Задача 1.
Последовательная консистентность.
1) Определение
При параллельном выполнении, все процессы должны «видеть» одну и ту же последовательность записей в память.
2) Алгоритм
Пробую нецентрализованный – используется надежный неделимый широковещательный алгоритм:

1) при записи – отправить всем информацию о записи, дождаться, пока все пришлют логическое время, выбрать максимальное время и отправить его всем; выполнить запись, только если запись стала первой в очереди модификаций
2) при чтении – проверить, что пришли подтверждения о всех предыдущих операциях записи данного процесса
3) если пришло от кого-то сообщение о записи – посмотреть логическое время и отправить логическое время процессу-отправителю; сообщение указать как «недоставленное» и установить в очередь; когда пришло сообщение о конечном логическом времени от процесса-отправителя, уточнить очередь сообщения; если первое сообщение в очереди стало «доступным» - выполнить запись
4) процесс может продолжать выполнение работы, при этом он блокируется при следующей операции чтения/записи общей переменной, пока не будет выполнена текущая операция записи

Централизованный алгоритм:

1) при записи – отправить сообщение о записи координатору, ждет реакцию координатора; когда пришло подтверждение – обновить переменную
2) при чтении – убедиться, что предыдущие записи прошли
3) если пришла информация о модификации – модифицировать переменную
4) блокироваться при ожидании сообщения от координатора (можно выполнять работу до следующего чтения/записи, видимо)

3) Для нецентрализованного алгоритм:
- для каждой модификации: 9 рассылок, 9 приемов и еще 9 рассылок – 27 сообщений, итого 10(27 Ts + 9 send Tb + 9 ans Tb + 9 resend Tb)
Для централизованного:
Для каждой модификации: 1 сообщение-запрос + 10 сообщений о модификации
Итого 10(11Ts + req Tb + 10 ans Tb)

Задача 2.
Причинная консистентность памяти
1) Последовательность операций записи, которые потенциально причинно зависимы, должна наблюдаться всеми процессами системы одинаково, параллельные операции записи могут наблюдаться разными узлами в разном порядке».

2) Алгоритм:
Каждая модификация нумеруется. В каждом процессе хранятся номеру всех известных модификаций других процессов

1) при записи – если есть информация о модификациях, которые еще не проведены, то блокируемся до прихода сообщений с модификацией; отсылаем модификацию и все известные ранее модификации всем процессам
2) при чтении – без блокировок
3) если пришла информация о записи, то модификация откладывается до того момента, как все модификации не осуществятся
4) блокировки – при модификациях, если какие-то другие модификации еще не выполнены

3) - Нулевой нумерует свое сообщение, отсылает его всем – 9(Ts + a Tb)

- второй нумерует свое сообщение, добавляет информацию о сообщении нулевого, отсылает всем – 9(Ts + a Tb)

…

Итого: 10*9(Ts + a Tb)

Задача 3.

Процессорная консистентность:

1) «Для каждой переменной x есть общее согласие относительно порядка, в котором процессоры модифицируют эту переменную, операции записи в разные переменные - параллельны»

2) Алгоритм:

Все очень тупо – у каждого процесса – своя переменная, он нумерует изменения, отсылает всем модификацию

1) при записи – отсылаем информацию о модификации переменной процессу-владельцу, ждем, когда он вернет сообщение о модификации переменной
2) при чтении – без блокировок
3) если процесс – владелец переменной, то при получении сообщении о модификации отсылает всем информацию о модификации; если пришло позднее сообщение о модификации (у него слишком большой номер и мы что-то пропустили), то процесс блокируется и ждет недостающих модификаций
4) блокировка – при ожидании ответа от процесса-владельца

3) Вариант, когда каждый из процессов модифицирует переменную, которой владеет; получается, что только отсылается сообщение о модификации остальным процессам, время: 9 (Ts + a Tb).

Задача 4.

PRAM консистентность.

1) «Операции записи, выполняемые одним процессором, видны всем остальным процессорам в том порядке, в каком они выполнялись, но операции записи, выполняемые разными процессорами, могут быть видны в произвольном порядке».

