Отчет по индивидуальному домашнему заданию №4 по дисциплине “Архитектура вычислительных систем”.
Владимиров Дмитрий Андреевич, БПИ218.
Вариант 4.

Условие. Задача о читателях и писателях. Базу данных разделяют два типа процессов – читатели и писатели. Читатели выполняют транзакции, которые просматривают записи базы данных, транзакции писателей и просматривают и изменяют записи. Предполагается, что в начале БД находится в непротиворечивом состоянии (например, если каждый элемент — число, то они все отсортированы). Каждая отдельная транзакция переводит БД из одного непротиворечивого состояния в другое. Для предотвращения взаимного влияния транзакций процесс-писатель должен иметь исключительный доступ к БД. Если к БД не обращается ни один из процессов-писателей, то выполнять транзакции могут одновременно сколько угодно читателей. Создать многопоточное приложение с потоками-писателями и потоками-читателями. Реализовать решение, используя семафоры.

Аннотация. Для ввода данных из консоли необходимо передать в параметры командной строки слово console, для ввода из файла – передать имя файла, а для рандомной генерации – ничего не передавать. Корректные входные данные – это два неотрицательных числа, записанные через пробел – количество читателей и писателей соответственно. В остальных случаях, если входные данные некорректные или если не открывается файл, программа напишет что-то вроде wrong input или wrong filename и завершит работу.

На 4 балла:
1. Условие задачи приведено.
2. Модель параллельного программирования, подходящая под условие задачи – взаимодействующие равные. Главный поток, не занимающийся непосредственными вычилениями, исключен. Распределение работа происходит либо фиксированно, либо динаминчески. Имеется разделенная переменная, доступ к которой одновременно есть либо у одного писательского потока, либо у нескольких читательских.
3. Входные данные – количество читателей и количество писателей, которые в рандомный момент времени могут подключаться к библиотеке и выполнять рандомные операции.
4. Приложение реализовано (файл main.cpp), в качестве синхропримитива используются семафоры, потому что это просят в условии.
5. Ввод данных с консоли доступен при вводе в командую строку параметра console. (запуская командой > ./main console)

На 5 баллов:
1. Комментарии, поясняющие выполняемые действия и описание используемых переменных, присутстсвуют.

2. Предполагается реализовать следующий сценарий: есть библиотека (в программе – база данных в виде массива на 11 элементов для простоты), у этой библиотеки соответственно есть читатели и писатели. Необходимо правильно распределить доступ к ресурсам это базы данных. Читатели могут только просматривать записи, а писатели могут их корректировать. В связи с этим накладываются следующие ограничения: одновременно может быть открыт доступ к библиотеке либо только одному писателю, либо скольким угодно читателям. То есть если, к примеру, писатель на данный момент делает какие-то поправки в базе, то все остальные писатели и читатели не смогут подключиться, пока он не закончит. А если, к примеру, один или несколько читателей на данный момент пользуются ресурсами библиотеки, то в нее могут зайти еще читатели, а все писатели ждут, пока все читатели не закончат свое пользование, после чего один из них сможет подключиться.

На 6 баллов:

1. Для начала хотелось бы описать два вспомогательных семафора, не связанных напрямую с решением задачи распределенного доступа к библиотеке. 1. семафор information_output. В нашей задаче несколько читательских потоков могут работать одновременно, а каждый поток должен что-то выводить в файл и в консоль. Потоки вывода представлены в единственных экземплярах, следовательно может возникнуть борьба за ресурсы и потоки будут перехватывать власть после очередного << в std::cout к примеру. В таком случае в выводе хоть и будет все необходимое, да вот только в хаотичном порядке, для чего мы и пользуемся этим семафором. При необходимости какому-либо потоку вывести что-либо сначала семафор блокируется sem_wait(&information_access), затем выводится все необходимое, а после сразу же семафор освобождается sem_post(&information_access). Таким образом, если кто-то на данный момент управляет потоком вывода, то ни один другой поток не сможет ничего туда вывести, пока текущий поток не освободит этот самый поток вывода. Все это будет происходить за пренебрежительно короткое время по сравнению с пользованием (оно имитируется при помощи usleep() и может достигать местами 5 секунд), поэтому практически никакого влияния на работу потоков оказываться не будет, зато и информация будет выводиться как положено. 2. семафор readers_changing. Данный семафор блокирует возможность смены количества активных читателей, а после разблокирует ее. Проблема в том, что если не использовать этот семафор, то сторонний поток может влезть после проверки (if (readers_count) == 1), полностью отработать, на этот момент у него так же будет 1 действующий поток, заблокировать доступ к базе данных как первый входящий читатель и спровоцировать дедлок, из-за того что текущий поток все-таки войдет в ветку условного оператора, т.к. на момент проверки все действительно было так.

