В данном пособии представлены задачи программированию на Си под управлением ОС UNIX, предлагаемые студентам в качестве практических заданий на семинарах в поддержку лекционного курса «Операционные системы».

Задание практикума №1. Свойства транслятора.

Необходимо провести для конкретного компилятора тестирование некоторых свойств из разряда тех, которые не определены жестко в языке Си, а оставлены на усмотрение реализации. Написать программу, в которой реализованы соответствующие варианту тесты, и привести их устное обоснование.

1. Выяснить способ представления типа char по умолчанию: signed- или unsigned-вариант.
Исследовать особенности выполнения операции % с отрицательными операндами.

2. Проверить наличие и качество диагностических сообщений при неверной записи и использовании констант: целых и вещественных, символьных и строковых, указателей. Проконтролировать, происходит ли конкатенация рядом расположенных строковых констант.

3. Проконтролировать, допускается ли инициализация переменных при описании; происходит ли инициализация по умолчанию.

4. Проверить, реагирует ли компилятор на попытку изменить константу (внешнюю, автоматическую, статическую; арифметических типов, строковую).

5. Проверьте, выполняется ли правило "ленивых вычислений" выражений в Си, т.е. прекращается ли вычисление выражений с логическими операциями, если возможно "досрочно" установить значение результата.

6. Выяснить способ представления типа char: signed- или unsigned- вариант.
Проверьте, все ли виды операнда операции sizeof, определяемые стандартом для арифметических типов, допускаются компилятором; действительно ли аргумент-выражение не вычисляется.

7. Выяснить способ представления типа char: signed- или unsigned- вариант.
Определите, каким образом "малое" целое расширяется до int (старшие разряды int заполняются нулями или знаковым разрядом "малого" целого).

8. Определите, каким образом расширяются unsigned- варианты "малых" целых (до unsigned int как в "классическом" Си или сначала делается попытка расширить до int, и только в случае неудачи - до unsigned int).

9. Определите, каким образом при выполнении операции присваивания и явном приведении происходит преобразование знаковых целых (M-битовое представление) к знаковым целым (N-битовое представление) при M > N, M = N, M < N.

10. Определите, каким образом при выполнении операции присваивания и явном приведении происходит преобразование беззнаковых целых (M-битовое представление) к знаковым целым (N-битовое представление) при M > N, M = N, M < N.
11. Определите, каким образом при выполнении операции присваивания и явном приведении происходит преобразование вещественных чисел (X) к знаковым целым (N-битовое представление) при $| X | < 2^{N-1}$, $| X | >= 2^{N-1}$.

12. Определите, каким образом при выполнении операции присваивания и явном приведении происходит преобразование знаковых целых X (M-битовое представление) к беззнаковым целым (N-битовое представление) при M > N, M = N, M < N; X >= 0, X < 0.

13. Определите, каким образом при выполнении операции присваивания и явном приведении происходит преобразование беззнаковых целых (M-битовое представление) к беззнаковым целым (N-битовое представление) при M > N, M = N, M < N.

14. Определите, каким образом при выполнении операции присваивания и явном приведении происходит преобразование вещественных чисел (X) к беззнаковым целым (N-битовое представление) при $0<=X< 2^{N}$, X < 0, $X >= 2^{N}$.

Задание практикума №2. Указатели и массивы.

Программа состоит как минимум из двух функций: **main()**, в которой производится чтение строк, и функции, обрабатывающей строки, полученные в качестве параметров. Результат работы этой функции должен быть возвращен в **main()**.

1. Написать функцию, определяющую, какой символ чаще других встречается в строке **str** и сколько раз он в нее входит. Если таких символов несколько, то взять первый из них по алфавиту.
Строка (может содержать и пробелы) должна быть считана в **main()** со стандартного ввода и передана в функцию как параметр. Продемонстрировать работу функции.

2. Написать функцию, определяющую, какой символ реже других (но не нуль раз) встречается в строке **str** и сколько раз он в нее входит. Если таких символов несколько, то взять первый из них по алфавиту.
Строка (может содержать и пробелы) должна быть считана в **main()** со стандартного ввода и передана в функцию как параметр. Продемонстрировать работу функции.

