МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
факультет вычислительной математики и кибернетики

Кузина Л.Н.
Сборник задач по практикуму: язык Си, ОС UNIX. Учебно-методическое пособие для студентов бакалавриата 2 курса, обучающихся по направлению «Фундаментальные информатика и информационные технологии». Москва, 2014.

Данное методическое пособие содержит задачи, предлагаемые на семинарских занятиях по практикуму студентам бакалавриата 2 курса специализации ФИИТ. Даются указания по решению задач, а также приводится ряд методических рекомендаций по их формулировке. Все задачи могут быть предложены как для аудиторной работы, так и для практического выполнения на ЭВМ в среде операционной системы семейства UNIX.
Пособие может быть использовано студентами для самостоятельной подготовки, а также может быть полезно преподавателям и аспирантам для проведения занятий по программированию на языке Си в среде операционной системы семейства UNIX.

Содержание

1. Аргументы командной строки, работа со строками, динамические структуры данных…….1
2. Работа с файлами……....5
3. Процессы. Каналы, перенаправление ввода/вывода. Сигналы……..…9
4. Использование средств IPC ……..……...14
5. Список литературы……...15

1. Аргументы командной строки, работа со строками, динамические структуры данных

<u>Указания к задачам раздела 1</u>

1. Функции для выделения/освобождения динамической памяти, определены в <stdlib.h>:

void * malloc (size_t size); // выделяет неинициализированный блок памяти размером size байтов, выровненный в соответствии с максимальными требованиями на выравнивание, т.е. пригодный для размещения объекта любого типа. Возвращает указатель на выделенный участок памяти, или NULL при ошибке.

void free(void * ptr); // освобождает память, на которую указывает ptr, память эта должна быть ранее выделена динамически.
Повторное освобождение памяти является ошибкой, но **free(NULL)** разрешено.

void * calloc (size_t num, size_t size); // выделяет блок памяти из **num** элементов, каждый размером **size** байтов (общий размер num*size байтов). Инициализирует память нулями. Возвращает указатель на выделенный участок памяти, или **NULL** при ошибке.

void * realloc (void * ptr, size_t size); // перераспределение памяти: освобождает память, на которую указывает **ptr**, и выделяет новую размером **size** байтов. Содержимое копируется на новое место, дополнительная новая память не инициализируется. Возвращает указатель на выделенный участок памяти, или **NULL** при ошибке.

realloc (0, size) == malloc (size) **realloc (ptr, 0)==free(ptr)**

2. Память, динамически выделенная в программе, должна быть в программе освобождена по окончании использования.

<u>Задача 1-1</u>

Написать программу. В командной строке получаем последовательность из нескольких строк. Напечатать последние три символа каждой строки. Если символов в строке меньше 3, вывести строку целиком.

<u>Указания к задаче</u>

Используйте разные способы прохода по массиву **argv**.
Подумайте, что произойдет, если попытаться считать лишний элемент из командной строки? А если с ним работать?

<u>Задача 1-2</u>

Написать программу. В командной строке получаем последовательность из нескольких целых чисел. Вычислить их среднее арифметическое и вывести его значение на экран.

<u>Указания к задаче</u>

Аргументы командной строки представляют собой Си-строки (т.е. последовательность символов с завершающим '\0' на конце).
Для преобразования строковой записи числа собственно в число можно написать

собственную функцию, а можно воспользоваться одной из стандартных.
Так, для преобразований строки в данные различных типов может быть использована функция **sscanf**, которая читает форматированные данные из строки **buf** точно так же, как функция scanf читает данные из потока стандартного ввода stdin.
int sscanf(const char *buf, const char *format [, argument] ...);

В **<stdlib.h>** определены функции для преобразования строки в число соответствующего типа (см. тип возвращаемого значения):
int atoi (const char * str);
long int atol (const char * str);
double atof (const char * str);
long double atold (const char * str);
Строка должна содержать корректную запись числа указанного типа. В противном случае возвращается 0.
Чтение строки прекращается на первом символе, который не может быть интерпретирован как часть числа.
См. также функции
long strtol(const char* start, char end, int radix);** //пропускает в начале строки start пробелы, '\n', '\t', затем читает число, останавливается на первом не входящем в представление числа символе, в end возвращается адрес этого символа, т. е. указатель за концом числа.
radix – основание системы счисления (2<=radix<=36), если radix равен 0, основание определяется как при записи констант.
Если результат не может быть представлен как число типа long int, возвращается либо значение LONG_MAX, либо значение LONG_MIN. Если параметр start не указывает на число, никакого преобразования не выполняется и функция возвращает ноль.

