МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Юго-Западный государственный университет» (ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«23» 03 2023 г.

АНАЛИЗ ОБРАЩЕНИЙ ПОТОКОВ К ОБЩЕМУ РЕСУРСУ

Методические указания по выполнению практических работ для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Курск 2023

УДК 681.3

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры «Информационная безопасность» А.Л. Марухленко

Анализ обращений потоков к общему ресурсу: методические указания по выполнению практической работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 10 с. Библиогр.: с. 10.

Содержат сведения об администрирование и управление программно-аппаратными средствами контроля и фильтрации сетевых пакетов способах, а так же защиты от несанкционированного доступа к ресурсам персонального компьютера. Указывается порядок выполнения лабораторной работы, правила оформления и содержание отчета.

Методические указания по выполнению практических работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 152. Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ.

Изучить и научиться применять на практике организации вычислений с помощью нитей (потоков) и овладеть приѐмами работы с участками исполняемого кода, в которых происходит обращение к разделяемым ресурсам. Овладеть приѐмами получения численных характеристик многонитевых программ.

2. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

2.1. Задание на лабораторную работу

2.1.1. Написание многонитевой программы

Студент должен написать программу, в которой несколько потоков осуществляют обращения к нескольким общим ресурсам (например, к области активного окна приложения, закрашивая его каждый своим цветом или выводя в нѐм сообщение, что конкретный ресурс занят конкретным потоком). При этом недопустимо, чтобы поток занимал более чем один ресурс (для отсутствия возможности попадания в тупиковую ситуацию).

Поток занимает ресурс в течение определѐнного количества относительных единиц времени. В качестве такой единицы может быть определѐнное студентом количество пустых циклов. При этом для демонстрации работающей программы преподавателю число пустых циклов должно быть несколько десятков миллионов – чтобы занятие/освобождение ресурса можно было наблюдать в процессе работы программы. При моделировании и снятии численных характеристик данное значение можно уменьшить.

Процедура потока должна работать определѐнное (определяется студентом) количество циклов. В каждом цикле она случайным образом обращается к ресурсам (см. пример):

```
Const
  Porog1=0.09,
  Porog2=0.04;
…
Procedure Thread1.Execute
…
for j:= 1 to 1000 do
begin
```

```
    rand_num:=random;
    if rand_num> (1-Porog1) then
     begin
      {вход в критическую секцию, связанную
с ресурсом 1}
    {выход из критической секции занятие
ресурса,}
     end;
                          else
    begin
    rand_num:=random;
    if rand_num> (1-Porog2)  then
     begin
      {вход в критическую секцию, связанную
с ресурсом 2}
    {выход из критической секции занятие
ресурса,}
     end;
    …
    end;
    …
    {цикл, формирующий задержку в одну
относительную}
    {единицу времени}
    end;
    …
```

При этом числа Porog1 и Porog2 есть вероятности занятия потоком ресурса 1 и 2 соответственно.

Участок программы, соответствующий занятию ресурса (пребыванию в критической секции), может быть описан так:

```
    …
    CS1.enter {вход в критическую секцию
CS1}
    …
    {действия, визуализирующие занятие
потоком ресурса}
    …
```

```
    for i:=1 to 10000 do
     begin
     r_n:=random;
     if r_n> (1-Porog3) then break;
    {цикл, формирующий задержку в одну
относительную}
    {единицу времени}
     end;
    …
    {действия, визуализирующие освобождение
потоком ресурса}
    …
    CS1.leave {выход их критической секции
CS1}
    …
```

При этом число Porog3 есть вероятность освобождения потоком ресурса.

2.1.2. Исследование работы многонитевой программы

После написания и отладки программы в соответствии с индивидуальным вариантом студенту необходимо исследовать еѐ работу и получить ряд численных характеристик. Параметрами, которые задаются до запуска программы является множество значений вероятности занятия $p^з_{i,j}$ и освобождения $p^o_{i,j}$ i–го ресурса j–м потоком.

После одного полного цикла работы программы можно получить следующие числа:

−время работы программы[1] $T_{общ}$;

−время занятия i–го ресурса j–м потоком $T_{i,j}$;

−время работы j–го потока T^n_j;

В результате 10 – 15 запусков многонитевой программы с фиксированными вероятностями занятия освобождения ресурсов можно получит средние значения вышеприведенных характеристик.

[1] Всѐ время определяется в относительных единицах

На основе средних полученных значений можно рассчитывать следующие характеристики, считая, что *M* – число ресурсов *N* – число потоков:

–долю времени работы *j*–го потока $T^n{}_j \big/ T_{общ}$;

–долю времени занятия *i*–го ресурса *j*–м потоком $T_{i,j} \big/ T^n{}_j$;

–долю времени занятия всех ресурсов *j*–м потоком от времени его работы $\sum_{i=1}^{M} T_{i,j} \Big/ T^n{}_j$;

–долю времени занятия *i*–го ресурса $\sum_{j=1}^{N} T_{i,j} \Big/ T_{общ}$;

–долю времени занятия всех ресурсов всеми потоками $\sum_{i=1}^{M}\sum_{j=1}^{N} T_{i,j} \Big/ T_{общ}$;

–долю времени занятия *i*–го ресурса от времени занятия всех ресурсов всеми потоками $\sum_{j=1}^{N} T_{i,j} \Big/ \sum_{i=1}^{M}\sum_{j=1}^{N} T_{i,j}$

В соответствии с индивидуальным заданием студент должен нарисовать графики зависимости исследуемого параметра от переменного параметра моделирования при постоянных остальных параметрах

2.2. Индивидуальные варианты заданий.