3. Написать функцию, которая в заданной строке меняет местами ее первую и вторую половины.
Например, **abcdefgh => efghabcd, vwxyz => yzxvw**.
Строка должна быть считана в **main()** со стандартного ввода и передана в функцию как параметр. Продемонстрировать работу функции.

4. Написать функцию, которая изменяет заданную строку следующим образом: сначала записывает все элементы с четными индексами, а затем все элементы с нечетными индексами (с сохранением их относительного порядка в каждой группе).
Например, **abcdefgh => acegbdfh, vwxyz => vxzwy.**
Строка должна быть считана в **main()** со стандартного ввода и передана в функцию как параметр. Продемонстрировать работу функции.

5. Написать функцию от двух параметров **f(a,n)**, определяющую, у скольких элементов целочисленного массива равные соседи (предыдущий и последующий элементы). Размерность массива и его элементы должны быть считаны в **main()** со стандартного ввода и переданы в функцию как параметры. Продемонстрировать работу функции.

6. Написать функцию сравнения двух строк на равенство (без учета пробельных символов).
Например, **”1 2 3 4 5 3” и “ 12 34 53 “ -> true != 0**
“123” и ” 12 0” -> false == 0
Строки должны быть считаны в **main()** со стандартного ввода и переданы в функцию как параметры. Продемонстрировать работу функции.

7. Написать функцию, которая все цифры строки **str1** записывает в начало строки **str2**, а остальные символы **str1** – в конец **str2**.
Например, **s1: “ist12d3as5?”, s2: “was” => s2: “1235wasistdas?”**
Строки (могут содержать и пробелы) должны быть считаны в **main()** со стандартного ввода и переданы в функцию как параметры. Продемонстрировать работу функции.

8. Написать функцию, которая все буквы строки **str1** записывает в начало строки **str2**, а остальные символы **str1** – в конец **str2**. Строка **str2** изменяется.
Например, **s1: “was0 78i1 2s3t5?”, s2: “das” -> s2: “wasistdas0 781 235?”**
Строки (могут содержать и пробелы) должны быть считаны в **main()** со стандартного ввода и переданы в функцию как параметры. Продемонстрировать работу функции.

9. Составьте программу удаления из строки **S1** каждого символа, совпадающего с каким-либо символом строки **S2**. Строка **S1** изменяется.
Например, **S1: “1_string1_ + string2” S2: “1_+ 2” => S2: “stringstring”**
Строки (могут содержать и пробелы) должны быть считаны в **main()** со стандартного ввода и переданы в функцию как параметры. Продемонстрировать работу функции.

10. Напишите программу, заменяющую в строке **S** все вхождения подстроки **P** на строку **Q**, например**: P = "ура"; Q = "ой"; S = "ура-ура-ура!";**
Результат: **"ой-ой-ой!"**
Строки (могут содержать и пробелы) должны быть считаны в **main()** со стандартного ввода и переданы в функцию как параметры. Продемонстрировать работу функции.

11. Написать функцию от трех параметров, определяющую, чередуются ли положительные и отрицательные элементы в целочисленном массиве из **n** элементов и вычисляющую целочисленное значение **p**. Если элементы чередуются, то **p** - это сумма положительных элементов, иначе **p** - это произведение отрицательных элементов. Размерность массива и его элементы должны быть считаны в функции **main()** со стандартного ввода и переданы в функцию как параметры. Возврат результата **p** также должен быть осуществлен через параметр функции.

12. Написать функцию от трех параметров, определяющую, упорядочены ли строго по возрастанию элементы в целочисленном массиве из **n** элементов, и вычисляющую целочисленное значение **p**. Если элементы упорядочены, то **p** - это произведение разностей рядом стоящих элементов, иначе **p** - это количество нарушений порядка в массиве.

Размерность массива и его элементы должны быть считаны в main() со стандартного ввода и переданы в функцию как параметры. Возврат результата **p** также должен быть осуществлен через параметр функции.

Задание практикума № 3. Динамические структуры.

Требуется:

1) Самостоятельно поставить перед собой задачу, для реализации которой необходимо использование динамических структур данных (например, небольшая база данных, частотный словарь, разреженный массив и т.п.). Динамические структуры данных можно использовать любые: списки любого вида, деревья, графы.
Замечание: Реализацию графа можно делать вдвоем, но каждый сдающий должен знать программу полностью.

2) Разработать динамическую структуру данных, необходимую для решения поставленной задачи.

