Национальный исследовательский университет “Высшая школа экономики”.

Факультет компьютерных наук. Программная инженерия.

Операционные системы.

Индивидуальное домашнее задание №1 студента группы БПИ213 Абрамова Александра Сергеевича.

Вариант 30.

30. Разработать программу, которая ищет в ASCII-строке слова, **начинающиеся с заглавной буквы и формирует из них новую строку**, в которой слова разделяются пробелами. Слова состоят из букв. Все остальные символы являются разделителями слов.

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ РАБОТЫ

1. Каталоги `4 - 10` содержат программы на соответствующую оценку. Каждый из них содержит директорию `bin`, где сохраняются исполняемые файлы программы, а также директорию `output`, где представлены выходные данные программы на соответствующих тестах.
2. Каталог `lib` для удобства организации кода содержит заголовочные файлы и реализации методов, которые могут быть полезны сразу нескольким программам.
   2.1. Заголовочный файл `messages.h` содержит объявления методов `input`, `solve` и `output`, которые используются программой `10` для решения поставленной задачи с использованием очереди сообщений. Их определения представлены в директории `messages`.
   2.2. Заголовочный файл `pipes.h` содержит объявления методов `transfer` и `solve`, которые используются программами `4 - 9` для решения поставленной задачи с использованием именованных или неименованных каналов. Их определения представлены в директории `pipes`.
   2.3. Подробное описание алгоритмов представлено в виде комментариев в исходном коде.
3. Каталог `testing` содержит программы, позволяющие производить тестирование разработанных решений:
   3.1. Программа `gen.js` производит генерацию наборов тестовых данных и сохраняет входные и эталонные выходные данные в директории `tests`. Каждый тест описывается длиной вводимой строки.
   3.2. Программа `run.js` осуществляет последовательный запуск программы, номер которой указывается единственным аргументом командной строки, на всех наборах тестовых данных и печатает в стандартный поток вывода результат её выполнения на каждом тесте. В случае обнаружения ошибки во время выполнения выводится вердикт Runtime Error (RE); при получении неверных выходных данных - Wrong Answer (WA); иначе - OK.
4. Файл `compile.sh`, собирающий программу, номер которой указывается единственным аргументом командной строки при исполнении скрипта
5. Файл `package.json` содержит настройки утилиты `npm`, которая (совместно со средой `node` языка `JavaScript`) использовалась для запуска и тестирования программ. В файле определены следующие скрипты:
   5.1. `gen`, запускающий программу генерации тестовых данных и размещения их в директории `tests` каталога `testing`

5.2. `4 - 10`, производящие сборку программы на соответствующую оценку и запускающие её тестирование программой `run.js`

5.3. `all`, последовательно исполняющий все скрипты `4 - 10`

5.4. `clear`, удаляющий все полученные при сборке и запуске исполняемые файлы и файлы с выходными данными для удобства повторения тестирования

СБОРКА ПРОГРАММ

Для удобства сборки программ был реализован скрипт `compile.sh`, собирающий программу, номер которой указывается единственным аргументом командной строки (пр.: `./compile.sh 10`)

1. Для компиляции программы используется утилита `gcc` со следующими параметрами:
   1.1. `-O2`, позволяющий компилятору производить оптимизации программы для более эффективной работы алгоритма
   1.2. `-fsanitize=address,undefined,leak`, указывающий компилятору на необходимость добавления в код программы дополнительных инструкций, позволяющих отлавливать ошибки во время её выполнения, которые могут приводить к некорректному поведению
   1.3. `-fno-sanitize-recover=all`, требующий немедленного завершения программы с ненулевым кодом возврата в случае обнаружения любой ошибки во время выполнения
   1.4. `-std=c17`, указывающий версию спецификации языка C, которую необходимо использовать при компиляции - `C17`
   1.5. `-Werror` для прекращения процесса сборки программы с ошибкой в случае обнаружения любого предупреждения компилятора
   1.6. `-Wall` и `-Wextra` для включения всех предупреждений компилятора.
   1.7. `-Ilib` для подключения заголовочных файлов директории `lib`, описанных в п.2
   1.8. `-DCHUNK_SIZE=4000` или `-DCHUNK_SIZE=100` для указания размера буфера для хранения вводимых данных и результатов их обработки
2. Программы `4 - 7` компилируются одинаково путём сборки общих реализаций `lib/pipes/solve.c` и `lib/pipes/transfer.c`, а также основного файла `index.c` соответствующего каталога в исполняемый файл `index.exe` директории `bin`.
3. Программы `8 - 9` также компилируются одинаково за исключением размера буфера для хранения вводимых данных и результатов их обработки: для программы `8` размер буфера составляет `4001`, а для программы `9 - 101`; размер буфера равен значению макро-параметра `CHUNK_SIZE`, увеличенному на `1` (см. комментарии в исходном коде программ). Так как программы `8` и `9` реализуют взаимодействие независимых процессов, происходит сборка двух программ: `io.c` в исполняемый файл `io.exe`, а также `solver.c` в исполняемый файл `solver.exe`. При этом к обеим программам добавляются дополнительные модули - `lib/pipes/solve.c` и `lib/pipes/transfer.c`.

