Лабораторные работы №1

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

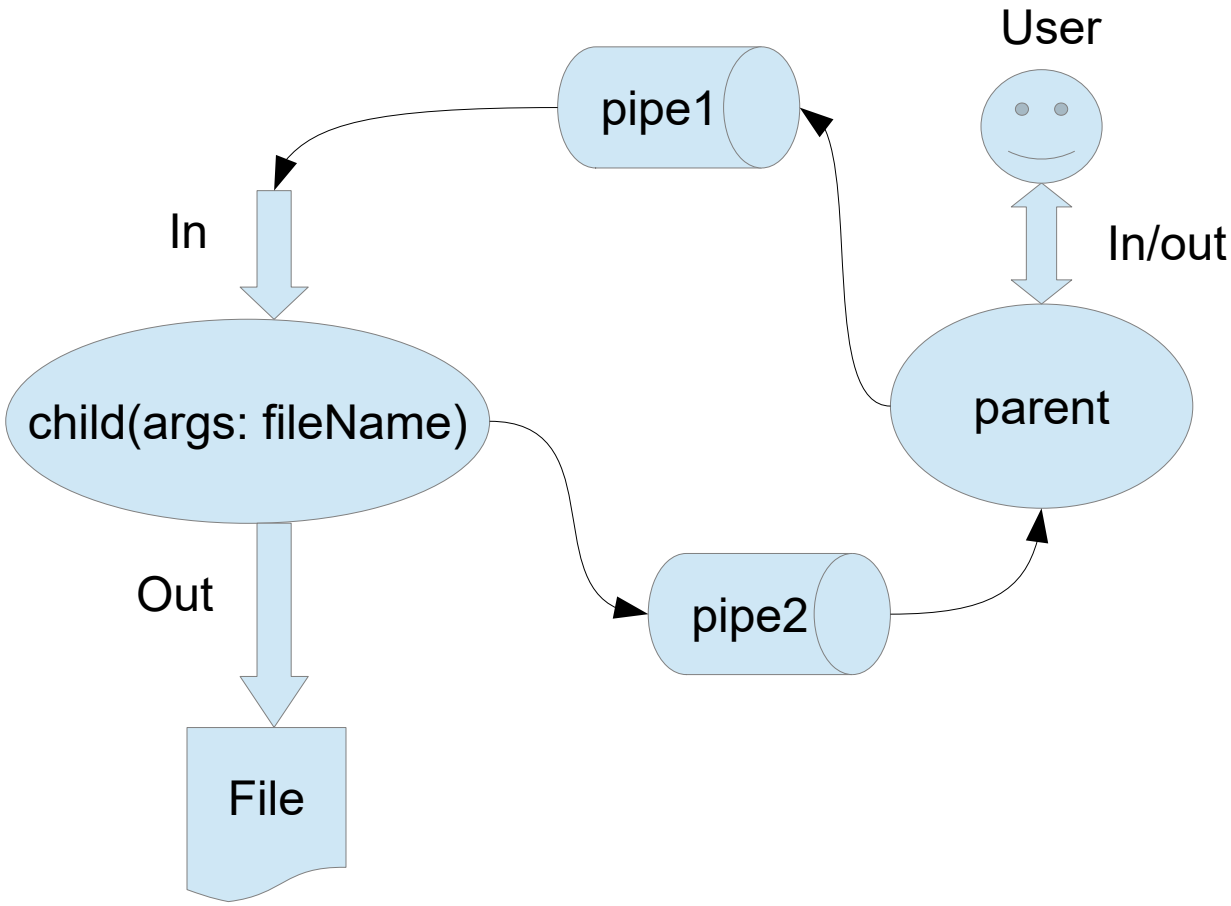
Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe).
Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Варианты задания

Варианты разбиты на группы. Для каждого варианта приложена схема организации межпроцессорного взаимодействия.