3) **Реализовать набор функций для работы с данной структурой**:

Обязательно:

- Создание;
- Удаление;
- Выборка элемента по некоторым признакам (с контролем допустимости значения);
- Изменение элемента (с контролем допустимости значения);
- Удаление элемента;
- Вывод объекта на экран;

Дополнительно:

- Вывод на экран некоторой части объекта (часть определяется постановкой задачи);
- Сравнение двух объектов (на ==, на >-< и т.п.);
- И т.д. и т.п.

4) Обеспечить возможность работы в интерактивном режиме. Ввод новых значений, выбор следующего производимого над объектом действия должны осуществляться во взаимодействии с пользователем. Необходимо разработать и реализовать простой интерфейс.

5) Программа должна быть структурирована следующим образом: текст программы должен быть разбит как минимум на два **.c** – файла (файл, содержащий main(), и файл, содержащий определения функций для работы с объектом); прототипы функций, typedef-определения должны быть вынесены в заголовочный файл **.h** .

6) Отчет (можно в электронном виде) должен содержать:

- Постановку задачи;
- описание разработанной структуры данных;
- текст программы с комментариями

Пример задачи.

Работа с разреженным массивом.
Рассматривается двумерный массив (матрица) целых чисел. Известно, что большая часть элементов матрицы не меняют своего начального значения в процессе работы с нею (например, равны 0). Предполагается, что размерность массива известна и настолько велика, что хранить его целиком в памяти нецелесообразно. Достаточно хранить информацию об измененных элементах.
Требуется :

1) Разработать структуру данных для хранения разреженного массива;
2) Реализовать набор функций для работы с разреженным массивом:
   Обязательно:
   - Создание массива;
   - Удаление массива;
   - Выбрать элемент с указанными индексами (с контролем выхода за границы массива);
   - Изменить элемент с указанными индексами (с контролем выхода за границы массива);
   - Сумма массивов (как сумма матриц);
   - Вывод массива на экран;

   Дополнительно:
   - Вывод на экран указанной строки массива;
   - Вывод на экран указанного столбца массива;
   - Сравнение массивов на равенство;
   - Произведение матриц.
   - И т.д. и т.п.
3) Работа с массивами ведется в интерактивном режиме, необходимо разработать и реализовать простой интерфейс.
4) Программа должна быть структурирована следующим образом: текст программы должен быть разбит как минимум на два **.c** – файла (файл, содержащий main(), и файл, содержащий определения функций для работы с массивом) ; прототипы функций, typedef-определения должны быть вынесены в заголовочный файл **.h .**

Задание практикума № 4. Работа с файлами. Стандартная библиотека.

В предлагаемом задании для работы с файлами использовать только функции стандартной библиотеки. Длина строк в файле не ограничена, если в задаче не оговорено обратное.

1. Программа. Даны два непустых файла. Определить номер строки и номер символа в этой строке, где встречается первый символ, отличающий содержимое одного файла от другого. Если содержимое файлов полностью совпадает, то результат – **0, 0** и соответствующее сообщение; если один из файлов является началом другого, то результат **- n+1, 1**, где **n -** количество строк в коротком файле, и соответствующее сообщение. Имена файлов задаются в командной строке.

2. Программа. Дан файл. Определить, какие из строк чаще других встречаются в данном файле. Записать эти строки во второй файл. Длина строк не ограничена. Имена файлов задаются в командной строке.

3. Программа. Дан файл **f**. Создать два файла **f1** и **f2** следующим образом:
в файл **f1** записать в том же порядке все строки из файла **f**, состоящие только из латинских букв (прописных и строчных);
в файл **f2** – строки файла **f**, состоящие только из цифр;
все остальные строки файла **f** не записываются ни в один из этих файлов. Имена файлов задаются в командной строке.

4. Программа. Даны два файла, строки в которых упорядочены по алфавиту. Написать программу, осуществляющую слияние этих двух файлов в третий, строки которого тоже упорядочены по алфавиту. Имена всех трех файлов задаются в командной строке.

5. Программа. Дан файл и две строки. Все вхождения первой строки в файл заменить второй строкой (вхождения первой строки в качестве подстроки не рассматривать). Имя файла и строки задаются в командной строке.