Таблица 1. – Варианты заданий.

Номер варианта	Количество потоков	Количество ресурсов	Исследуемый параметр	Переменный параметр
	3	4	$p^з{}_{i,j}$	$T^n{}_j \big/ T_{общ}$
	4	3	$p^o{}_{i,j}$	$T_{i,j} \big/ T^n{}_j$
	5	3	$p^з{}_{i,j}$	$\sum_{i=1}^{M} T_{i,j} \Big/ T^n{}_j$
	3	5	$p^o{}_{i,j}$	$\sum_{j=1}^{N} T_{i,j} \Big/ T_{общ}$

Номер варианта	Количество потоков	Количество ресурсов	Исследуемый параметр	Переменный параметр
	3	4	$p^{o}_{i,j}$	$\sum_{i=1}^{M}\sum_{j=1}^{N}T_{i,j}\Big/T_{об}$
	4	3	$p^{з}_{i,j}$	$\sum_{j=1}^{N}T_{i,j}\Big/\sum_{i=1}^{M}\sum_{j=}^{N}$
	5	3	$p^{з}_{i,j}$	$T^{n}{}_{j}\Big/T_{общ}$
	3	5	$p^{o}_{i,j}$	$T_{i,j}\Big/T^{n}{}_{j}$
	3	4	$p^{з}_{i,j}$	$\sum_{i=1}^{M}T_{i,j}\Big/T^{n}{}_{j}$
	4	3	$p^{o}_{i,j}$	$\sum_{j=1}^{N}T_{i,j}\Big/T_{общ}$
	5	3	$p^{o}_{i,j}$	$\sum_{i=1}^{M}\sum_{j=1}^{N}T_{i,j}\Big/T_{об}$
	3	5	$p^{з}_{i,j}$	$\sum_{j=1}^{N}T_{i,j}\Big/\sum_{i=1}^{M}\sum_{j=}^{N}$

Студенты формируют бригады по два человека в каждой. Бригада выполняет вариант, указанный преподавателем. Исследуемые характеристики получать для первого потока (или ресурса)

2.3. Содержание отчѐта

1) Вариант задания.
2) Листинг программ.
3) Вид главного окна программ.
4) Численные характеристики одного запуска программы.
5) График исследуемой зависимости

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое нити (потоки)?
2. Принцип работы многонитевой программы?
3. Что такое пустые циклы?

4. БИБЛИОГРАФИЧЕСКИЙ СПИСОК.

1) Таненбаум Э., Вудхал А. Операционные системы: разработка и реализация. – СПб.: Издательский дом «Питер», 2006.

2) Олифер В.Г., Олифер Н.А.. Сетевые операционные системы. - СПб.: Издательский дом «Питер», 2003.

3) Ахо В., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. - М.: Издательский дом «Вильямс», 2001.

4) Вильямс А. Системное программирование в Windows 2000 для профессионалов. - СПб.: Издательский дом «Питер», 2001.

5) Гордеев А.В., Молчанов А.Ю.. Системное программное обеспечение – Спб.: Питер, 2001. – 736с. илл.

6) Гордеев, А.В., Кучин, Н.В. Проектирование взаимодействующих процессов в операционных системах: Учебное пособие. – Л.: ЛИАП, 1991. -72 с.

7) Кэнту, М. Delphi 5 для профессионалов. – СПб.: Издательский дом «Питер», 2001.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Юго-Западный государственный университет» (ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«_23_» _03_____ 2023 г.

ИССЛЕДОВАНИЕ ТУПИКОВЫХ СИТУАЦИЙ

Методические указания по выполнению практических работ для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Курск 2023

УДК 681.3

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры «Информационная безопасность» А.Л. Марухленко

Исследование тупиковых ситуаций: методические указания по выполнению практической работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 18 с. Библиогр.: с. 18.

Содержат сведения об администрирование и управление программно-аппаратными средствами контроля и фильтрации сетевых пакетов способах, а так же защиты от несанкционированного доступа к ресурсам персонального компьютера. Указывается порядок выполнения лабораторной работы, правила оформления и содержание отчета.

Методические указания по выполнению практических работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 151. Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ.

Исследование на марковской модели возможности возникновения ситуаций взаимоблокировки при обращении процессов к разделяемому счѐтному ресурсу.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.

2.1. Понятие тупика. Условия возникновения тупиков.

Взаимная блокировка, называемая также *дедлоком (deadlocks), клинчем (clinch)* или *тупиком* есть ситуация при которой каждый процесс из группы двух или более процессов удерживает ресурс, необходимый для продолжения работы другого процесса группы. Тупиковые ситуации надо отличать от простых очередей, хотя и те и другие возникают при совместном использовании ресурсов и внешне выглядят похоже: процесс приостанавливается и ждет освобождения ресурса. Однако очередь возникает тогда, когда ресурс недоступен в данный момент, но через некоторое время он освобождается, и процесс продолжает свое выполнение. Тупик же является неразрешимой ситуацией.