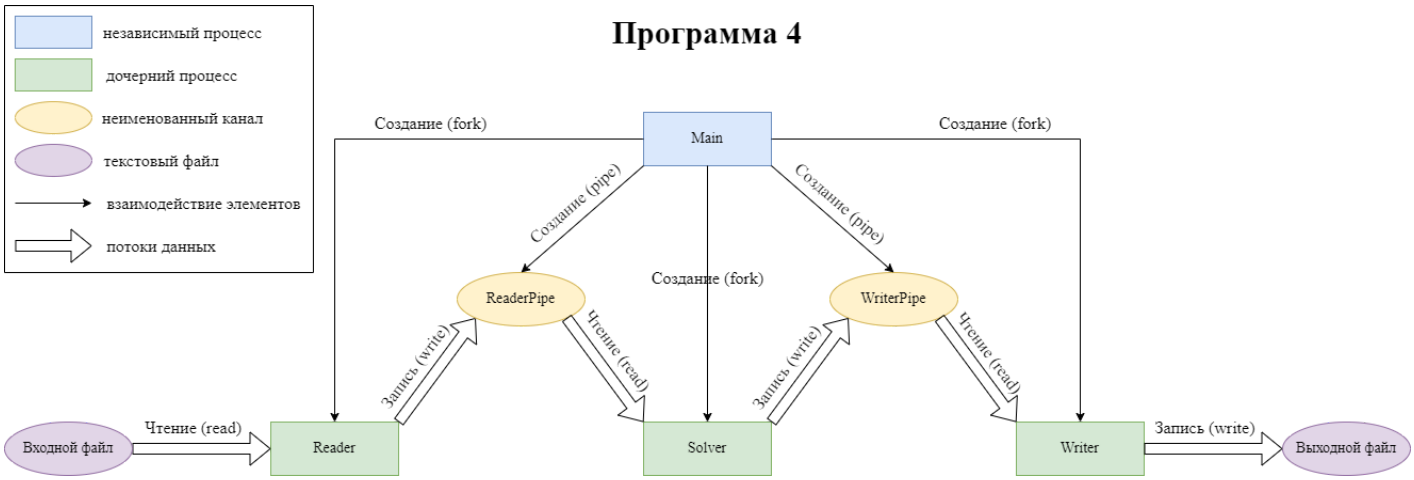
4. Программа `10` также реализует взаимодействие двух независимых процессов, поэтому её сборка состоит из двух частей аналогично программе `9` за исключением того, что вместо реализаций `lib/pipes/*` в исполняемый файл добавляются `input.c`, `solve.c` и `output.c` директории `lib/messages`