6. Программа. Дан файл и строка. Все вхождения строки в файл заменить строкой с обратным порядком слов (вхождения строки в качестве подстроки не рассматривать). Имя файла и строка задаются в командной строке. Строка с пробелами при передаче ее в командной строке должна быть заключена в двойные кавычки.

7. Программа. Дан файл и две строки. Все вхождения первой строки в файл (в том числе и в качестве подстроки) заменить второй строкой. Имя файла и строки задаются в командной строке.

8. Программа. Дан файл и строка. Все вхождения строки в файл (в том числе и в качестве подстроки) заменить на удвоенные. Имя файла и строка задаются в командной строке.

9. Написать программу, центрирующую строки файла относительно середины экрана, т.е. добавляющую в начало строки такое количество пробелов, чтобы середина строки печаталась в 40-ой позиции (считаем, что обычный экран имеет ширину 80 символов и длина строк в файле не больше 80).

10. Программа. В данном файле упорядочить все строки по возрастанию их длин. Имя файла задается в командной строке.

11. Программа. В данном файле упорядочить все строки по убыванию их длин. Имя файла задается в командной строке.

12. Программа. В файле записана непустая последовательность целых чисел (целое число – это непустая последовательность десятичных цифр, возможно начинающаяся знаком + или -). Числа разделены пробелами или переводами строки. Создать новый файл, где все отрицательные числа заменены нулем.
Имя исходного файла задается в командной строке.

13. Программа. Дан файл. Изменить в файле порядок слов на обратный. Слово – последовательность символов, ограниченная пробелами, переводом строки или концом файла. Имя файла задается в командной строке.

14. Напишите программу, которая выдает на экран **n** строк файла, начиная со строки номер **m** (нумерация строк с единицы). Имя файла, **n** и **m** задаются в командной строке.

15. Программа. В командной строке передаётся имя файла. В интерактивном режиме по номеру строки в файле на экран выдаётся её содержимое. Напишите функции для "быстрого доступа" к строкам файла. Идея такова: сначала прочитать весь файл от начала до конца и смещения начал строк (адреса по файлу) запомнить в массив чисел типа *long* (точнее, *off_t*), используя функции *fgets*() и *ftell*(). Для быстрого чтения **n**-ой строки используйте функции *fseek*() и *fgets*().

Задание практикума № 5. Реализация команд UNIX.

Реализовать команду с указанным набором опций. Вызов должен происходить аналогично командам shell: > <команда> <опции> <параметры>
Опции в командной строке могут появляться в произвольном порядке и количестве.

1. Вывод информации о файлах **ls [-a –l -R] [файл]**

Если в качестве файла задано имя каталога, то выводится информация обо всех содержащихся в нём файлах, если имя файла, то выводится информация только об этом файле. По умолчанию выводится информация о файлах текущего каталога.
-a – вывести имена всех файлов (т.е. и тех, имена которых начинаются с точки);
–**l** - вывести подробную информацию о файлах. Имя файла, тип файла, права доступа, число ссылок на файл, размер файла в байтах - обязательно, имена владельца файла и владеющей группы, дата и время последнего изменения – по желанию;
–**R** – рекурсивно обойти встретившиеся подкаталоги.

2. Сортировка файлов **sort [-r +n –m -o] файлы**

Сортировать строки каждого файла в лексикографическом порядке.
sort -r файлы - сортировка в обратном порядке.
sort +n файлы – сортировать файл, начиная с **n**-ой строки.
sort -m файлы – слияние исходных (предварительно отсортированных) файлов.
sort -o выходной файл - результат направляется не на стандартный вывод (как происходит по умолчанию), а в **выходной файл**, который может совпадать с одним из исходных.

3. Сортировка файлов **sort [-c –o –n –f] файлы**

Сортирует строки каждого файла в лексикографическом порядке.
sort -c файл - проверить, является ли (единственный) исходный файл уже отсортированным.
В выходной файл ничего не записывается, на экран вывести диагностику.
Результат (код завершения) : 0 – успех, 1 – неуспех, >1 – ошибка
sort -n файлы – числовое сравнение: сортировать файл, используя для сравнения только начальные числовые цепочки строк. Эти цепочки могут содержать пробельные символы, знак минус и цифры.
sort -f файлы – при сравнении преобразовывать малые буквы в большие (т.е. сравнение букв без учета регистра).
sort -o выходной файл - результат направляется не на стандартный вывод (как происходит по умолчанию), а в **выходной файл**, который может совпадать с одним из исходных.