Группа вариантов 1



Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

1 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

2 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

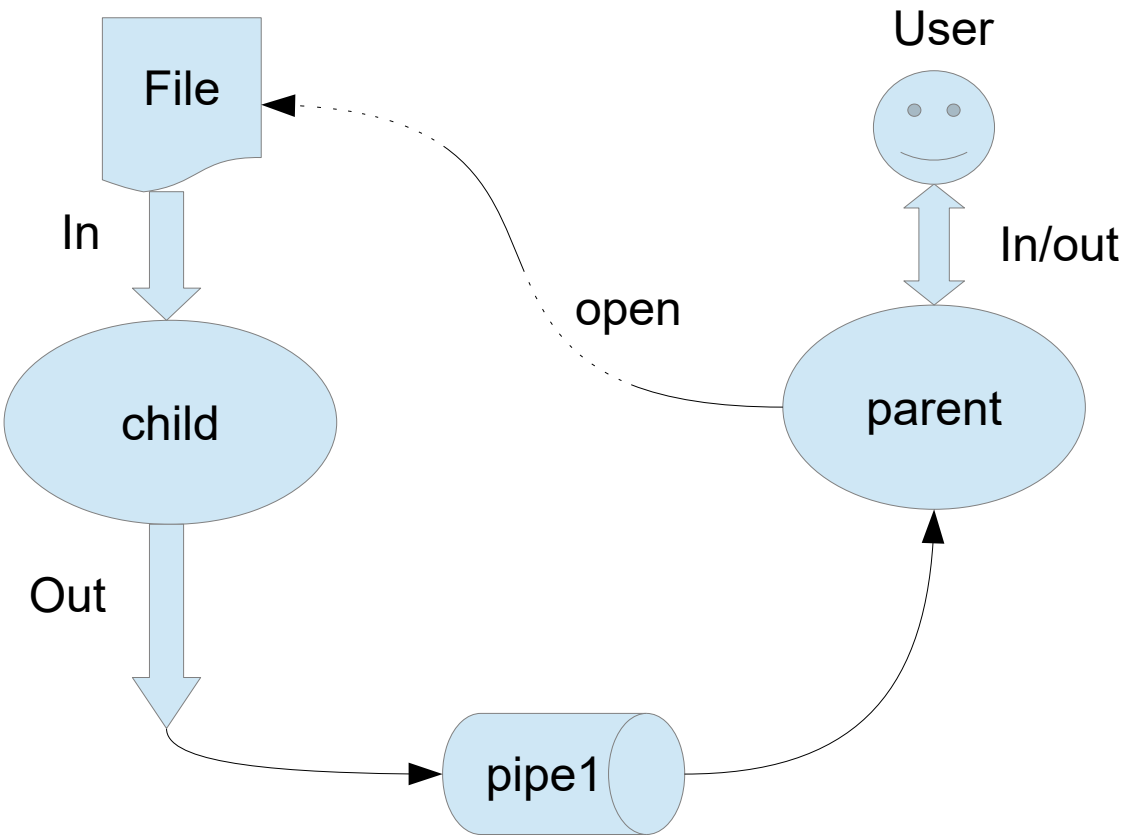
3 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

4 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление

первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

5 вариант) Пользователь вводит команды вида: «число<endline>». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