ЗАМЕЧАНИЯ ПО РЕАЛИЗАЦИИ

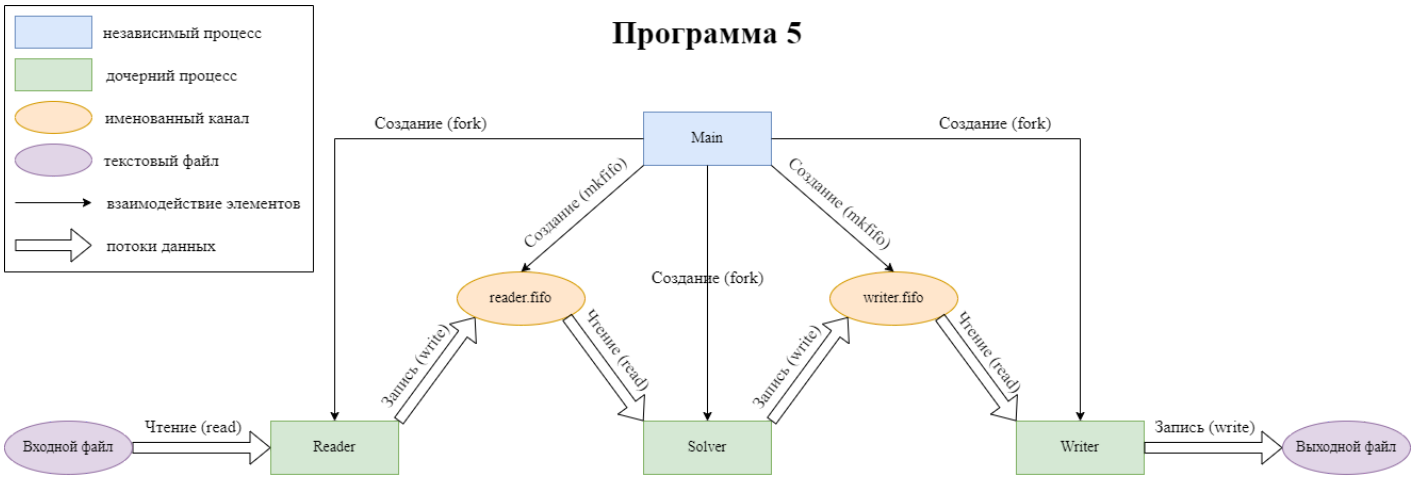
1. Программы `4` - `7` состоят из одной части, которая организует запуск нескольких дочерних процессов и обработку входных данных в соответствии с требованиями. Исходные коды программ представлены в файлах `index.c` соответствующих каталогов. При запуске программ должны указываться два аргумента командной строки: имя файла, содержащего входные данные, и имя файла для вывода результата работы соответственно.
   1.1. Программы `4` и `6` для взаимодействия процессов используют неименованные каналы, создаваемые с помощью системного вызова `pipe`.
   1.2. Программы `5` и `7` для взаимодействия процессов используют именованные каналы `reader.fifo` и `writer.fifo`, создаваемые родительским потоком с помощью функции `mkfifo` и удаляемые вызовом `unlink` при завершении работы.
2. Программы `8` - `10` состоят из двух подпрограмм, которые работают независимо. При их исполнении сначала должна быть запущена программа `io` (с исходным кодом из файла `io.c` соответствующего каталога) с указанием имени файла, содержащего входные данные, и имени файла для вывода результата работы в качестве первого и второго аргументов командной строки соответственно, а затем программа `solver` (с исходным кодом из файла `solver.c` соответствующего каталога) без аргументов командной строки.
   2.1. Программы `8` и `9` для взаимодействия процессов используют именованные каналы `reader.fifo` и `writer.fifo`, создаваемые подпрограммой `io` с помощью функции `mkfifo` и удаляемые вызовом `unlink` при завершении работы.
   2.2. Программа `10` использует очереди сообщений с номерами, на `1` и `2` превышающими номер “приватной” очереди сообщений `IPC_PRIVATE`.
3. Подробное описание алгоритмов работы программ представлено в виде комментариев в исходном коде.