4. Сортировка файлов **sort [-r +n –n –o] файлы**
Сортирует строки каждого файла в лексикографическом порядке.
sort -r файлы - сортировка в обратном порядке.
sort +n файлы – сортировать файл, начиная с **n**-ой строки.
sort -n файлы – числовое сравнение: сортировать файл, используя для сравнения только начальные числовые цепочки строк. Эти цепочки могут содержать пробельные символы, знак минус и цифры.
sort -o выходной файл - результат направляется не на стандартный вывод (как происходит по умолчанию), а в **выходной файл**, который может совпадать с одним из исходных.

5. Комада **uniq [-c -d –s число] файл** - позволяет сократить до одной подряд идущие одинаковые строки.
Опции : **-c –** перед каждой выходной строкой помещать её кратность во входном файле.
-d - подавить вывод неповторяющихся строк.
-s число - при сравнении игнорировать заданное число начальных символов строки.

6. Выявление различий и совпадений текстовых файлов:
comm [-123] файл1 файл2
Читает файл1 и файл2 (их содержимое должно быть отсортировано в лексикографическом порядке) и выводит результаты работы в три столбца:
1) строки, входящие только в файл1;
2) строки, входящие только в файл2;
3) строки, входящие в оба файла.

Опции **–1, -2, -3** подавляют вывод соответствующих столбцов.
Так, **comm –12** выводит строки, общие для обоих файлов,
comm –23 - имеющиеся только в файле1, **comm –123** не выводит ничего.

Задание практикума № 6. Командный интерпретатор.

Необходимо реализовать под управлением ОС Unix интерактивный командный интерпретатор (некоторый аналог shell), осуществляющий в цикле считывание командной строки со стандартного ввода, ее анализ и исполнение соответствующих действий.
В командной строке могут присутствовать следующие операции:
указаны в порядке убывания приоритетов (на одной строке приоритет одинаков)
| , > , >> , <
&& , ||
; , &
Допустимы также **круглые скобки,** которые позволяют изменить порядок выполнения операций. Для выполнения команды в скобках создаётся отдельный экземпляр Shella.
В командной строке допустимо также произвольное количество пробелов между составляющими ее словами.

Разбор командной строки осуществляется Shellom по следующим правилам:

<Команда Shella > →
< Команда с условным выполнением >{ [**;** | **&**] < Команда Shella>}{ **;** |**&**}

<Команда с условным выполнением > →

<Команда> { [**&&** | **||**] <Команда с условным выполнением>}
<Команда> → {<перенаправление ввода/вывода>}<Конвейер> |
<Конвейер>{<перенаправление ввода/вывода>} | **(** <Команда Shella>**)**
<перенаправление ввода/вывода> →
{<перенаправление ввода > } <перенаправление вывода> |
{<перенаправление вывода>}<перенаправление ввода >
<перенаправление ввода > → '**<'** файл
<перенаправление вывода> → '**>'** файл | '**>>'** файл
<Конвейер>→ <Простая команда> {'|' <Конвейер>}
<Простая команда>→ <имя команды><список аргументов>

{X} – означает, что X может отсутствовать;
[x|y] – значит, что должен присутствовать один из вариантов : **x либо y**
| - в описании правил то же, что «ИЛИ»

pr1 | …| prN – конвейер: стандартный вывод всех команд, кроме последней, направляется на стандартный ввод следующей команды конвейера. Каждая команда выполняется как самостоятельный процесс (т.е. все pri выполняются параллельно). Shell ожидает завершения последней команды.
Код завершения конвейера = код завершения последней команды конвейера.
Простую команду можно рассматривать как частный случай конвейера.

com1 ; com2 – означает, что команды будут выполняться последовательно

com & - запуск команды в фоновом режиме (т.е. Shell готов к вводу следующей команды, не ожидая завершения данной команды **com**, а **com** не реагирует на сигналы завершения, посылаемые с клавиатуры, например, на нажатие Ctrl-C). После завершения выполнения фоновой команды не должно остаться процесса – зомби. Посмотреть список работающих процессов можно с помощью команды **ps.**

com1 && com2 - выполнить **com1**, если она завершилась успешно, выполнить **com2**;
com1 || com2 - выполнить **com1**, если она завершилась неуспешно, выполнить **com2.**
Должен быть проверен и системный успех и значение, возвращенное exit (0 – успех).