Для преобразований в вещественные числа служат:
double strtod(const char* start, char end);**
float strtof(const char* start, char end);**
long double strtold(const char* start, char end);**
Недостаток этих функций состоит в том, что невозможно различить случаи возврата нуля в качестве ошибки и как нормального результата преобразования строки в число 0.

Задача 1-3
Написать программу. В командной строке передаются вещественные числа и строки в произвольном порядке. Объединить строки в одну строку, а числа сложить. Результаты вывести на экран.

Указания к задаче
Подумайте, сколько понадобится памяти для новой строки, и выделите за один раз участок памяти не более необходимого максимального размера.

Задача 1-4
Напечатать таблицу значений функции на отрезке **[a,b]** с шагом **h**. Имя функции (**sin, tan, log, cos, sqrt** и т.п.) передается в командной строке. Значения **a, b, h** (типа **double**) также передаются в командной строке. Написать функцию **FunPrint**, которая выполняет действия по вычислению и печати таблицы значений для произвольной функции, указатель на которую

передается ей в качестве параметра. Вызвать **FunPrint** из **main()**.

<u>Указания к задаче</u>

sin и др. математические функции описаны в заголовочном файле **<math.h>**. Если в программе используются эти функции, необходимо при компиляции подключить математическую библиотеку, опция **-lm** .

<u>Задача 1-5</u>

Даны 2 списка **L1** и **L2**, элемент данных – строка (char *). Написать функцию, которая выполняет объединение двух списков без дублирования одинаковых элементов.
Параметры функции — указатели на начала списков,
возвращаемое значение — указатель на список-результат.
Варианты:
1.
а) Добавить к **L1** только те элементы **L2**, которые не содержатся в **L1**. Из **L1** дубликаты не удалять. **L2** не менять.
б) Создать новый список, в который включить только различающиеся элементы **L1** и **L2**. Исходные списки не менять.
в) Создать новый список, в который включить только различающиеся элементы **L1** и **L2**. Использовать непосредственно элементы исходных списков. Не включенные в новый список элементы должны быть освобождены.
2.
а) Списки однонаправленные.
б) Списки двунаправленные.

<u>Указания к задаче</u>

Не забывайте, что в последнем элементе списка указатель на следующий элемент должен быть равен NULL. В двунаправленном списке кроме этого, указатель на предыдущий элемент в первом элементе списка тоже должен быть равен NULL.
При удалении элемента списка не забывайте освобождать память, которую выделяли для элемента данных.

<u>Задача 1-6</u>

Дан двунаправленный список L, элемент данных char*. Написать функцию удаления из списка элемента с заданным порядковым номером. Если номер больше длины списка, ничего не удалять.

<u>Задача 1-7</u>

Дан двунаправленный список L, элемент данных int. Написать функцию вставки элемента в список после элемента с заданным порядковым номером. Если номер равен 0, вставить элемент в начало списка. Если номер больше длины списка, список не менять.

<u>Задача 1-8</u>

Написать функцию чтения строки со стандартного ввода. Длина строки неизвестна, считаем, что любой приготовленный буфер может оказаться мал. Системные ограничения не рассматриваем. Функция возвращает указатель на введенную строку или NULL в случае ошибки.

Указания к задаче

Реализовать два варианта чтения строки:

1) с помощью списков;
2) с помощью перераспределения памяти.

Дополнительную память выделять блоками, длина которых кратна 4 (4, 8, 16,...). Для тестирования лучше взять небольшой блок, например, 4 байта.

Запрещено использовать функцию realloc для добавления памяти по 1 байту, т.к. это крайне неэффективно.

2. Работа с файлами

Указания к задачам раздела 2

1) Для задач данного раздела можно указывать в качестве условия требование решить задачу, используя для работы с файлами:

а) только стандартные функции языка Си.

б) системные функции ОС UNIX.

2) Ограничения на размер файла (или их отсутствие), допустимость считывания всего файла в память, допустимость создания временных вспомогательных файлов также можно задавать как дополнительные условия.

3) Наличие символа перевода строки '\n' в конце последней строки текстового файла не является обязательным требованием, хотя некоторые текстовые редакторы добавляют его автоматически. Тем не менее, программы должны корректно работать и при наличии '\n' в конце файла, и при его отсутствии.