Группа вариантов 2



Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

6 вариант) В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

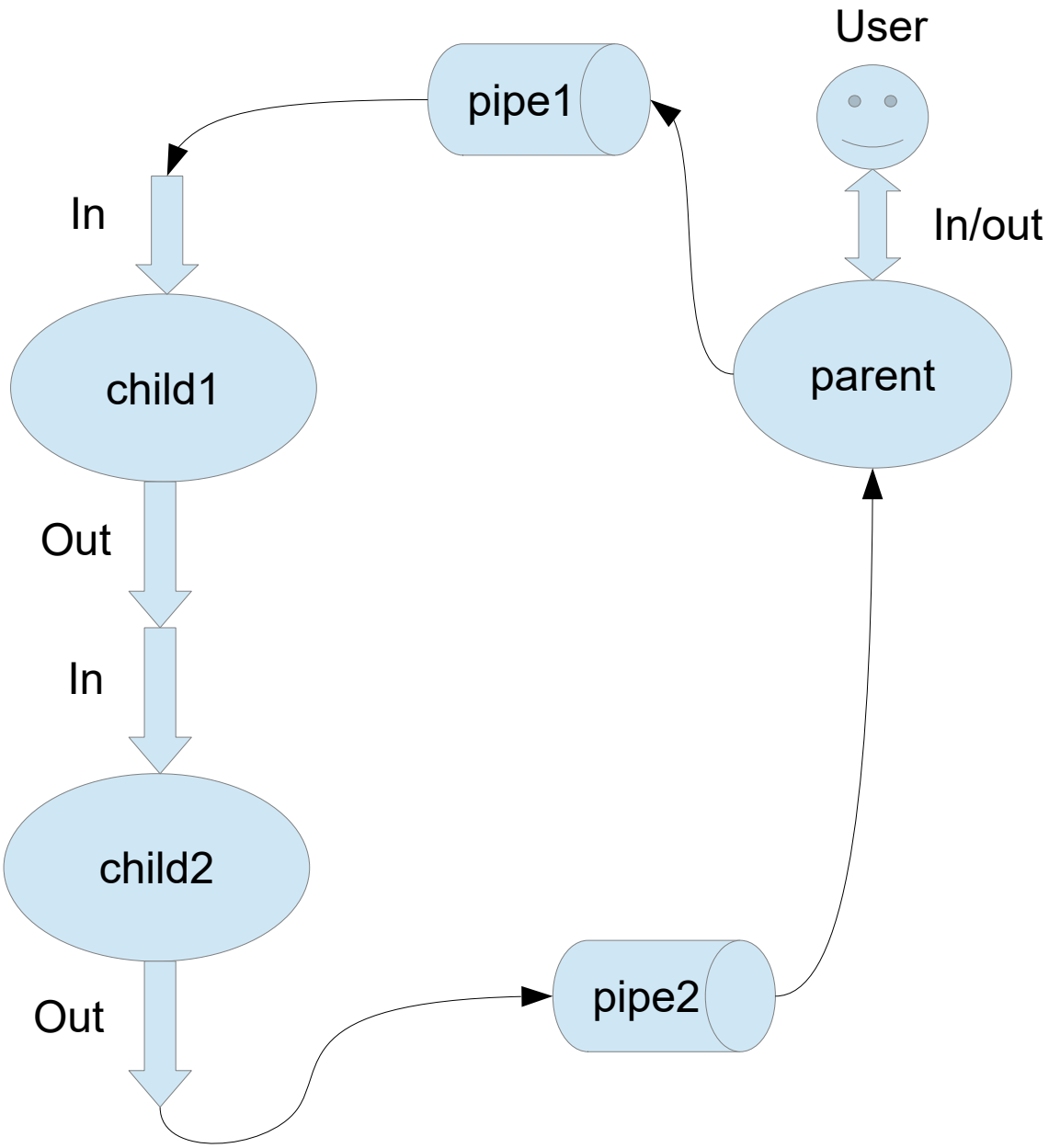
7 вариант) В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

8 вариант) В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

9 вариант) В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