Перенаправление ввода-вывода :
< **файл** - файл используется в качестве стандартного ввода;
> **файл** - стандартный вывод направляется в файл (если файла не было - он создается, если файл уже существовал, то его старое содержимое отбрасывается, т.е. происходит вывод с перезаписью);
>> **файл** – стандартный вывод направляется в файл (если файла не было - он создается, если файл уже существовал, то его старое содержимое сохраняется, а запись производится в конец файла)
Замечание. В приведенных правилах указаны все возможные способы размещения команд перенаправления ввода/вывода в командной строке, допустимые стандартом POSIX. Shell, как правило, поддерживает лишь какую-то часть из них. Для реализации можно выбрать любой (один) вариант размещения.

Обязательный минимум (достаточный для получения 3) – это реализация конвейера, фонового режима и перенаправлений ввода-вывода.

Про моделирование фонового режима.

Основные требования, которым должен удовлетворять фоновый процесс в вашей программе:

- Он должен работать параллельно с основной программой.
  После запуска фонового процесса Шелл может запускать на выполнение следующую команду, не дожидаясь, пока фоновый процесс закончит работу.

- Он не должен реагировать на сигналы, приходящие с клавиатуры.
  Вообще таких сигналов несколько, но в вашей программе достаточно не реагировать на SIGINT (сигнал, который вызывается нажатием Ctrl-C). Сигналы с клавиатуры получают только процессы основной (не фоновой) группы. Они завершаются, а фоновые процессы продолжают работать.

- Фоновый процесс не имеет доступа к терминалу, т.е. не должен читать со стандартного ввода (это достигается перенаправлением стандартного ввода на файл устройства /dev/null, чтение из которого сразу дает EOF). Вывод на экран можно разрешить, а можно и запретить, перенаправив стандартный вывод на тот же /dev/null (вывод будет просто пропадать).

- После завершения фонового процесса не должно остаться процесса «зомби». А его не остается либо, когда родительский процесс завершается раньше, чем «сын», либо, когда в родительском процессе вызывается функция wait или waitpid..

Первый вариант моделирования фонового режима, применявшийся в шеллах до того как появились системы управления заданиями, использует сигналы.

Схема такая:

Процесс, созданный для запуска фоновой команды,
перенаправляет стандартный ввод на файл “/dev/null” – теперь при попытке чтения со стандартного ввода сразу будет получен конец файла, так что не будет конфликта чтения между основным процессом и фоновым;
вывод тоже можно перенаправить на “/dev/null”, тогда он будет просто пропадать, но можно и оставить для отладки;
устанавливает игнорирование сигнала SIGINT (signal(SIGINT,SIG_IGN));
запускает на выполнение собственно фоновый процесс.

Другой, простой способ сделать процесс фоновым (разумеется, простой для нашего случая моделирования, поскольку реально усилий требуется больше) – это выделить его в отдельную группу, фоновую.
При создании новый процесс автоматически помещается в ту же группу, что и его родительский процесс.
Поместить процесс с номером **pid** в группу с номером **pgid** можно с помощью функции
int setpgid (pid_t pid, pid_t pgid) , возвращает 0 при успехе, -1 при возникновении ошибки.
Вызов функции **setpgid(0, 0)** (в некоторых системах вызов должен быть без параметров, а функция может называться **setpgrp**) помещает текущий процесс в новую группу, номер которой становится равным номеру текущего процесса.

Чтобы не оставалось процесса-«зомби», запускать фоновую команду можно, например, следующим образом:
Основной процесс- шелл создает «сына», дожидается его окончания и считывает следующую команду.
«Сын» запускает «внука» и умирает. При этом «отцом» «внука» становится **init** (процесс с номером 1), что избавляет от возникновения «зомби» после окончания «внука».
Во «внуке» запускается уже собственно фоновая команда.

Впрочем, решать проблему «зомби» можно и другим способом, обеспечивая вызов функции **waitpid** без блокирования нужное количество раз, например, перед вводом очередной команды или при получении сигнала SIGCHLD.

Задание практикума №7. IPC.