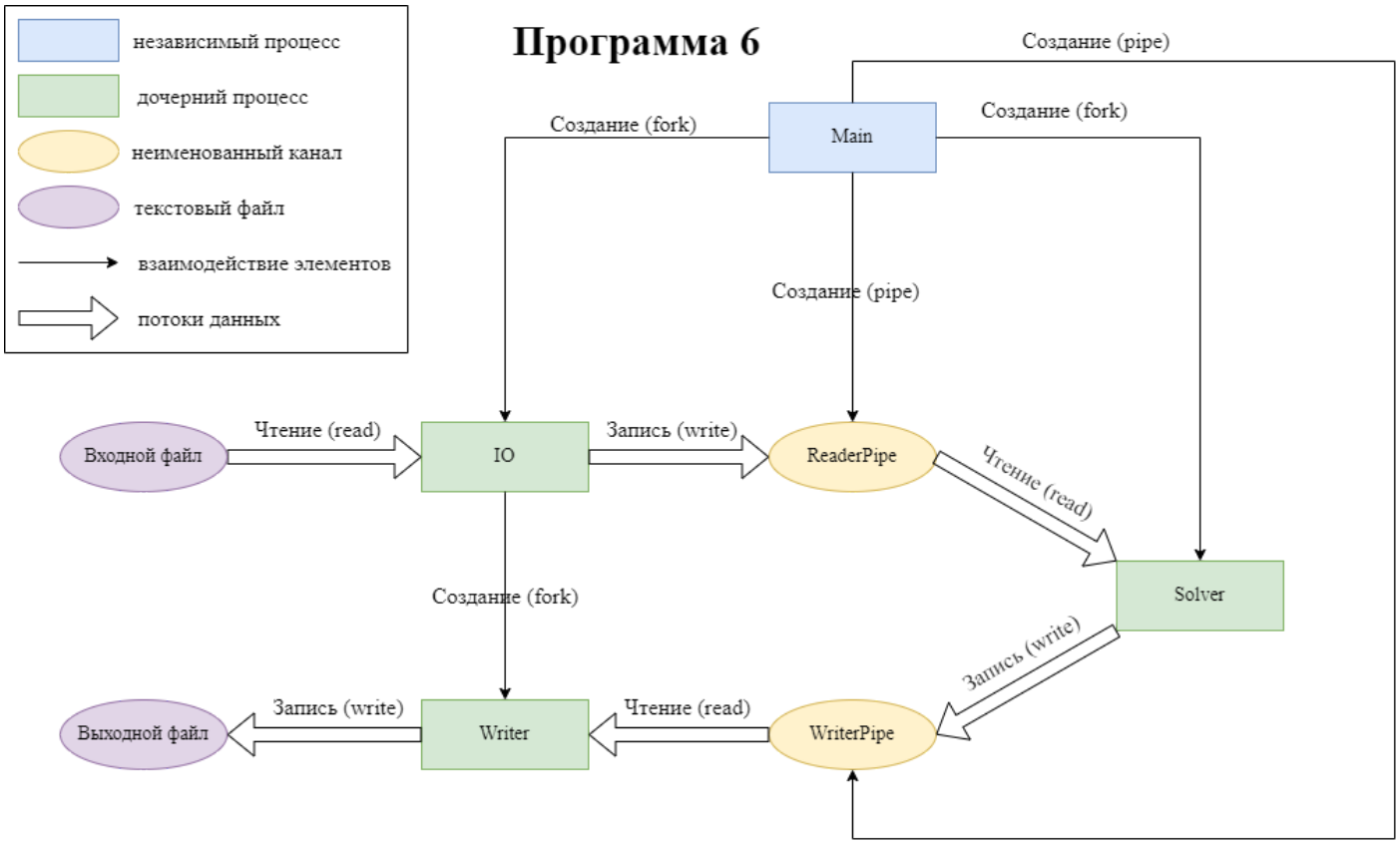
СХЕМЫ РЕШЕНИЙ

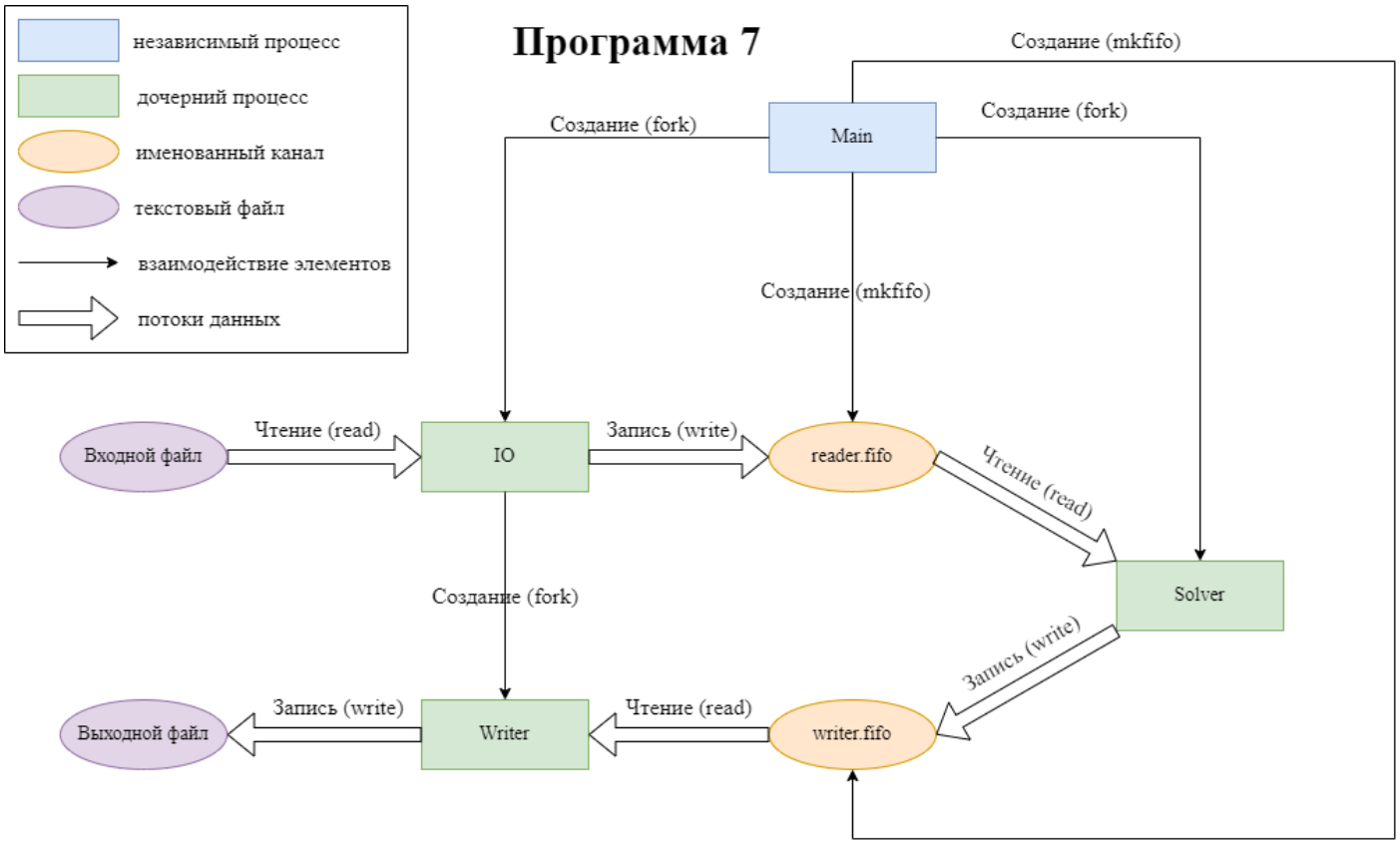
Программа 4



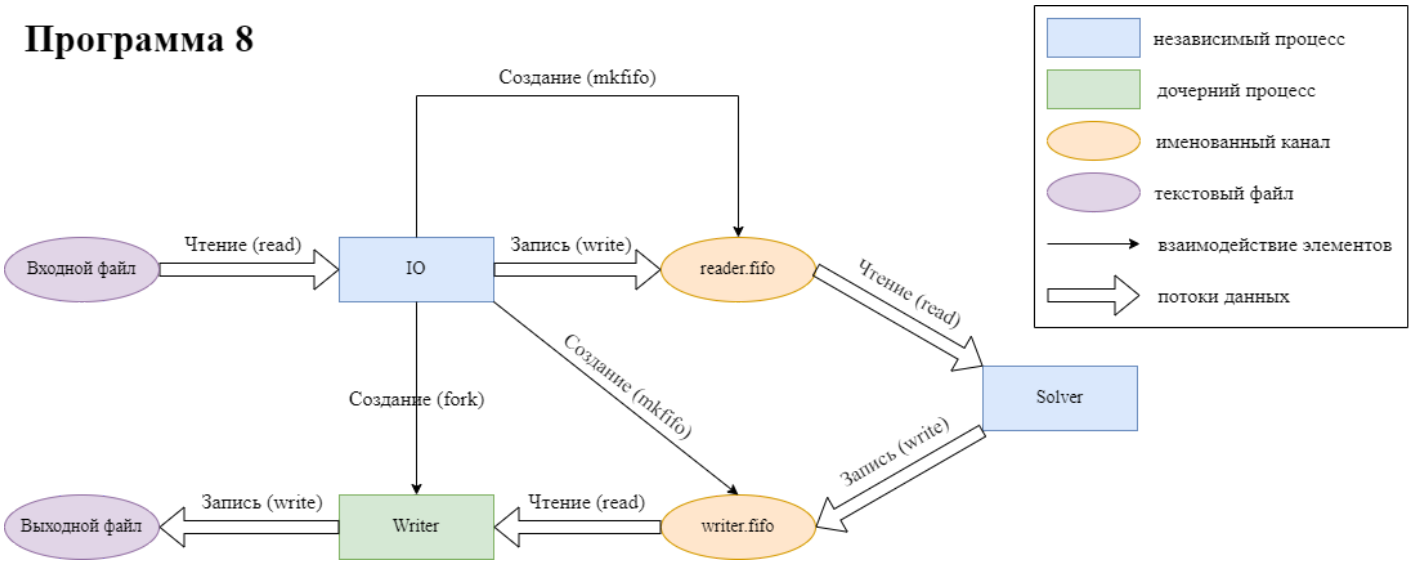
Программа 5



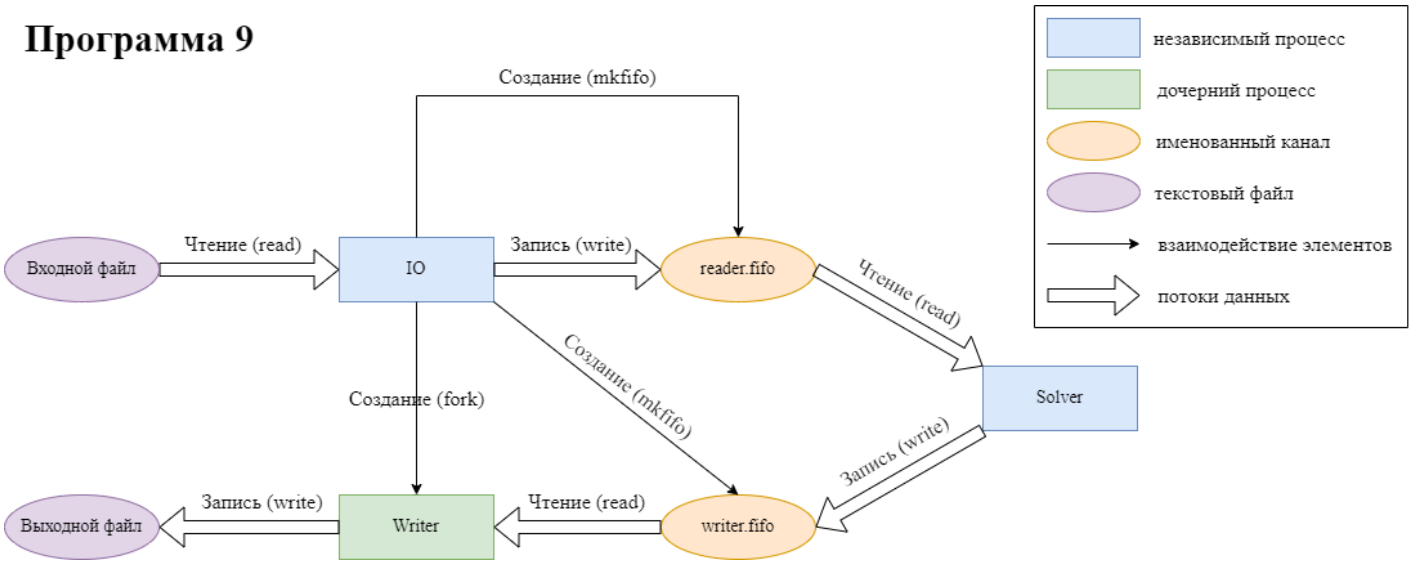

независимый процесс
дочерний процесс
неименованный канал
текстовый файл
взаимодействие элементов
потоки данных
Программа 6
Создание (pipe)
Main
Создание (fork)
Создание (fork)
Создание (pipe)
Входной файл
Чтение (read)
IO
Запись (write)
ReaderPipe
Чтение (read)
Solver
Создание (fork)
Запись (write)
Выходной файл
Запись (write)
Writer
Чтение (read)
WriterPipe


независимый процесс
дочерний процесс
именованный канал
текстовый файл
взаимодействие элементов
потоки данных
Программа 7
Создание (mkfifo)
Main
Создание (fork)
Создание (fork)
Создание (mkfifo)
Входной файл
Чтение (read)
IO
Запись (write)
reader.fifo
Чтение (read)
Solver
Создание (fork)
Запись (write)
Выходной файл
Запись (write)
Writer
Чтение (read)
writer.fifo

Программа 8



Программа 9



Программа 10