Необходимые условия возникновения тупиковых ситуаций:

1) Условие взаимного исключения. Каждый ресурс в данный момент или отдан ровно одному процессу, или доступен.

2) Условие удержания и ожидания. Процессы, в данный момент удерживающие полученные ранее ресурсы, вправе запрашивать новые ресурсы.

3) Условие отсутствия принудительной выгрузки ресурса. У процесса нельзя принудительным образом забрать ранее полученные ресурсы. Процесс, владеющий ими, должен сам освободить ресурсы.

4) Условие циклического ожидания. Должна существовать круговая последовательность из двух и более процессов, каждый из которых ждет доступа к ресурсу, удерживаемому следующим членом последовательности.

Для того чтобы произошла взаимоблокировка, должны выполниться все эти четыре условия. Если хоть одно из них отсутствует, тупиковая ситуация невозможна.

2.2. Меры борьбы с тупиками

При столкновении с взаимоблокировками практикуются четыре стратегии.

1) Пренебрежение проблемой в целом.

2) Обнаружение и восстановление. Позволить взаимоблокировке произойти, обнаружить ее и предпринять какие-либо действия.

3) Динамическое избежание тупиковых ситуаций с помощью аккуратного распределения ресурсов.

4) Предотвращение с помощью структурного опровержения одного из четырех условий, необходимых для взаимоблокировки.

Большая часть операционных систем, включая UNIX и Windows, игнорируют проблему тупиков. Они исходят из предположения, что большинство пользователей скорее предпочтут иметь дело со случающимися время от времени взаимоблокировками, чем с правилом, по которому всем пользователям разрешается только один процесс, один открытый файл и т. д. Если бы можно было легко устранить взаимоблокировки, не возникло бы столько разговоров на эту тему. Сложность заключается в том, что цена достаточно высока, и исчисляется она в наложении неудобных ограничений на процессы.

2.2.1. Обнаружение и устранение взаимоблокировок.

Система не пытается предотвратить попадание в тупиковые ситуации. Проверяется наличие процессов, которые были заблокированы долго, скажем, в течение одного часа. Если такие процессы обнаруживаются, они завершаются.

Стратегия обнаружения и восстановления применяется в больших компьютерных системах, особенно в системах пакетной обработки, где принудительное завершение и повторный запуск обычно приемлемы. Однако необходимо с осторожностью производить восстановление любых модифицированных файлов и устранение любых побочных эффектов, которые могли произойти.

2.2.2. Предотвращение взаимоблокировок

Предотвращение тупика основывается на предположении о чрезвычайно высокой его стоимости, поэтому лучше потратить

дополнительные ресурсы системы, чтобы исключить вероятность возникновения тупика при любых обстоятельствах. Этот подход используется в наиболее ответственных системах, часто это системы реального времени.

Предотвращение можно рассматривать как запрет существования опасных состояний. Поэтому дисциплина, предотвращающая тупик, должна гарантировать, что какое-либо из четырех условий, необходимых для его наступления, не может возникнуть.

Условие взаимного исключения можно подавить путем разрешения неограниченного разделения ресурсов. Это удобно для повторно вводимых программ и ряда драйверов, но совершенно неприемлемо к совместно используемым переменным в критических интервалах.

Условие ожидания можно подавить, предварительно выделяя ресурсы. При этом процесс может начать исполнение, только получив все необходимые ресурсы заранее. Следовательно, общее число затребованных параллельными процессами ресурсов должно быть не больше возможностей системы. Поэтому предварительное выделение может привести к снижению эффективности работы вычислительной системы в целом. Необходимо также отметить, что предварительное выделение зачастую невозможно, так как необходимые ресурсы становятся известны процессу только после начала исполнения.

Условие отсутствия перераспределения можно исключить, позволяя операционной системе отнимать у процесса ресурсы. Для этого в операционной системе должен быть предусмотрен механизм запоминания состояния процесса с целью последующего восстановления. Перераспределение процессора реализуется достаточно легко, в то время как перераспределение устройств ввода/вывода крайне нежелательно.

Условие кругового ожидания можно исключить, предотвращая образование цепи запросов. Это можно обеспечить с помощью принципа ***иерархического выделения ресурсов***. Все ресурсы образуют некоторую иерархию. Процесс, затребовавший ресурс на одном уровне, может затем потребовать ресурсы только на более высоком уровне. Он может освободить ресурсы на данном уровне только после освобождения всех ресурсов на всех более высоких уровнях. После того как процесс

получил, а потом освободил ресурсы данного уровня, он может запросить ресурсы на том же самом уровне. Пусть имеются процессы ПР1 и ПР2, которые могут иметь доступ к ресурсам R1 и R2, причем R2 находится на более высоком уровне иерархии. Если ПР1 захватил R1, то ПР2 не может захватить R2, так как доступ к нему проходит через доступ к R1, который уже захвачен ПР1. Таким образом, создание замкнутой цепи исключается. Иерархическое выделение ресурсов часто не дает никакого выигрыша, если порядок использования ресурсов, определенный в описании процессов, отличается от порядка уровней иерархии. В этом случае ресурсы будут использоваться крайне неэффективно.