4) В программах должны корректно обрабатываться случаи передачи неверного количества параметров, неудачного открытия файлов и т.п.

Задача 2-1

Даны два файла, содержащие последовательности целых чисел (не в текстовом виде), упорядоченных по возрастанию. Объединить их в один файл, также упорядоченный по возрастанию. Имена исходных файлов и файла, в который должен быть записан результат, передаются в командной строке.

Задача 2-2

В командной строке передается имя файла и символ. Удалить из файла все строки, содержащие (не содержащие) заданный символ.

а) Длина строки в файле ограничена.

б) Длина строки в файле не ограничена.

Задача 2-3

Удалить из файла все строки, длина которых больше (меньше, равна) заданного числа n. Имя файла и число n передаются в командной строке. Длина строк в файле не ограничена.

Задача 2-4

а) Написать функцию, которая читает со стандартного ввода по слову и создает список слов. Конец чтения фиксируется, когда достигнут перевод строки или конец файла.
Возвращает указатель на начало списка.

б) Сделать то же, но для чтения из заданного файла, имя файла передается в командной строке. Файл содержит последовательность слов, читать слова только из строки файла с заданным номером. Если строки с таким номером нет, вернуть NULL.

Написать программу, в которой вызывается эта функция (эти функции) и полученный список слов выводится на печать в обратном порядке.

Указания к задаче

Слово — последовательность символов, ограниченная пробелами, символом перевода строки, началом или концом файла.

Обратите внимание, что в обратном порядке выводятся именно слова, а не символы строки.

Обеспечьте корректную работу при работе с пустым файлом, с пустой строкой.

По окончании работы освободите всю динамически выделенную память.

Задача 2-5

Для каждого режима открытия файла, которые используются функцией **fopen()** (“w”, ”r”, ”a”, “w+” , “r+”, “a+”), определить значение второго параметра в вызове функции **open()**, задав набор флагов, обеспечивающий аналогичный режим открытия.

Задача 2-6

Программа. В командной строке передается имя файла и число **N**. Если длина файла не больше 1024 байт, то продублировать в этом файле каждую группу из **N** байт (остаток <**N** байт в файл записать, но не дублировать).
Если длина файла больше 1024, файл не изменять.

Указания к задаче

Длину файла можно узнать
а) с помощью функции **stat()**, определив значение поля **st_size** в **struct stat**. При этом не требуется открывать файл.
б) с помощью функции **lseek(),** в этом случае файл должен быть предварительно открыт.

Задача 2-7

Написать программу слияния двух файлов в третий. Файл-результат формируется чередованием **N**-ок символов первого и второго файлов. Если один из файлов длиннее другого, то оставшаяся часть приписывается в конец файла-результата. Имена файлов и число **N** заданы в командной строке. Использовать системные функции для работы с файлами.

Задача 2-8

Программа. Дан файл и строка. Все вхождения строки в файл (в том числе и в качестве подстроки) удалить. Имя файла и строка задаются в командной строке. Строка с пробелами при передаче ее в командной строке должна быть заключена в двойные кавычки.

Задача 2-9

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. Строка с пробелами при передаче ее в командной строке должна быть заключена в двойные кавычки.

Задача 2-10

Программа. Дан файл. Имя файла задается в командной строке. Все строки файла продублировать.

Пример работы:	Исходный файл:	Измененный файл:
	str1	str1
	str2	str1
		str2
		str2

Задача 2-11

Программа. Дан файл. Имя файла задается в командной строке. Поменять в файле местами каждую пару строк. Длина строки неограниченна.

Пример работы:

Исходный файл:	Измененный файл:
str1	str2
str2	str1
str3	str4
str4	str3
str5	str5

Задача 2-12

Написать программу. В командной строке передается имя файла и символ.
Удалить из файла строки, начинающиеся с заданной буквы. Длина строки неограниченна.

Задача 2-13

Написать программу. В командной строке передается имя файла и символ.
Удвоить в файле строки, начинающиеся с заданной буквы. Длина строки неограниченна.

Задача 2-14

Программа. В командной строке передается имя файла. Если у файла есть только одно имя (т.е. одна жесткая ссылка), преобразовать файл, «перевернув» его (1-ый символ становится последним, 2-ой — предпоследним и т.д.) без использования вспомогательных файлов. Запрещается также считывать весь файл в память.