10 вариант) В файле записаны команды вида: «число<endline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Группа вариантов 3



Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.
Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

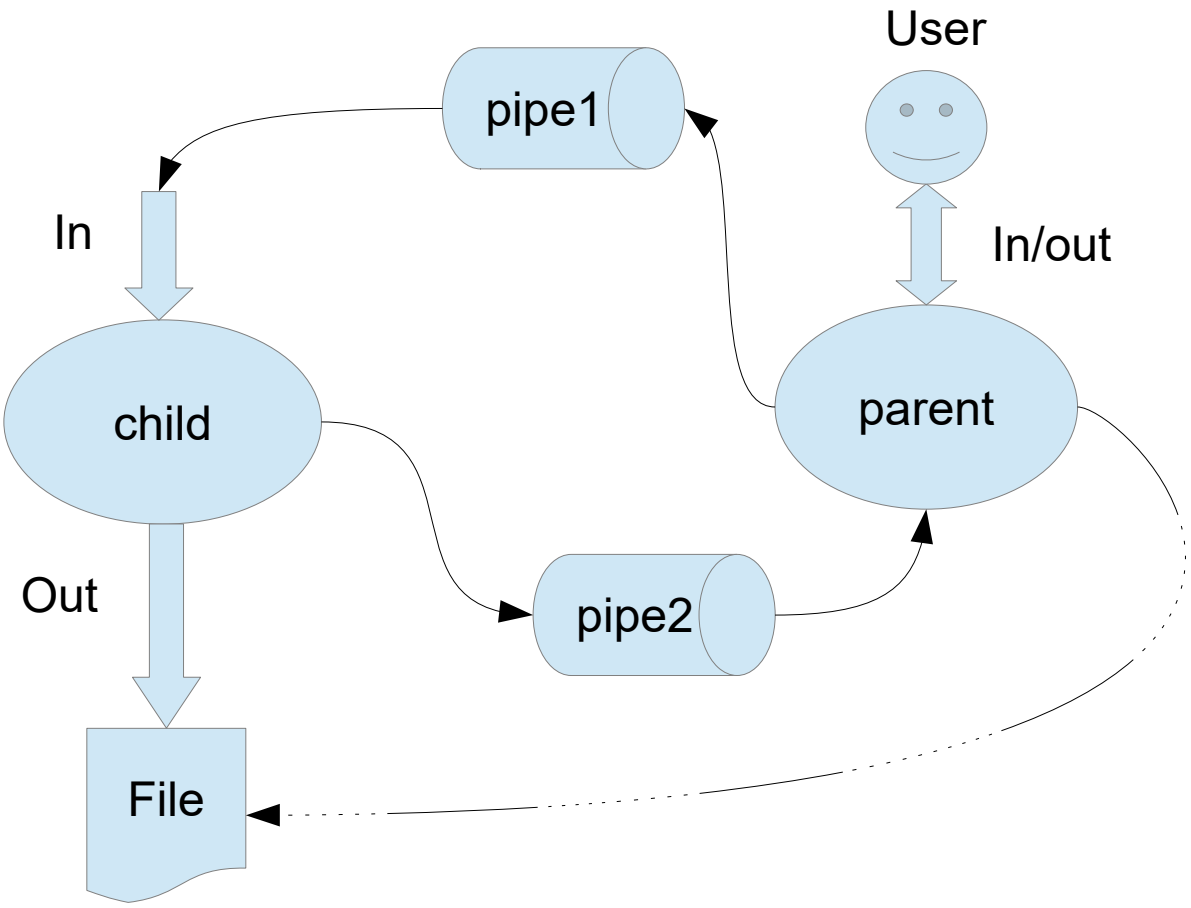
11 вариант) Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