2.2.3. Обход тупиков

Обход тупика можно интерпретировать как запрет входа в опасное состояние. Если ни одно из упомянутых четырех условий не исключено, то вход в опасное состояние можно предотвратить при наличии у системы информации о последовательности запросов, связанных с каждым параллельным процессом. Доказано, что если вычисления находятся в любом неопасном состоянии, то существует по крайней мере одна последовательность состояний, которая обходит опасное. Следовательно, достаточно проверить, не приведет ли выделение затребованного ресурса сразу же к опасному состоянию. Если да, то запрос отклоняется. Если нет, его можно выполнить. Определение того, является ли состояние опасным или нет, требует анализа последующих запросов процессов.

Часто бывает так, что последовательность запросов, связанных с каждым процессом, неизвестна заранее. Но если заранее известен общий запрос на ресурсы каждого типа, то выделение ресурсов можно контролировать. В этом случае необходимо для каждого требования, предполагая, что оно удовлетворено, определить, существует ли среди общих запросов от всех процессов некоторая последовательность требований, которая может привести к опасному состоянию. Данный подход является примером контролируемого выделения ресурса.

Классическое решение этой задачи известно как *алгоритм банкира* Дейкстры. Алгоритм банкира напоминает процедуру принятия решения, может ли банк безопасно для себя дать взаймы денег. Принятие решения основывается на информации о

потребностях клиента (текущих и максимально возможных) и учете текущего баланса банка.

Алгоритм банкира позволяет продолжать выполнение таких процессов, которым в случае системы с предотвращением тупиков пришлось бы ждать. Основным накладным расходом стратегии обхода тупика с помощью контролируемого выделения ресурса является время выполнения алгоритма, так как он выполняется при каждом запросе. Причем алгоритм работает медленнее всего, когда система близка к тупику. Необходимо отметить, что обход тупика неприменим при отсутствии информации о требованиях процессов на ресурсы.

3. КРАТКОЕ ОПИСАНИЕ МАРКОВСКИХ ПРОЦЕССОВ.

Марковские модели. Основополагающими в теории вычислительных систем являются модели и аппарат теории марковских процессов. *Марковским* называется случайный процесс, состояние которого в очередной момент времени $t+\delta$ зависит только от текущего состояния в момент времени t. Это означает, что поведение марковского процесса в будущем определяется текущим состоянием процесса и не зависит от предыстории процесса – состояний, в которых пребывал процесс до момента t.

Марковские цепи классифицируются в зависимости от возможности перехода из одних состояний в другие. Основными являются два класса: поглощающие и эргодические цепи.

Поглощающая марковская цепь содержит поглощающее состояние, достигнув которого, процесс уже никогда его не покидает, т. е. по сути прекращается. Из какого бы состояния ни начался процесс, при $t\to\infty$ с вероятностью 1 он окажется в поглощающем состоянии. Основная характеристика случайного процесса, порождаемого поглощающей марковской цепью, – вероятность пребывания процесса в состояниях $S = \{s_1, \ldots, s_K\}$ к моменту времени τ. Поглощающие марковские цепи широко используются в качестве временных моделей программ и вычислительных процессов.

Эргодическая марковская цепь представляет собой множество состояний, связанных матрицей переходных вероятностей таким образом, что из какого бы состояния процесс ни исходил, после

некоторого числа шагов он может оказаться в любом состоянии. Это означает, что в любое состояние эргодической цепи можно перейти из любого другого состояния за сколько–то шагов. По этой причине состояния эргодической цепи называются эргодическими (возвратными). Процесс, порождаемый эргодической цепью, начавшись в некотором состоянии, никогда не завершается, а последовательно переходит из одного состояния в другое, попадая в различные состояния с разной частотой, зависящей от переходных вероятностей. Поэтому основная характеристика эргодической цепи – вероятности пребывания процесса в состояниях $S = \{s_1, \ldots, s_K\}$, – доля времени, которую процесс проводит в каждом из состояний.

Эргодические цепи широко используются в качестве моделей надежности систем. При этом состояния системы, различающиеся составом исправного и отказавшего оборудования, трактуются как состояния эргодической цепи, переходы между которыми связаны с отказами и восстановлением устройств и реконфигурацией связей между ними, проводимой для сохранения работоспособности системы. Оценки характеристик эргодической цепи дают представление о надежности поведения системы в целом. Кроме того, эргодические цепи широко, используются в качестве базовых моделей взаимодействия устройств с задачами, поступающими на обработку.