Указания к задаче

Количество жестких ссылок на файл можно получить с помощью функции **stat()**, определив значение поля **st_nlink** в **struct stat.**

Задача 2-15

Написать программу. В командной строке передается имя файла и два числа **n** и **m**. Вывести на экран **m** строк файла, начиная с **n**-ой строки. Если строк в файле меньше **n**, ничего не делать. Если не меньше **n**, но меньше **n+m-1**, выводить нужные строки до конца файла.

Задача 2-16

Реализовать команду UNIX **cp [-ip] исходный_файл целевой_файл** // копирование файла
Копируется содержимое исходного файла (или указуемого исходным, если последний представляет собой символьную ссылку) в целевой.
Копирование в себя является ошибкой (должно отслеживаться).
Замечание: Копирование осуществлять блоками, а не по 1 байту.
Опции:
-i – запрос подтверждения перед попыткой копирования в существующий файл,
-p – вызывает копирование не только содержимого, но и атрибутов исходного файла (режима, идентификаторов владельца и владеющей группы, времени последнего доступа и изменения).
Достаточно реализовать копирование режима доступа.
Порядок опций при задании команды может быть любым, и любая из них (в том числе и обе) может отсутствовать.

Указания к задаче

Однозначно идентифицируют файл пара – номер устройства и номер индексного дескриптора.

Для изменения режима доступа использовать одну из функций:
int chmod (const char* path, mode_t mode);
int fchmod (int filedes, mode_t mode);
Необходимо подключить заголовочные файлы <sys/types.h> и <sys/stat.h>.

Задача 2-17

Написать программу, работающую подобно команде **more**, которая осуществляет вывод файла на экран порциями по нажатию клавиши.
more [-n <целое_число>] [-s] <список файлов>
Опции:

> **-n <целое число>** - целое число задает количество выводимых за один раз строк. Длина одной порции по умолчанию, т.е. в том случае, если эта опция не задана, 10 строк.
>
> **-s –** предписывает заменять последовательность пустых строк одной пустой строкой.

Порядок опций при задании команды может быть любым, и любая из них (в том числе и обе) может отсутствовать.
Для каждого файла из списка отсчет строк начинать заново.

3. Процессы. Каналы, перенаправление ввода/вывода. Сигналы

Задача 3-1

Процесс-«отец» создает «сына». Процесс-«сын» создает еще 2 процесса («внуков» для исходного), которые выполняют некоторые произвольные действия. Условие: «отец» должен завершиться только после того, как завершат существование все его «внуки».

Указания к задаче

Получить информацию о завершении процесса может только его родительский процесс, а если родительский процесс уже закончил работу, то процесс init (№1).

Задача 3-2

В командной строке передаются имена 2-ух исполняемых файлов, для каждого по 2 параметра, т.е. командная строка выглядит так:

>./a.out prog1 arg11 arg12 prog2 arg21 arg22

а) Запустить эти программы для параллельного исполнения.

б) Запустить эти программы для последовательного исполнения.

Определить, завершились ли они успешно (с точки зрения программы, с точки зрения системы).

По окончании работы не должно остаться процессов «зомби».

Указания к задаче

Напишите тестовые программы prog1 и prog2 (достаточно одной). Они должны работать не слишком быстро и осуществлять вывод на экран. Примерно так:

```
int main (int argc, char *argv[])
{
  int i=0;
  printf("my arguments are: %s %s\n",argv[1],argv[2]);
  while(i<15){
          printf("my pid = %d\n",getpid());
          i++;
           }
  return 0;
}
```

Чтобы определить успешность завершения порожденного процесса, нужно исследовать значение, возвращенное через аргумент status функцией wait (или waitpid). В случае успешного завершения оба младших байта в status будут равны 0.

Задача 3-3

В командной строке передаются имена исходного файла и файла-результата. Процесс выбирает из 1-ого файла четные строки и передает их для сортировки командой **sort.** Результат сортировки должен оказаться во 2-ом файле.

Команда **sort** при вызове без параметров читает стандартный ввод до появления EOF, сортирует введенные строки в лексикографическом порядке и выводит результат на стандартный вывод.

Задача 3-4

Протестировать работу с каналами. Определить, что происходит, если данных в канале

находится меньше, чем запрашивается функцией **read**: блокируется ли **read** или возвращается, считав имеющееся?