12 вариант) Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

13 вариант) Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «_».

14 вариант) Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

Группа вариантов 4

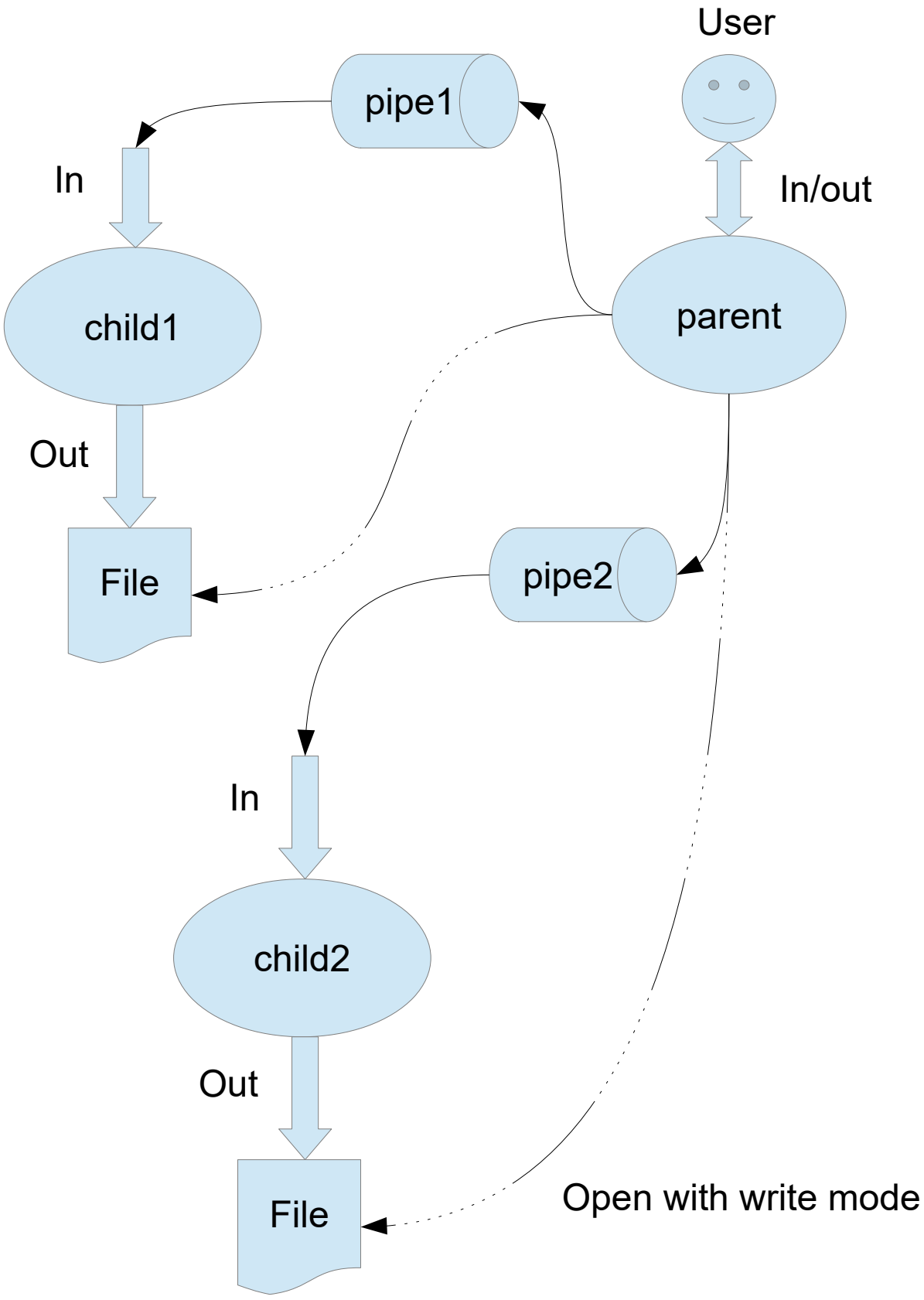


Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Вариант 15) Правило проверки: строка должна начинаться с заглавной буквы

Вариант 16) Правило проверки: строка должна оканчиваться на «.» или «;»

Группа вариантов 5



Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Вариант 17) Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк.

Вариант 18) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

Вариант 19) Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют все гласные из строк.

Вариант 20) Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

Вариант 21) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Вариант 22) Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

Примечание: Программы должны быть на языке Си.

Варианты задания выдаются преподавателем.

Системные вызовы

Для ОС Linux

- pid_t fork() - создание дочернего процесса
- int execve(const char *filename, char *const argv[], char *const envp[]) (и другие вариации exec) - замена образа памяти процесса
- pid_t waitpid(pid_t pid, int *status, int options) - Ожидание завершения дочернего процесса
- void exit(int status) - завершения выполнения процесса и возвращение статуса
- int pipe(int pipefd[2]) - создание неименованного канала для передачи данных между процессами
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- int open(const char *pathname, int flags, mode_t mode) - открытие\создание файла
- int close(int fd) - закрыть файл
- int mkfifo(const char *pathname, mode_t mode) - создание именованного канала

Для ОС Windows

- BOOL WINAPI CreateProcess(…) - создание нового процесса
- WaitForSingleObject(…) - ожидание завершения процесса
- ExitProcess(…) - завершение выполнения процесса
- CreateFile/SetNamedPipeHandleState - создание именованного канала и установления режима его использования
- Int _dup2(int fd1, int fd2) - переназначение файлового дескриптора
- OpenFile(…) - открытие нового файла
- CreatePipe(…) - создание безымянного канала
- CreateFile(…) - создание нового файла
- CloseHandle(…) - закрытие объекта ОС по "заголовку". Подходит для закрытия файлов.

Лабораторные работы №2

Цель работы

Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Варианты задания

1. Отсортировать массив целых чисел при помощи битонической сортировки
2. Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки
3. Отсортировать массив целых чисел при помощи параллельной сортировки слиянием
4. Отсортировать массив целых чисел при помощи TimSort
5. Отсортировать массив целых чисел при помощи четно-нечетной сортировки Бетчера
6. Произвести перемножение 2-ух матриц, содержащих комплексные числа
7. Два человека играют в кости. Правила игры следующие: каждый игрок делает бросок 2-ух костей K раз; побеждает тот, кто выбросил суммарно большее количество очков. Задача программы экспериментально определить шансы на победу каждого из игроков. На вход программе подается K, какой сейчас тур, сколько очков суммарно у каждого из игроков и количество экспериментов, которые должна произвести программа
8. Есть K массивов одинаковой длины. Необходимо сложить эти массивы. Необходимо предусмотреть стратегию, адаптирующуюся под количество массивов и их длину (по количеству операций)
9. Рассчитать детерминант матрицы (используя определение детерминанта)
10. Решить систему линейных уравнений методом Гаусса
11. Наложить K раз медианный фильтр на матрицу, состоящую из целых чисел. Размер окна задается пользователем
12. Наложить K раз фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы

13. Наложить K раз фильтр, использующий матрицу свертки, на матрицу, состоящую из вещественных чисел. Размер окна задается пользователем
14. Есть набор 128 битных чисел, записанных в шестнадцатеричном представлении, хранящихся в файле. Необходимо посчитать их среднее арифметическое. Округлить результат до целых. Количество используемой оперативной памяти должно задаваться "ключом"
15. Есть колода из 52 карт, рассчитать экспериментально (метод Монте-Карло) вероятность того, что сверху лежат две одинаковых карты. Количество раундов задаётся ключом программы
16. Задаётся радиус окружности. Необходимо с помощью метода Монте-Карло рассчитать её площадь
17. Найти в большом целочисленном массиве минимальный и максимальный элементы
18. Найти образец в строке наивным алгоритмом
19. Дан массив координат (x, y). Пользователь вводит число кластеров. Проведите кластеризацию методом k-средних
20. Дан массив координат (x, y, z). Необходимо найти три точки, которые образуют треугольник максимальной площади

Лабораторные работы №3

Цель работы

Приобретение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Варианты задания

См. лабораторная работа №1.

Варианты выбираются такие же как и в лабораторной работе №1.

Лабораторные работы №4

Тема

Динамические библиотеки

Цель работы

Целью является приобретение практических навыков в:

- Создание динамических библиотек
- Создание программ, которые используют функции динамических библиотек

Задание

Требуется создать динамические библиотеки, которые реализуют определенный функционал. Далее использовать данные библиотеки 2-мя способами:

1. Во время компиляции (на этапе «линковки»/linking)
2. Во время исполнения программы. Библиотеки загружаются в память с помощью интерфейса ОС для работы с динамическими библиотеками

В конечном итоге, в лабораторной работе необходимо получить следующие части:

- Динамические библиотеки, реализующие контракты, которые заданы вариантом;
- Тестовая программа (*программа №1*), которая используют одну из библиотек, используя знания полученные на этапе компиляции;
- Тестовая программа (*программа №2*), которая загружает библиотеки, используя только их местоположение и контракты.

Провести анализ двух типов использования библиотек.

Пользовательский ввод для обоих программ должен быть организован следующим образом:

1. Если пользователь вводит команду «0», то программа переключает одну реализацию контрактов на другую (необходимо только для *программы №2*). Можно реализовать лабораторную работу без данной функции, но максимальная оценка в этом случае будет «хорошо»;
2. «1 arg1 arg2 … argN», где после «1» идут аргументы для первой функции, предусмотренной контрактами. После ввода команды происходит вызов первой функции, и на экране появляется результат её выполнения;
3. «2 arg1 arg2 … argM», где после «2» идут аргументы для второй функции, предусмотренной контрактами. После ввода команды происходит вызов второй функции, и на экране появляется результат её выполнения.

Контракты и реализации функций

№	Описание	Сигнатура	Реализация 1	Реализация 2
1	Рассчет интеграла	Float	Подсчет	Подсчет

	функции sin(x) на отрезке [A, B] с шагом e	SinIntegral(float A, float B, float e)	интеграла методом прямоугольников.	интеграла методом трапеций.
2	Рассчет производной функции cos(x) в точке A с приращением deltaX	Float Derivative(float A, float deltaX)	f'(x) = (f(A + deltaX) – f(A))/deltaX	f'(x) = (f(A + deltaX) – f(A-deltaX))/(2*deltaX)
3	Подсчёт количества простых чисел на отрезке [A, B] (A, B - натуральные)	Int PrimeCount(int A, int B)	Наивный алгоритм. Проверить делимость текущего числа на все предыдущие числа.	Решето Эратосфена
4	Подсчёт наибольшего общего делителя для двух натуральных чисел	Int GCF(int A, int B)	Алгоритм Евклида	Наивный алгоритм. Пытаться разделить числа на все числа, что меньше A и B.
5	Рассчет значения числа Пи при заданной длине ряда (K)	float Pi(int K)	Ряд Лейбница	Формула Валлиса
6	Рассчет значения числа e(основание натурального логарифма)	Float E(int x)	(1 + 1/x) ^ x	Сумма ряда по n от 0 до x, где элементы ряда равны: (1/(n!))
7	Подсчет площади плоской геометрической фигуры по двум сторонам	Float Square(float A, float B)	Фигура прямоугольник	Фигура прямоугольный треугольник
8	Перевод числа x из десятичной системы счисления в другую	Char* translation(long x)	Другая система счисления двоичная	Другая система счисления троичная
9	Отсортировать целочисленный массив	Int * Sort(int * array)	Пузырьковая сортировка	Сортировка Хоара