В классе марковских процессов выделяют процессы с дискретными состояниями, называемые *марковскими цепями.* Когда множество состояний процесса $S = \{s_1, \ldots, s_K\}$ конечно марковскую цепь называют *конечной.* Конечная марковская цепь может быть определена в непрерывном или дискретном времени. В первом случае переходы процесса из одного состояния в другое связываются с произвольными моментами времени $t_1, t_2, t_3, \ldots$ цепь называют *непрерывной;* во втором – только в фиксированные моменты времени, обозначаемые порядковыми номерами $t = 0, 1, 2,\ldots$ и цепь называется *дискретной.*

В ходе данной лабораторной работы обращение процессов к ресурсам моделируется непрерывной марковской цепью

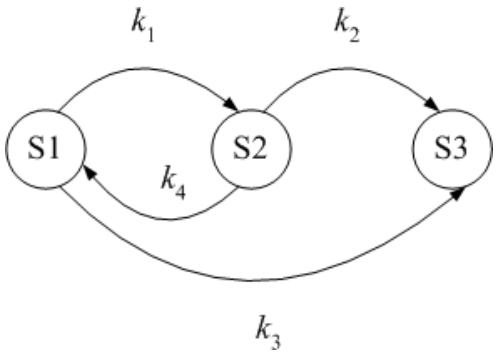

Рисунок 1. Граф непрерывной марковской цепи с поглощающим состоянием.

Марковский процесс с дискретными состояниями $S = \{s_1, \ldots, s_K\}$, переходы между которыми разрешаются в любой момент времени, называется *непрерывной марковской цепью.* Однородная непрерывная марковская цепь, поведение которой в любой момент времени подчиняется одному и тому же закону, задается матрицей интенсивностей переходов $Q = [q_{ij}]$ $i, j = 1, \ldots, K$. Интенсивность переходов определяется следующим образом:

$$q_{ij} = \lim_{\Delta t \to 0} \frac{p_{ij}(\Delta t)}{\Delta t};$$

где $p_{ij}(\Delta t)$ – вероятность перехода процесса из состояния S_i в состояние S_j, за время Δt. Это означает, вероятность нахождения в состоянии S_i в течение промежутка времени Δt в уменьшится на величину $q_{ij} \cdot \Delta t$ за счѐт увеличения вероятности нахождения в состоянии S_j, на ту же величину. Интенсивность переходов должна удовлетворять условию

$$\sum_{i=1}^{K} q_{ij} = 0, \qquad j = 1, \ldots, K$$

Выше представлен граф непрерывной марковской цепи с тремя состояниями S1, S2, S3. Дуги графа нагружены интенсивностями переходов. Графу соответствует следующая матрица интенсивностей переходов:

$$Q = [q_{ij}] =$$

	s_1	s_2	s_3
s_1	$-(k_1 + k_3)$	k_4	0
s_2	k_1	$-(k_2 + k_4)$	0
s_3	k_3	k_2	0

Элементы каждой строки определяют интенсивности входящего (если элемент со знаком плюс) или исходящего (если элемент со знаком минус) потоков в/из соответствующего состояния системы.

Основная характеристика непрерывной марковской цепи – распределение вероятностей состояний $P = \{p_1, \ldots, p_K\}$, где $\{\alpha_1, \ldots, \alpha_K\}$ – вероятности пребывания процесса в состояниях $\{s_1, \ldots, s_K\}$ соответственно. Распределение задается вероятностным решением системы линейных дифференциальных уравнений

$$\frac{dP}{dt} = Q \times P,$$

с заданными начальными условиями $P(0) = \pi$, решаемую либо численными методами, либо аналитическими.

В соответствии с марковским свойством вся предыстория процесса сказывается на его поведении в будущем только через текущее состояние, которое и определяет дальнейший ход процесса. Таким образом, нет необходимости знать, как долго процесс находится в текущем состоянии. Отсюда следует, что распределение остающегося времени пребывания процесса в состоянии s_j должно зависеть только от самого состояния, а не от времени пребывания в нем. Этим свойством обладает только одно распределение – экспоненциальное, функция плотности вероятности которого имеет следующий вид: $p(t) = (1/\tau) \cdot \exp(-t/\tau)$, где τ – параметр распределения, определяющий математическое ожидание случайной величины t. Таким образом, непременное свойство непрерывного марковского процесса – экспоненциальность распределения времени пребывания процесса в каждом из состояний.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ.

4.1. Общие сведения.

В ходе данной работы моделируется обращение двух процессов к разделяемому счѐтному ресурсу. Делается допущение, что время удержания единицы ресурса распределено по экспоненциальному закону. Процессы запрашивают и освобождают по одной единице ресурса. Запрос и освобождение ресурса есть случайное событие. Таким образом, обращения процессов к

ресурсам может быть промоделированы марковской цепью. В зависимости от варианта задания рассматриваются системы: не предусматривающие возникновение тупиковых ситуаций, предусматривающие обнаружение и устранение взаимоблокировок, предусматривающие обход тупиковых ситуаций за счѐт контроля выделения ресурса.

4.2. Варианты заданий.

Варианты моделируемых систем приведены в таблице 1. Номер варианта определяется порядковым номером студента по списку подгруппы.

Таблица 1.

Номер варианта	Максимальное количество ресурса	Максимальная потребность в ресурсе 1–го процесса	Максимальная потребность в ресурсе 2–го процесса	Система контроля тупиковых ситуаций

	4	3	3	нет
	5	4	2	устранение
	4	4	3	обход
	3	2	3	нет
	4	3	3	устранение
	5	4	2	обход
	4	4	3	нет
	3	2	3	устранение
	4	3	3	обход
	5	4	2	нет
	4	4	3	устранение
	3	2	3	обход

Интенсивности обращения к ресурсам и восстановления после тупиков в относительную единицу времени приведены в таблице 2. Вариант определяется суммой номера группы студента и номера подгруппы.