Задача 3-5

Двустороннее взаимодействие с использованием каналов.
В командной строке передается имя файла. Процесс-отец создает «сына», открывает файл и передает «сыну» по строке. «Сын» читает строку, реверсирует ее, оставляя на месте '\n', если он есть, и возвращает измененную строку «отцу». Тот читает ее и выводит в файл. В результате должен быть изменен исходный файл.

Указания к задаче

Необходимо использовать 2 канала. Почему?

Задача 3-6

Реализовать игру в «ping-pong» 3 процессов через каналы. Главный процесс порождает двух потомков по любой схеме (2 сына или сын-внук) и начинает передачу данных. Начальное и конечное целые числа переданы в командной строке. Принимающий увеличивает число на 1 и передает дальше, и так по кругу до тех пор, пока число не превысит заданный максимум. Каждый процесс выводит на экран текущее число и свой pid.

Задача 3-7

Программа. В командной строке передается имя файла. Длина строк в файле не больше MAXLEN. Работают два процесса : «отец» выводит на экран № строки, а «сын» печатает очередную строку из файла. Вывод происходит строго поочередно.
Синхронизировать работу с помощью
а) сигналов, б) каналов.
Работа завершается, когда файл исчерпан, либо через 3 секунды после начала работы «сына» (если файл слишком велик).
По окончании работы не должно остаться процесса «зомби».

Задача 3-8

Написать программу, которая после запуска не выполняет никаких видимых действий, но и не завершается; при нажатии ^C печатает сообщение; при нажатии ^C дважды с интервалом менее 2 секунд – завершает работу.

Указания к задаче

По нажатию клавиш Ctrl-C (^C) генерируется сигнал SIGINT.
Для того, чтобы определить, прошел ли заданный интервал времени, воспользуйтесь функцией int **alarm**(unsigned int sec), которая возбуждает сигнал SIGALRM через sec секунд. Сигналы SIGINT и SIGALRM по умолчанию вызывают завершение процесса.

Задача 3-9

Программа. Подсчитать, сколько поступило (перехвачено) сигналов SIGTRAP между 2-ым и 4-ым нажатием ^C. В остальное время работы процесса игнорировать SIGTRAP. Закончить работу по 6-ому нажатию ^C.

Указания к задаче

Сигнал SIGTRAP используется при трассировке процессов при отладке, по умолчанию

вызывает завершение процесса.

Задача 3-10

В командной строке заданы имя файла и число N. Написать программу, выводящую на экран содержимое файла порциями по N строк.
Каждая последующая порция выдаётся после
а) нажатия клавиш Ctrl-C,
б) через 2 секунды после предыдущей.

Задача 3-11

Программа. Процесс в командной строке получает имя файла. Длина строк в файле ограничена. Создает 2 сыновей. Сыновья читают по строке из файла и выводят их на печать строго поочередно.
Синхронизировать работу сыновей через отца с помощью а) сигналов, б) каналов.

А) файл каждый сын открывает самостоятельно, т.е. дескрипторы будут независимы.
Б) файл открывает отец, т.е. дескрипторы связаны.

Варианты вывода:
а) каждый выводит ВСЕ строки файла,
б) 1-ый выводит нечетные строки, а 2-ой – четные.

Задача 3-12 Программа. В командной строке передается имя файла . Процесс создает сына, затем отец и сын поочередно читают по слову со стандартного ввода и пишут их в файл. Порядок слов должен сохраняться. Синхронизировать работу с помощью сигналов (каналов). Не должно остаться процесса «зомби». Работа завершается через 120 секунд либо по нажатию ^C.

Задача 3-13 В командной строке передаются имена двух исполняемых файлов. Запустить эти программы на выполнение так, чтобы вывод первой программы поступал на ввод второй (как в конвейере). Должно выполняться условие: если 2ая программа закончится раньше 1ой, 1ая должна продолжать работать, и ее вывод должен идти в стандартный вывод. Если раньше закончится 1ая, 2ая доделывает свою работу и тоже благополучно завершается.