Студенты могут выполнить одну из предложенных задач или сформулировать собственную (с использованием любого из средств IPC, и не повторяющую дословно задания из методических пособий).

1. Сервер – 2 клиента.
Сервер читает строки из файла и передает их клиентам в порядке очереди. Для передачи информации от сервера к клиентам используется
а) единственная очередь сообщений;
б) для каждого клиента своя очередь сообщений.
Клиент считывает из очереди сообщений свою строку, дописывает в её начало идентификационные данные (например, свой PID)и выводит строку в файл-результат. Вывод в файл осуществляется поочередно. Синхронизацию реализовать с помощью
а) аппарата семафоров,
б) очереди сообщений.
Очереди сообщений должны быть корректно удалены по окончании работы.

2. Основной процесс читает из файла и передает каждую строку двум процессам-обработчикам. Один из них подсчитывает количество цифр в строке, другой подсчитывает количество пробелов (варианты обработки можно менять). Результаты обработки передаются основному процессу. Тот выводит на печать номер строки и полученные результаты.
Для организации взаимодействия между процессами использовать разделяемую память и очереди сообщений.

3. Сервер – очередь клиентов.
Сервер создает и заполняет буфер в разделяемой памяти (массив из N строк). Клиенты запрашивают доступ к ресурсу (строке). С каждой строкой может работать не более одного клиента в каждый момент времени. В качестве разрешения на работу клиенту передается идентификатор разделяемой памяти и номер доступной строки. На время работы клиента строка блокируется. По окончании работы клиент «говорит спасибо» серверу, строка открывается для доступа очередного клиента.
Если все ресурсы заняты, клиент получает отказ и может повторить запрос через некоторое время (возможна организация очереди не обслуженных запросов). Результат работы клиента выводится на экран.

4. Игра «Быки и Коровы».
Сервер – клиент.
Сервер генерирует число, которое требуется отгадать за некоторое количество шагов (например, не более 10), и анализирует передаваемые ему клиентом ответы. Клиент организует интерфейс с пользователем (ввод ответов, вывод на экран сообщений).
Взаимодействие между клиентом и сервером организовать
а) через очередь сообщений,
б) через разделяемую память,
в) с помощью сокетов (1с-1к, 1с-Nк).

Цель игры - угадать четырехзначное число задуманное компьютером. Каждая цифра числа принадлежит диапазону 1- 6. Ни одна из цифр в числе не повторяется. Например, числа 1234,5361,4236. Очередной ход заключается в том, что Вы задаете некоторое четырехзначное число, компьютер сравнивает его с задуманным и сообщает результат. Сначала выдаются сообщения о количестве цифр, которые были правильно угаданы и чья позиция в указанном Вами числе совпадает с позицией в числе, загаданном компьютером (это «быки»). Затем выдается сообщение о количестве цифр, которые были правильно угаданы, но чья позиция не совпадает с позицией в числе компьютера (это «коровы»). Никакие сообщения по поводу цифр, которые не принадлежат задуманному числу, не выдаются.

5. Реализовать одну из классических задач («читатели и писатели», «о спящем парикмахере», «Обедающие философы») с использованием семафоров System V.

6. Игра «Отгадай слово».
Сервер по запросу клиента предлагает отгадать слово (выбранное из базы данных сервера, возможно с комментарием), сообщает клиенту количество букв в слове. Клиент за один ход может предложить одну букву или назвать слово целиком. Если буква содержится в слове, сервер указывает, на каких именно позициях она расположена, в противном случае сообщает об ошибке. Если слово, названное целиком, не совпадает с загаданным, игра прекращается. Количество попыток может быть ограничено. Клиент работает с пользователем-игроком в интерактивном режиме.
Взаимодействие между клиентом и сервером организовать
а) через очередь сообщений,
б) через разделяемую память,
в) с помощью сокетов (1с-1к, 1с-Nк).

7. Игра «Крестики-нолики».
Размер поля по желанию.
Состояние доски хранится в разделяемой памяти.
Доступ к доске синхронизировать с помощью:
а) Семафоров,
б) Очереди сообщений.

8. Игра «Шашки».
Реализовать игру в шашки для двух игроков. Текущее положение на доске хранить в разделяемой памяти.
Доступ к доске синхронизировать с помощью:
а) Семафоров
б) Очереди сообщений