Таблица 2.

Номер варианта	Процесс 1		Процесс 2		Интенсивность восстановления после тупика
	Интенсивность запросов единицы ресурса	Интенсивность освобождения единицы	Интенсивность запросов единицы ресурса	Интенсивность освобождения единицы	

	0,2	0,15	0,25	0,25	0,02
	0,15	0,15	0,3	0,25	0,01
	0,2	0,25	0,3	0,3	0,015
	0,2	0,2	0,2	0,25	0,005

4.3. Ход работы.

В ходе выполнения работы студенту необходимо выполнить следующее:

–в соответствии со своим вариантом составить марковскую цепь, имитирующую работу системы;

–по составленной цепи составить систему дифференциальных уравнений для расчѐта вероятности пребывания системы в состояниях;

–выбрать начальные условия (вероятность пребывания в начальный момент времени в начальном состоянии равна единице, в остальных – нулю) и решить систему дифференциальных уравнений на интервале $[0; T_{max}]$[1];

–привести результат моделирования[2]

4.4. Содержание отчѐта

- Индивидуальный вариант.
- Граф марковской цепи.
- Графики зависимости вероятности пребывания в требуемых состояниях системы (или их суммы) от времени.
- Результаты.

[1] T_{max} выбирается исходя из варианта задания: если нет системы обнаружения или обхода тупиков, то оно равно времени, через которое система окажется в тупиковой ситуации с вероятностью 0.1; если «устранение» или «обход» (при этом цепи будут эргодическими), то время, через которое вероятности пребывания системы в состояниях перестанут изменяться (примут установившиеся значения)

[2] Результатом, исходя из варианта задания, является следующее: если нет системы обнаружения или обхода тупиков, то временя, через которое система окажется в тупиковой ситуации с вероятностью 0.1, если «устранение» – доля времени (вероятность) пребывания системы в состоянии тупика, если «обход» – доля времени (вероятность) пребывания системы в состояниях, когда какой – либо из процессов заблокирован.

5. ПРИМЕР СОСТАВЛЕНИЯ МАРКОВСКОЙ ЦЕПИ.

Пусть дана система, в которой существует 4 единицы ресурса. Работают 2 процесса, максимальная потребность каждого из них в ресурсе равна трѐм. Интенсивность запросов единицы ресурса со стороны первого процесса – k_1^i, второго – k_1^i (данные переходы на рисунке обозначены соответственно пунктиром и точками). Интенсивность освобождения единицы ресурса со стороны первого процесса – k_1^o, второго – k_1^o. Рассмотрим ситуацию, когда система не имеет механизмов обхода или устранения тупиковых ситуаций. Тогда граф цепи Маркова, имитирующей работу такой системы, будет выглядеть, как показано ниже (Рисунок 2). Состояние [*i*,*j*] данной цепи соответствует использованию процессом 1 *i* единиц ресурса, а процессом 2 – *j* единиц. Переходы между состояниями соответствуют занятию или освобождению единицы ресурса. Состояние [(*i*),*j*] соответствует использованию процессом 2 *j* единиц ресурса в то время, когда процесс 1 блокирован, так как система не может предоставить ему *i* – ю единицу ресурса. Состояние [*,*] – тупиковая ситуация, в которую попала система, после того как каждый из процессов, заняв по 2 единицы ресурса, запросил ещѐ по одной. Состояние [2,2] можно охарактеризовать как ненадѐжное, попадание в тупиковую ситуацию возможно только из него. Поэтому, если в системе существует механизм обхода тупиковых ситуаций, попадание в данное невозможно.