Указания к задаче

Нужен 3-ий процесс, который организует взаимодействие 1ого и 2ого.
Для того, чтобы узнать, что 2ая программа завершилась, можно
а) использовать перехват сигнала SIGCHLD. По умолчанию этот сигнал не оказывает влияния на родительский процесс.
б) использовать перехват сигнала SIGPIPE. По умолчанию этот сигнал завершает процесс. Сигнал SIGPIPE будет послан процессу, который пытается писать в канал, все дескрипторы на чтение для которого закрыты.
Для отладки можно использовать 1ый процесс, который, например, читает из файла и пишет прочитанное в стандартный вывод, и 2ой процесс, который читает со стандартного ввода определенное число строк (или символов) и пишет эти строки, например, в стандартный вывод. Рассмотреть случаи, когда количество строк в файле больше, чем ожидается 2-ым процессом, и меньше этого.

Задача 3-14

Реализовать игру в «ping-pong» 3 процессов через один канал.
Главный процесс порождает двух потомков по любой схеме (2 сына или сын-внук) и начинает передачу данных. Начальное и конечное целые числа переданы в командной строке.
Принимающий увеличивает число на 1 и передает дальше, и так по кругу до тех пор, пока число не превысит заданный максимум.
Каждый процесс выводит на экран текущее число и свой pid.
Синхронизация обращений к каналу должна быть организована с помощью сигналов.

Задача 3-15

Протестировать работу сигналов в используемой Вами системе.

2. Сбрасывается ли обработка сигнала на значение по умолчанию после вызова обработчика, если нет явной переустановки?
3. Обработка сигнала SIGCHLD: возникает ли процесс «зомби», если этот сигнал не перехватывается, игнорируется, перехватывается. Вообще, позволяет ли система явное игнорирование SIGCHLD?
4. Блокируется ли на время работы обработчика

а) тот же сигнал, что в данный момент обрабатывается, б) другие сигналы?

Задача 3-16

Программа. В командной строке имена 3 файлов: f1, f2, f3.
Процесс создает двух потомков, возможны два варианта:
а) работают отец и два сына,
б) работают отец, сын и внук.
1ый потомок читает из f1, 2ой потомок из f2 - по строке и строго поочередно пишут эти строки в f3. Синхронизация вывода в файл должна быть организована
а) с помощью сигналов,
б) с помощью каналов.
Работу следует завершить, когда закончится хотя бы один файл.
Процесс, у которого файл еще не закончился, должен сообщить отцу, сколько строк осталось необработанными, а отец выводит эту информацию на экран.
Файлы должны быть корректно закрыты по завершении работы.

4. Использование средств IPC

Указания к задачам раздела 4

Для отладки программ этого раздела могут оказаться полезными команды
ipcs – вывод информации обо всех объектах IPC в системе,
ipcrm – удаление объекта IPC по ключу либо по идентификатору.

Задача 4-1
Использование IPC. Написать программу создания ресурсов, а также две программы Производитель и Потребитель.
Производитель заполняет буфер в разделяемой памяти, а Потребитель читает из него. Производитель должен помещать данные в буфер только после того, как Потребитель прочитает эти данные, а Потребитель может читать данные только после того, как Производитель закончил обновление данных. Использовать для синхронизации аппарат очередей сообщений.

Указания к задаче
Общую память использовать а) для хранения строки ограниченной длины,
б) для хранения целого числа.
Процесс чтения-записи продолжается определенное число раз и завершается по инициативе
а) Производителя,
б) Потребителя.
Используются сообщения трех разных типов, тело сообщений возможно пустое, т.к. важен в данном случае лишь факт прихода сообщения.
Удаление очереди сообщений и разделяемой памяти выполняется
а) Производителем (в этом случае требуется синхронизация для гарантии того, что Потребитель закончил работу),
б) вручную по завершении работы процессов.

Задача 4-2
Решите задачу1, используя для синхронизации аппарат семафоров.

Задача 4-3
Семафоры. Имеется разделяемый ресурс, например, файл. Начальные данные в него уже записаны. Есть один писатель, который периодически в файл пишет, и много читателей, которые файл читают. Читатели обращаются за доступом к ресурсу в произвольные моменты времени и могут читать одновременно.
Организовать синхронизацию с помощью аппарата семафоров, если
А) Приоритет имеют читатели, т.е. писатель может писать только тогда, когда нет желающих читать файл.
Б) Приоритет у писателя. Если писатель готов писать, то уже читающие дочитывают, но новые читатели уже не могут присоединяться к чтению, а должны дождаться обновления данных.

Задача 4-4
Приведены фрагменты трех программ, использующих семафоры.
Предполагается, что все системные вызовы проработали успешно.