Варианты

№	Функция 1	Функция 2
1	1	2
2	1	3
3	1	4
4	1	5
5	1	6
6	1	7
7	1	8
8	1	9
9	2	3
10	2	4
11	2	5
12	2	6
13	2	7
14	2	8
15	2	9
16	3	4
17	3	5
18	3	6
19	3	7
20	3	8
21	3	9
22	4	5
23	4	6
24	4	7
25	4	8
26	4	9

27	5	6
28	5	7
29	5	8
30	5	9
31	6	7
32	6	8
33	6	9
34	7	8
35	7	9
36	8	9

Справочный материал

1. https://msdn.microsoft.com/en-us/library/ms235636.aspx
2. https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175(v=vs.85).aspx
3. https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212(v=vs.85).aspx
4. http://www.ibm.com/developerworks/library/l-dynamic-libraries/

Лабораторные работы №5-7

Цель работы

Целью является приобретение практических навыков в:

- Управлении серверами сообщений (№5)
- Применение отложенных вычислений (№6)
- Интеграция программных систем друг с другом (№7)

Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы. Список основных поддерживаемых команд:

Создание нового вычислительного узла

Формат команды: create id [parent]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода:

«Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> create 10 5

Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error:id: [Custom error]» - любая другая обрабатываемая ошибка

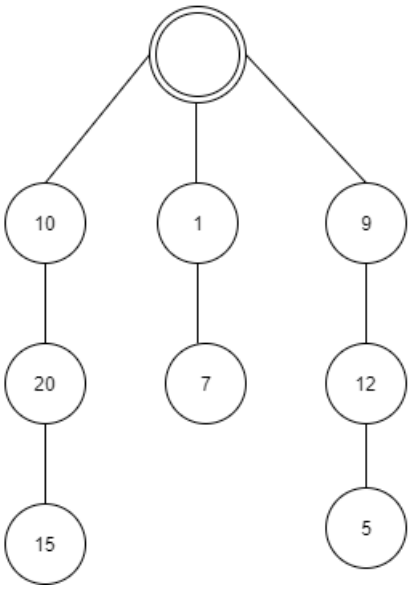
Пример:

Можно найти в описании конкретной команды, определенной вариантом задания.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

Типы топологий

Топология 1

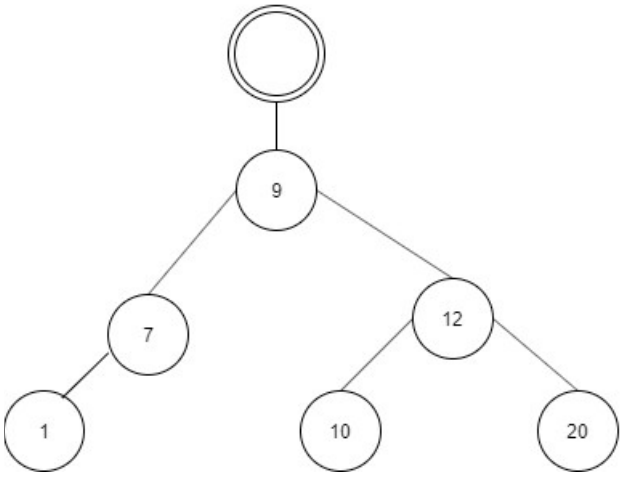


Все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

Топология 2

Аналогично топологии 2, но узлы находятся в дереве общего вида.

Топология 3



Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.

Топология 4

Аналогично топологии 4, но узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

Типы команд для вычислительных узлов

Набор команд 1 (подсчет суммы n чисел)

Формат команды: exec id n k_1 … k_n

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

n – количество складываемых чисел (от 1 до 10_8)

k_1 … k_n – складываемые числа

Пример:

> exec 10 3 1 2 3

Ok:10: 6

Набора команд 2 (локальный целочисленный словарь)

Формат команды сохранения значения: exec id name value

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

name – ключ, по которому будет сохранено значение (строка формата [A-Za-z0-9]+)

value – целочисленное значение

Формат команды загрузки значения: exec id name

Пример:

> exec 10 MyVar

Ok:10: 'MyVar' not found

> exec 10 MyVar 5

Ok:10

> exec 12 MyVar

Ok:12: 'MyVar' not found

> exec 10 MyVar

Ok:10: 5

> exec 10 MyVar 7

Ok:10

> exec 10 MyVar

Ok:10: 7

Примечания: Можно использовать std:map.