2) Алгоритм:

Выстраиваем операции модификации в конвеер, можем не ждать окончания операции, надо только быть уверенным, что модификацию увидят все в одном порядке

1) при записи – ставим сообщение о модификации в конвеер, продолжаем работу; сообщение посылается, если пришло подтверждение о предыдущей модификации (подтверждения гарантируются протоколом передачи)
2) при чтении – без блокировки
3) блокировок нет

3) 3 раза модифицируют переменную. 3 раза надо послать сообщение о модификации каждым процессом. Итого

3*10(Ts + a Tb) (a – длина сообщения)

Задача 5.

Слабая консистентность.

1) Основана на выделении среди переменных специальных синхронизационных переменных (доступ к которым производится специальной операцией синхронизации памяти) и описывается следующими правилами:

1. *Доступ к синхронизационным переменным определяется моделью последовательной консистентности;*
2. *Доступ к синхронизационным переменным запрещен (задерживается), пока не выполнены все предыдущие операции записи;*

3. *Доступ к данным (запись, чтение) запрещен, пока не выполнены все предыдущие обращения к синхронизационным переменным.*

2) Алгоритм:

Исходит из того, что доступ к синхронизационным переменным определяется моделью последовательной консистенции: все модификации делаются без отправки сообщений (запоминаются где-то), при вызове операции синхронизации все остальные процессы должны узнать о всех модификациях данного процесса, при этом синхронизации должны быть видны всеми процессами в одинаковом порядке. Для этого можно использовать централизованный алгоритм (неинтересно) или надежный и неделимый широковещательный алгоритм (ниже его использую):

1) при записи переменной – запомнить щапись в буфере
2) при чтении – без блокировки
3) при вызове синхронизации: отправить всем сообщение о всех модификациях переменных, дождаться, пока не вернутся логические времена приема сообщений, далее выбрать наибольшее время и отправить его всем – модификация считается отправленной; все модификации выстроить в очередь, когда первое сообщение станет отправленной, можно ее применить и продолжить работу;
4) если пришло сообщение о модификациях – отправить процессу-отправителю сообщение с логическим временем, когда он пришлет сообщение с максимальным логическим временем – подтвердить сообщение и выставить его в очередь; если оно первое – то можно его применить; блокировки при этом нету, только при выполнении операции синхронизации
5) блокировка – при синхронизации, пока она не выполнится

3) 1 процесс меняет 10 переменных (операций коммуникации нет), затем меняет 3 синхронизационные переменные. На каждую такую смену – шлем сообщение всем, принимаем сообщение от всех (с логическим временем), отсылаем сообщение всем (с максимальным логическим временем). Итого: 3*9(3Ts + send Tb + resv Tb + resend Tb)

Задача 6.
Консистентность по выходу.
1) Две ситуации: вход в критическую секцию и выход из нее.

(1) До выполнения обращения к общей переменной, должны быть полностью выполнены все предыдущие захваты синхронизационных переменных данным процессором.
(2) Перед освобождением синхронизационной переменной должны быть закончены все операции чтения/записи, выполнявшиеся процессором прежде.
(3) Реализация операций захвата и освобождения синхронизационной переменной должны удовлетворять требованиям процессорной консистентности (последовательная консистентность не требуется, захваты разных переменных осуществляются параллельно).

2) Алгоритм:
Фактически, используется маркер, в котором хранится информация о модификациях. Маркер присутствует у процесса, который выполнил последние преобразования с критической секцией (или выполняет). Все преобразования заносятся в маркер. Ленивая реализация: после преобразований процесс не отсылает результаты, а передает их по требованию. Процесс, которому нужен доступ в секцию, отсылает запрос о маркере.

1) при записи обычной (несинхранизационной) переменной – запись в локальную память
2) при чтении – из локальной памяти
3) выход из критической секции – запись всех модификаций в маркер
4) вход в критическую секцию – запрос маркера у всех процессов, блокировка до появления маркера
5) если пришел запрос о маркере – запоминается
6) блокировка – только при входе в критическую секцию

3) Каждый процесс 3 раза отсылает запрос на маркер всем (возможна реализация, когда каждый процесс 3 раза выполняет критическую секцию, после чего только выполняет передачу маркера). Каждый процесс 3 раза передает маркер. Итого:
3*9 * 10 (Ts + req Tb) + 3 * 10 (Ts + resp Tb)
(в реализации с 3мя последовательными заходами в Крит секцию – в 3 раза меньше)
З.Ы. в данном решении не учитываются аппаратные возможности широковещания, их можно легко учесть – поделить на 9 первое слагаемое, кажется)

Задача 7.
Консистенция по входу.
1) Связана с захватом критической секции.