Вывод на экран работает атомарно и без буферизации, **semd** – идентификатор массива семафоров, состоящего из **2** семафоров.
Оба семафора проинициализированы **1**.
Выписать все варианты вывода, возможные при выполнении приведенных фрагментов программ тремя процессами (A, B,C).

а)

Процесс A	Процесс B	Процесс C

`struct sembuf sb[2];`
`sb[0].flg = 0;`
`sb[1].flg = 0;`
 	
`sb[0].num = 0;`
`sb[1].num = 0;`
 	
`sb[0].op = 0;`
`sb[1].op = 1;`
 	
`semop(semd,&sb,2);`
`fprintf(stderr,"A\n");`
 	
`sb[0].op = -1;`
`semop(semd,&sb,1);`
 	
`sb[1].num = 1;`
`sb[1].op = 1;`
`semop(semd,&sb[1],1);`
	`struct sembuf sb[2];`
	`sb[0].flg =0;`
	`sb[1].flg = 0;`
	
	`sb[0].num = 0;`
	`sb[1].num = 0;`
	
	`sb[0].op = 0;`
	`sb[1].op = 1;`
	
	`semop(semd,&sb,2);`
	`fprintf(stderr,"B\n");`
	
	`sb[0].op = -1;`
	`semop(semd,&sb,1);`
	
	`sb[1].num = 1;`
	`sb[1].op = 1;`
	`semop(semd,&sb[1],1);`
		`struct sembuf sb[2];`
		`sb[0].flg = 0;`
		`sb[1].flg = 0;`
		
		`fprintf(stderr,"C\n");`
		`sb[0].num = 0;`
		`sb[1].num = 1;`
		
		`sb[0].op = -1;`
		`sb[1].op = -1;`
		
		`semop(semd,&sb,2);`
		
		`sb[1].op = -1;`
		
		`semop(semd,&sb[1],1);`
		
		`fprintf(stderr,"D\n");`

б)

Процесс A	Процесс B	Процесс C

`struct sembuf sb[2];`
`sb[0].flg = 0;`
`sb[1].flg = 0;`
 	
`sb[0].num = 0;`
`sb[1].num = 0;`
 	
`sb[0].op = 0;`
`sb[1].op = 1;`
 	
`semop(semd,&sb,2);`
`fprintf(stderr,"A\n");`
 	
`sb[0].op = -1;`
`semop(semd,&sb,1);`
 	
`sb[1].num = 1;`
`sb[1].op = 1;`
`semop(semd,&sb[1],1);`
	`struct sembuf sb[2];`
	`sb[0].flg =0;`
	`sb[1].flg = 0;`
	
	`sb[0].num = 0;`
	`sb[1].num = 0;`
	
	`sb[0].op = 0;`
	`sb[1].op = 1;`
	
	`semop(semd,&sb,2);`
	`fprintf(stderr,"B\n");`
	
	`sb[0].op = -1;`
	`semop(semd,&sb,1);`
	
	`sb[1].num = 1;`
	`sb[1].op = 1;`
	`semop(semd,&sb[1],1);`
		`struct sembuf sb[2];`
		`sb[0].flg = 0;`
		`sb[1].flg = 0;`
		
		`fprintf(stderr,"C\n");`
		`sb[0].num = 0;`
		`sb[1].num = 1;`
		
		`sb[0].op = -1;`
		`sb[1].op = -1;`
		
		`semop(semd,&sb,2);`
		
		`sb[1].op = -2;`
		
		`semop(semd,&sb[1],1);`
		
		`fprintf(stderr,"D\n");`

5. Список литературы

1. Б.Керниган, Д.Ритчи. Язык программирования Си. Издание 3-е, исправленное. Санкт-Петербург, Невский Диалект, 2001.
2. А.Робачевский. Операционные системы UNIX. 2-е изд., перераб. и доп. Санкт-Петербург, BHV-Санкт-Петербург, 2007.
3. У.Стивенс. UNIX: взаимодействие процессов. Санкт-Петербург, Питер, 2002.
4. Н.В.Вдовикина, И.В.Машечкин, А.Н.Терехин, А.Н.Томилин. Операционные системы: взаимодействие процессов. Москва, Макс-Пресс, 2008
5. Н.В.Вдовикина, И.В.Машечкин, А.Н.Терехин, В.В.Тюляева. Программирование в ОС UNIX на языке Си. Москва, Макс-Пресс, 2009
6. Руденко Т.В. Сборник задач и упражнений по языку Си. Москва, МГУ, 1999.