2. Доступ к базе данных осуществляет семафор database_access. С писателями-потоками все просто. Они лишь запрашивают доступ при помощи вызова sem_wait(&database_access), ждут пока текущий писатель закончит свои действия, либо все читатели закончат пользоваться, и затем спокойно может блокировать доступ и никому его не отдавать пока не поменяет что ему нужно.
3. С читателями же все сложнее. Для начала нужно дождаться возможности смены количества читателей (семафора readers_changing, я подробно описал зачем он нужне в первом пункте), далее законно увеличить счетчик активных читателей на 1. И если это оказался первый потенциально активный читатель (if (readers_count) == 1), то нужно дождаться доступа к базе данных и заблокировать его, чтоб все последующиие читатели могли им пользоваться, а ни один писатель не мог его получить. Если же это оказался не первый читатель, значит кто-то из читателей уже занял доступ к библиотеке и можно туда войти без sem_wait. Далее сразу же необходимо разблокировать семафор readers_changing, чтобы другие читатели могли тоже заходить, и только после этого считывать какие-то данные из библиотеки. По окончании работы с базой необходимо опять же заблокировать семафор readers_changing, безболезненно уменьшить на один количество активных читателей, и если текущий поток был последним таковым, то необходимо разблокировать доступ к базе данных sem_post(&database_access). В конце, как обычно, разблокировать семафор readers_changing.
4. Ввод данных из командной строки присутсвует (формально, можно ввести либо имя файла, либо заставить программу считывать с консоли, либо рандомно сгенерировать данные.

На 7 баллов:

1. Для ввода данных из файла необходимо передать име файла в параметрах командной строки при запуске программы. (> ./main <имя файла>). Вывод информации осуществляется в консоль и в файл output.txt
2. Ввод данных из командной строки расширен с учетом этих изменений.
3. Различные варианты выходных данных при входных 3 3 (сделал около 10 запусков > ./main input.txt) можно посмотреть в файле tests.txt. Несложно заметить, что к базе действительно одновременно могут подключаться либо несколько читателей, либо только один писатель, причем писатель не может подключиться, пока базой пользуется хотя бы один читатель, и наоборот, ни разу писатель не подключался, пока все подключенные читатели не завершили свое пользование. Таким образом, все необходимые требования выполнены.

На 8 баллов:

1. Генерация рандомных данных присутствует, если не передавать в программу никаких входных данных. При этом для удобства сгенерированные входные данные выводятся в консоль.
2. Ввод данных с командной строки расширен с учетом рандномной генерации данных.
3. Различные выходные значения проверялись в точно таком же критерии на 7 баллов, поэтому предполгаю, что в этом нужно доказать, что данные действительно генерируеются рандомно. Примеры 5 подряд запусков программы (> ./main) можно посмотреть в файле rand_tests.txt. Видно, что данные действительно генерируются рандомно в условленных диапазонах. Также получилось покрыть этими тестами угловые случаи, когда количество читателей или писателей равно нулю. Программа не падает и отрабатывает корректно.

На 9 баллов (вариант 1):
Все исходные коды и текстовые файлы для этого пункта лежат в папке 9
Рассмотрю 4 ситуации:
1. Вообще не использоваться семафоры, просто паралелльные потоки. Исходный код – в файле main2.cpp. В качестве входных данных возьмем 10 10, и запустим программу несколько раз. Также для наглядности проблем уберем все засыпания программ. Результаты нескольких прогонов можно посмотреть в файле tests2.txt. Видно, что одновременно могут получать доступ и писатели, и читатели, а также местами нарушается вывод, когда один поток перехватывает управление стандартным потоков вывода и получается полная бессмыслица (не получилось добиться прям полностью рандомной последовательности слов, но некоторые строчки начинаются с цифр, а некоторые не до конца допечатаны, думаю этого хватит для подтверждения проблем). В общем, ожидаемо много проблем.
2. Не использовать семафор information_output для вывода в консоль/файл. Исходный код – в файле main3.cpp. В качестве входных данных в этот раз возьмем 10 2, для наглядности возможных проблем снова уберем временные засыпания программы. Результаты неправильного выполнения программ можно посмотреть в файле tests3.txt. Опять же, не хватает нескольких слов в некоторых строках, уже неоднократно пояснялось из-за чего.
3. Не использовать семафор readers_changing для смены читателей. Исходный код – в файле main4.cpp. Временные засыпания снова были убраны, но даже с учетом этого оказалось очень непростой задачей добиться неправильного выполнения. Примерно за 50 попыток запуска на тесте 10 10 и еще столько же на тесте 10 2 мне удалось лишь один раз спровоцировать deadlock между потоками, когда они ждут друг друга и программа останавливается навсегда.
4. Не использовать семафор database_access для доступа к базе данных. Исходный код можно найти в файле main5.cpp. Программа была протестирована на тесте 3 3. Уже после двух запусков (результаты которых можно найти в файле tests5.txt) можно заметить ожидаемую ошибку, что и

читательские, и писательские потоки одновременно владеют доступом к базе данных.