(1) Процесс не может захватить синхронизационную переменную до того, пока не обновлены все переменные этого процесса, охраняемые захватываемой синхронизационной переменной;
(2) Процесс не может захватить синхронизационную переменную в монопольном режиме (для модификации охраняемых данных), пока другой процесс, владеющий этой переменной (даже в немонопольном режиме), не освободит ее;
(3) Если какой-то процесс захватил синхронизационную переменную в монопольном режиме, то ни один процесс не сможет ее захватить даже в немонопольном режиме до тех пор, пока первый процесс не освободит эту переменную, и будут обновлены текущие значения охраняемых переменных в процессе, запрашивающем синхронизационную переменную.

2) Тут опять маркер. Реализация почти как в предыдущем алгоритме, только с каждой синхронизационной переменной связан свой маркер и в нем хранятся модификации определенных переменных (т.к. с каждой синхронизационной переменной связаны свои переменные). Если процесс хочет захватить переменную в немонопольном режиме, то ждет сообщения от владельца переменной (хотя бы одного). Если в монопольном, то нужен ответ от всех процессов (либо ок, либо маркер).

1) запись – из локальной памяти
2) чтение – из локальной памяти
3) захват переменной в немонопольный режим – шлем всем сообщение «немонопольный захват», ждем маркер
4) захват в монопольном режиме – шлем всем сообщение «монопольный захват», ждем ответа от всех
5) если пришел «монопольный захват», то шлем «ок», если маркера нет и маркер нам не нужен (если нужен – ждем маркер и потом отправляем), либо шлем маркер по возможности
6) если пришло «немонопольный захват», то, если у нас есть маркер и мы немонопольно владеем им, то шлем копию маркера запросившему
7) при каждом запросе «монопольный захват» или «немонопольный захват» блокировка до получения необходимых ответов
8) выход из секции – модифицируем маркер

3) 3 раза крит. секция, каждый шлет всем сообщение о запросе крит. секции. Далее все отвечают (т.к. монопольная модификация). И одно сообщение на передачу маркера.
Итого:
3*(9)*10*(Ts + req * Tb) + 3*9*10*(Ts + resp*Tb) + 3*10(Ts + marker*Tb)
З.Ы. девятки убираются при широковещательной шине, тройки убираются, если последовательно входим в критические секции на каждом процессе.

Тема 7.
Задача 1.
Надежный и неделимый широковещательный алгоритм:
В каждом процессоре – очередь поступающих сообщений. Идентификатор – логическое время прибытия (всегда разное).
1) Отправляется сообщение группе процессов (в сообщении хранятся идентификаторы процессов!)

При приеме:

- помечается как «недоставленное», в качестве приоритета – временная метка
- информируется отправитель о присланном сообщении и о текущем логическом времени

2) Получает ответы от всех адресатов

- выбирает из присланных сообщений максимальный приоритет
- рассылает всем этот приоритет

При приеме:

- сообщение устанавливается как «доставленное»
- если оно становится первым в очереди, то выполняется

3) При обнаружении недоставленного сообщения (отправитель сломался) выполняется следующее:

- отправляем всем соседям «запрос» о данном сообщении
- все отвечают, у кого-то есть «доставленное» - приоритет посылается всем
- нету доставленного -> начинаем заново (шлем всем, ждем отметок о времени).

Решение задачи:
- 5 сообщений посылается, после чего падает отправитель
- 5 сообщений с временными метками возвращается
- таймаут
- 9 сообщений от какого-нибудь процесса о времени
- 9 ответов о том, что «недоставлено» или нету
- 9 сообщений-запросов
- 9 ответов с временем
- 9 сообщений с финальным временем

Итого:
55*Ts + 14*req*Tb + 9*ask*Tb + 9*ans*Tb + 9*time*Tb + 9*final*Tb
Req – сообщения запросы (например, 9 байт с идентификаторами, 1 полезный байт)
Ask – сообщения вопрос типа «есть ли у кого-нибудь доставленное сообщение?» (1 байт с номером сообщения, 1 байт с полезной инфой)
Ans – Ответы на вопрос (нету или недоставлено)
Time – сообщения с логическим временем
Final – финальное сообщение с итоговым логическим временем

Задача 3.
Консистентное и строго консистентное множества контрольных точек.