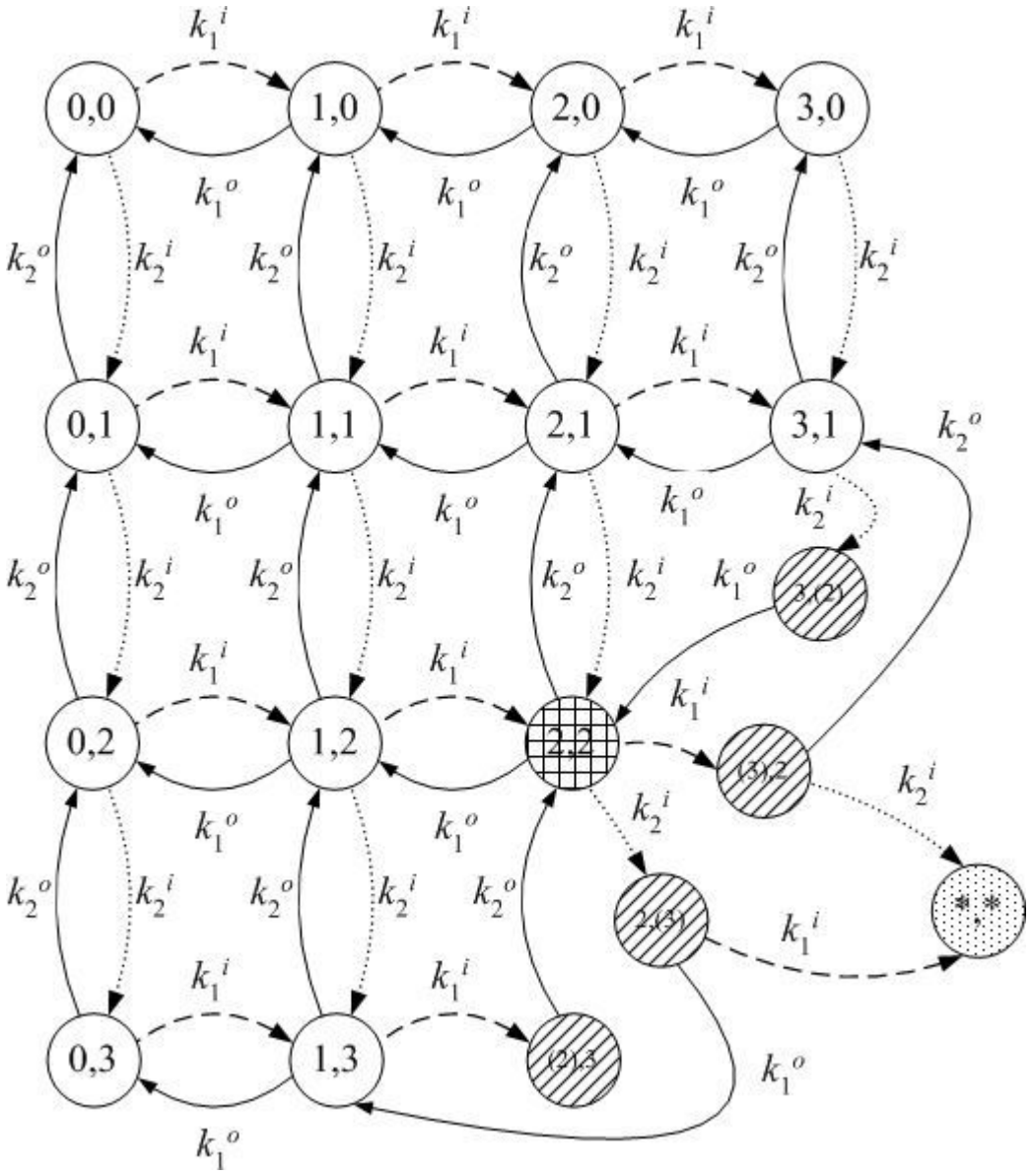

Рисунок 2. – Пример марковской цепи.

Составив по данной цепи систему дифференциальных уравнений и решив еѐ при следующих начальных условиях: начальная вероятность нахождения системы в состоянии [0,0] равна единице, во всех остальных – нулю, можно получить значение вероятности попадания в тупиковую ситуацию в любой момент времени.

Наличие связи между состоянием [*,*] и [0,0] означает наличие в системе механизма обнаружения и устранения взаимоблокировок. Интенсивность данного перехода будет моделируемой интенсивностью восстановления системы после

тупика. При этом данная цепь превратится из цепи с поглощающим состоянием в эргодическую.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ.

1) Что называют тупиковой ситуацией?
2) Приведите примеры возникновения тупиков при обращении процессов к счѐтным и несчѐтным ресурсам.
3) Какие существуют условия возникновения тупиковых ситуаций?
4) Назовите методы борьбы с тупиковыми ситуациями и кратко охарактеризуйте их.
5) Что такое марковский процесс? Охарактеризуйте основные типы марковских процессов.

7. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1) Олифер В.Г., Олифер Н.А.. Сетевые операционные системы. - СПб.: Издательский дом «Питер», 2003.

2) Ахо В., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. - М.: Издательский дом «Вильямс», 2001.

3) Гордеев, А.В., Молчанов, А.Ю. Системное программное обеспечение – Спб.: Питер, 2001. – 736с. ил.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«_23_» _03_ 2023 г.

МОДЕЛИРОВАНИЕ ДОСТУПА К РАЗДЕЛЯЕМОМУ РЕСУРСУ

Методические указания по выполнению лабораторных и практических работ для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Курск 2023

УДК 681.3

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры «Информационная безопасность» А.Л. Марухленко

Моделирование доступа к разделяемому ресурсу: методические указания по выполнению лабораторных и практических работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 15 с. Библиогр.: с. 15.

Излагаются методические указания по выполнению работы на персональной ЭВМ. Изучаются методы моделирования доступа к разделяемым ресурсам с помощью программных сред.

Методические указания по выполнению лабораторных и практических работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 146. Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ.

Изучить и научиться применять на практике механизмы работы с критическими секциями в операционной системе приложениями, а также механизмы предотвращения гонок при обращении к разделяемым ресурсам.

2. КРАТКАЯ ТЕОРИЯ.

2.1. Проблема межпроцессного взаимодействия.

Процессам часто нужно взаимодействовать друг с другом, например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла. Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и активизацией процессов, организацией очередей, блокированием и освобождением ресурсов.

Проблема разбивается на три компонента: передача информации от одного процесса к другому; контроль над деятельностью процессов (гарантии «непересечения» процессов в критических ситуациях); согласование действий процессов – если процесс А отвечает за поставку данных, а процесс В за их вывод на печать, то процесс В должен подождать, пока не поступят данные от процесса А.

