,$$

$$n_N = 10 - (4 - 1) \left\lfloor \frac{10}{4} \right\rfloor = 10 - 3 \cdot 2 = 10 - 6 = 4.$$

Таким образом, первым трём процессам достанется по 2 числа, а последнему – 4 числа.

Однако, если  $N > \frac{M}{2}$ , следует уменьшить количество дочерних процессов до  $\left\lfloor \frac{M}{2} \right\rfloor$  и вывести соответствующее предупреждение для пользователя.



№	Задача
1	<p>В файле записан ряд целых чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму записанных в файл чисел. Для расчёта суммы программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
2	<p>В файле записан ряд вещественных чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму записанных в файл чисел. Для расчёта суммы программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
3	<p>В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько заглавных букв содержится в строке. Для расчёта программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть строки. Каждый из дочерних</p>

	<p>процессов должен рассчитать количество заглавных букв в своей части строки и вернуть его родительскому. Родительский процесс должен рассчитать сумму полученных от дочерних процессов чисел и вывести на консоль результат. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
4	<p>В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько строчных (не заглавных) букв содержится в строке. Для расчёта программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть строки. Каждый из дочерних процессов должен рассчитать количество строчных (не заглавных) букв в своей части строки и вернуть его родительскому. Родительский процесс должен рассчитать сумму полученных от дочерних процессов чисел и вывести на консоль результат. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
5	<p>В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько цифр (не чисел, а именно цифр) содержится в строке. Для расчёта программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть строки. Каждый из дочерних процессов должен рассчитать количество цифр в своей части строки и вернуть его родительскому. Родительский процесс должен рассчитать сумму полученных от дочерних процессов чисел и вывести на консоль результат. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>

6	<p>В файле записан текст. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько управляющих символов (с номерами 0x00-0x1F и 0x7F в таблице ASCII) содержится в тексте. Для расчёта программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть текста. Каждый из дочерних процессов должен рассчитать количество управляющих символов в своей части текста и вернуть его родительскому. Родительский процесс должен рассчитать сумму полученных от дочерних процессов чисел и вывести на консоль результат. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
7	<p>В файле записан ряд целых чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму чётных чисел, записанных в файл. Для расчёта суммы программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму чётных из переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
8	<p>В файле записан ряд целых чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму нечётных чисел, записанных в файл. Для расчёта суммы программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму нечётных из переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и</p>

	вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.
9	В файле записан ряд целых чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму квадратов записанных в файл чисел. Для расчёта суммы квадратов программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму квадратов переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.
10	В файле записан ряд вещественных чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму квадратов записанных в файл чисел. Для расчёта суммы квадратов программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму квадратов переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.
11	В файле записан ряд положительных целых чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму квадратных корней записанных в файл чисел. Для расчёта суммы квадратных корней программа должна создать N дочерних процессов (N передаётся вторым аргументом

	<p>командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму квадратных корней переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
12	<p>В файле записан ряд положительных вещественных чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму квадратных корней записанных в файл чисел. Для расчёта суммы квадратных корней программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму квадратных корней переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.</p>
13	<p>В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и рассчитать количество вхождений в эту строку одного символа, переданного в качестве третьего аргумента командной строки. Для расчёта количества вхождений программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученной строки. Каждый из дочерних процессов должен рассчитать количество вхождений символа в переданную ему часть строки и вернуть его родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую</p>

	сумму. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.
14	В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и рассчитать количество таких символов в этой строке, которые стоят в алфавите раньше символа, переданного в качестве третьего аргумента командной строки. Для расчёта количества символов программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученной строки. Каждый из дочерних процессов должен рассчитать количество соответствующих условию символов в переданной ему части строки и вернуть его родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы. Под позицией в алфавите в данной задаче понимается позиция в таблице ASCII.
15	В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и рассчитать количество таких символов в этой строке, которые стоят в алфавите после символа, переданного в качестве третьего аргумента командной строки. Для расчёта количества символов программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученной строки. Каждый из дочерних процессов должен рассчитать количество соответствующих условию символов в переданной ему части строки и вернуть его родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 символов,

	следует вывести соответствующее сообщение для пользователя и завершить работу программы. Под позицией в алфавите в данной задаче понимается позиция в таблице ASCII.
--	--