1. Множество контрольных точек называется **строго консистентным**, если во время его фиксации никаких обменов между процессами не было.
2. Множество контрольных точек называется **консистентным**, если для любой зафиксированной операции приема сообщения, соответствующая операция посылки также зафиксирована (нет сообщений-сирот).

Простой метод фиксации консистентного множества контрольных точек - фиксация локальной контрольной точки после каждой операции посылки сообщения.

Синхронная фиксация контрольных точек и восстановление.
Требование: надежная передача сообщений (сообщения всегда приходят, схема FIFO).
Два вида точек – постоянная (уже точно установлена) и пробная – становится постоянной после окончания работы алгоритма.
Сохранение:

1) процесс создает пробную точку, шлет всем
   все создают пробную точку, отвечают процессу-инициатору
   З.Ы. при этом нельзя посылать неслужебные сообщения
2) процесс шлет всем информацию о создании новой постоянной точки, пробная становится постоянной

Оптимизация: если не было сообщений после последней контрольной точки, то можно не создавать новую
Восстановление:

1) инициатор шлет всем сообщение о готовке к откату
   все отвечают, что готовы
   З.Ы. после этого до конца алгоритма не посылается неслужебных сообщений
2) инициатор шлет сообщение об откате, все откатываются

Оптимизация: если не было сообщений с момента фиксации, то можно не откатываться.

Асинхронный алгоритм.
Предположение: есть стабильная область памяти, в которой хранится информация о всех присланных и отосланных сообщениях.

Точки создаются в произвольные моменты. Откат до консистентного состояния (когда все принятые и отправленные сообщения совпадают) – множество контрольных точек неконсистенто, может возникнуть эффект домино, но меньше нагрузки на создание точек.

Получение строгой консистентности:
- делаем нестрогую консистентность (по синхронному алгоритму)
- при этом считаем количество отосланных сообщений каждому процессу
- проверяем, все ли сообщения дошли до адресатов, для этого шлем им счетчик отосланных сообщений, если совпал – шлем «ок» процессу-инициатору консистентности
!!!З.Ы. этот пункт можно так сделать: все шлют одному процессу сообщения со счетчиками (9 сообщений), он все суммирует и шлет каждому процессу сообщение с общим количеством сообщений, которое тот должен был принять (9 сообщений), все отвечают готовностью, если готовы (9 сообщений) Итого 27 дополнительных сообщений; координатора проверять не надо, так как он уже всем отправлял сообщения -> каналы пусты.

Итого: $T1 + 3n (Ts + a*Tb)$

Задача 2.
Протоколы голосования.

1) Существует кворум записи Nw > N/2, кворум чтения Nr: Nw + Nr > N (N – количество процессов). Смысл в том, что мы должны быть уверены, что финальная версия данных есть у Nw процессов, таким образом, если мы опросим Nr процессов, то из присланных ими данных обязательно будет один ответ с самой последней версией данных.
2) Алгоритм:
    a. Запись: шлем всем(!) процессам запрос на модификацию – шлем M байт данных и K байт доп. информации (куда пишем данные, версия данных) Должны получить Vw ответов об успешной записи.
    b. Чтение – шлем запрос всем (можно, наверное, не всем, а только Nr процессам), получаем необходимые данные (шлем K байт – идентификатор данных)

Решении Задачи:
N - число байт.
10 – число процессов, с которыми контактирует данный процесс = M
2 – число записей = w
10 – число чтений = r
Накладные расходы сообщения (пускай 5 байт), в которых хранятся версия переменной и указание области памяти = k
На запись: w[M(Ts + (N + k)*Tb) + Vw*(Ts + Tb)] – отсылка записи и ответы от заданного числа процессов
На чтение: r[Vr(Ts + k *Tb) + Vr(Ts + N*Tb)]– запрос на чтение у Vr процессов и получение ответа от Vr процессов.
Считаем Vr + Vw = 11, после чего минимизируем число сообщений для Vr и Vw (при T = 300)