Набора команд 3 (локальный таймер)

Формат команды сохранения значения: exec id subcommand

subcommand – одна из трех команд: start, stop, time.

start – запустить таймер

stop – остановить таймер

time – показать время локального таймера в миллисекундах

Пример:

> exec 10 time

Ok:10: 0

>exec 10 start

Ok:10

>exec 10 start

Ok:10

прошло 10 секунд

> exec 10 time

Ok:10: 10000

прошло 2 секунды

>exec 10 stop

Ok:10

прошло 2 секунды

>exec 10 time

Ok:10: 12000

Набора команд 4 (поиск подстроки в строке)

Формат команды:

> exec id > text_string > pattern_string [result] – номера позиций, где найден образец, разделенный точкой с запятой

text_string — текст, в котором искать образец. Алфавит: [A-Za-z0-9]. Максимальная длина строки 10^8 символов

pattern_string — образец

Пример:

> exec 10

> abracadabra

> abra

Ok:10:0;7

> exec 10

> abracadabra

> mmm

Ok:10: -1

Примечания: Выбор алгоритма поиска не важен

Тип проверки доступности узлов

Команда проверки 1

Формат команды: pingall

Вывод всех недоступных узлов вывести разделенные через точку запятую.

Пример:

> pingall

Ok: -1 // Все узлы доступны

> pingall

Ok: 7;10;15 // узлы 7, 10, 15 — недоступны

Команда проверки 2

Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

Пример:

> ping 10

Ok: 1 // узел 10 доступен

> ping 17

Ok: 0 // узел 17 недоступен

Команда проверки 3

Формат команды: heartbit time

Каждый узел начинает сообщать раз в time миллисекунд о том, что он работоспособен. Если от узла нет сигнала в течении 4*time миллисекунд, то должна выводится пользователю строка: «Heartbit: node id is unavailable now», где id – идентификатор недоступного вычислительного узла.

Пример:

> heartbit 2000

Ok

Пример:

> ping 10

Ok: 1 // узел 10 доступен

> ping 17

Ok: 0 // узел 17 недоступен

Возможные технологии очередей сообщений

1. ZeroMQ
2. MSMQ
3. RabbitMQ
4. Nats

Варианты

№	Топология	Тип команд	Тип проверки доступности узлов
1	3	3	3
2	2	4	3
3	1	1	3
4	1	1	2
5	4	1	1
6	4	3	1
7	3	4	2
8	4	4	1
9	1	2	2
10	4	3	3
11	2	3	1
12	4	2	1
13	1	3	1
14	1	1	1
15	1	2	3
16	2	4	1
17	1	4	2
18	3	2	2
19	3	3	2
20	2	4	2
21	4	3	2
22	3	4	3
23	4	4	3
24	2	2	1
25	2	2	3
26	3	1	2
27	1	4	1
28	1	3	2
29	3	1	3
30	3	4	1
31	1	4	3
32	3	2	3
33	2	1	1
34	2	3	3
35	4	1	2
36	3	2	1
37	1	2	1
38	1	3	3
39	4	4	2
40	2	3	2
41	3	1	1
42	2	1	3
43	2	1	2
44	4	1	3
45	4	2	2
46	2	2	2
47	3	3	1
48	4	2	3

Лабораторные работы №8

Цель работы

Приобретение практических навыков диагностики работы программного обеспечения.

Задание

При выполнении лабораторных работ по курсу ОС необходимо продемонстрировать ключевые системные вызовы, которые в них используются и то, что их использование соответствует варианту ЛР.

По итогам выполнения всех лабораторных работ отчет по данной ЛР должен содержать краткую сводку по исследованию написанных программ.

Средства диагностики

Для ОС Windows

- Windbg
  - http://windbg.info/doc/1-common-cmds.html
- Sysinternals Suite
  - Handle.exe
  - Procmon.exe
  - Procexp.exe

Для ОС *nix:

- strace