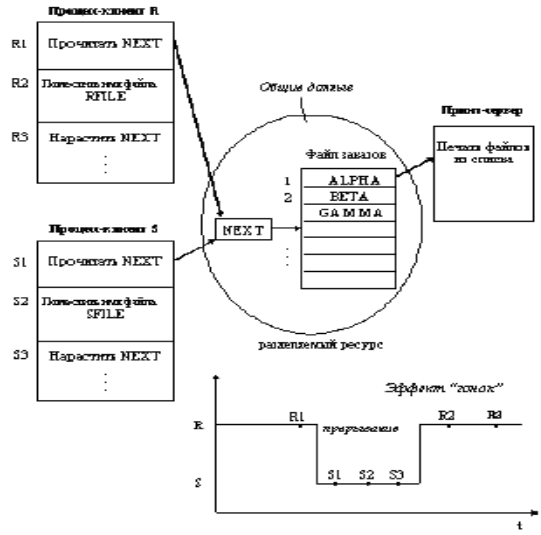

Рисунок 2.1. Пример необходимости синхронизации

Рассмотрим, например (Рисунок 2.1), программу печати файлов (принт-сервер). Эта программа печатает по очереди все файлы, имена которых последовательно в порядке поступления записывают в специальный общедоступный файл "заказов" другие программы. Особая переменная NEXT, также доступная всем процессам-клиентам, содержит номер первой свободной для записи имени файла позиции файла "заказов". Процессы - клиенты читают эту переменную, записывают в соответствующую позицию файла "заказов" имя своего файла и наращивают значение NEXT на единицу. Предположим, что в некоторый момент процесс R решил распечатать свой файл, для этого он прочитал значение переменной NEXT, значение которой для определенности предположим равным 4. Процесс запомнил это значение, но поместить имя файла не успел, так как его выполнение было прервано (например, в следствие исчерпания кванта). Очередной процесс S, желающий распечатать файл, прочитал то же самое значение переменной NEXT, поместил в четвертую позицию имя своего файла и нарастил значение переменной на единицу. Когда в очередной раз управление будет передано процессу R, то он, продолжая свое выполнение, в полном соответствии со значением текущей свободной позиции, полученным во время предыдущей итерации, запишет имя файла также в позицию 4, поверх имени файла процесса S. Таким образом, процесс S никогда не увидит свой файл распечатанным. Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций: в предыдущем примере можно представить и другое развитие событий: были потеряны файлы нескольких процессов или, напротив, не был потерян ни один файл. В данном случае все определяется взаимными скоростями процессов и моментами их прерывания. Поэтому отладка взаимодействующих процессов является сложной задачей.

Ситуации подобные той, когда два или более процессов обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей процессов, называются *гонками*. Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме

мультипрограммирования, может привести к их неправильной работе или даже к краху системы.

2.2. Критическая секция

Важным понятием синхронизации процессов является понятие "критическая секция" программы (CS). *Критическая секция* – это часть программы, в которой осуществляется доступ к разделяемым данным. Для правильной совместной работы параллельных процессов и эффективного использования общих данных необходимо выполнение четырех условий:

- Два процесса не должны одновременно находиться в критических областях.
- В программе не должно быть предположений о скорости или количестве процессоров.
- Процесс, в состоянии вне критической области, не может блокировать другие процессы.
- Недопустима ситуация, в которой процесс вечно ждет попадания в критическую секцию.

Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс. Этот прием называют взаимным исключением.

2.3. Методы предотвращения гонок.

2.3.1. Запрет на прерывания.

Простейший способ обеспечить взаимное исключение – позволить процессу, находящемуся в критической секции, запрещать все прерывания. Однако этот способ непригоден, так как опасно доверять управление системой пользовательскому процессу; он может надолго занять процессор, а при крахе процесса в критической области крах потерпит вся система, потому что прерывания никогда не будут разрешены.

С другой стороны, для ядра характерна блокировка прерываний для некоторых команд при работе с переменными или списками. Возникновение прерывания в момент, когда, например, список готовых процессов находится в неопределенном состоянии, могло бы привести к состоянию состязания. Запрет прерываний бывает полезным в самой ОС, но это решение неприемлемо в качестве механизма взаимного исключения для пользовательских процессов.

2.3.2. Переменные блокировки.

С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным процессом), и значение 0, если ресурс занят. Ниже показан фрагмент алгоритма процесса, использующего для реализации взаимного исключения доступа к разделяемому ресурсу D блокирующую переменную F(D). Перед входом в критическую секцию процесс проверяет, свободен ли ресурс D. Если он занят, то проверка циклически повторяется, если свободен, то значение переменной F(D) устанавливается в 0, и процесс входит в критическую секцию. После того, как процесс выполнит все действия с разделяемым ресурсом D, значение переменной F(D) снова устанавливается равным 1.

Если все процессы написаны с использованием вышеописанных соглашений, то взаимное исключение гарантируется. Следует заметить, что операция проверки и установки блокирующей переменной должна быть неделимой. Пусть в результате проверки переменной процесс определил, что ресурс свободен, но сразу после этого, не успев установить переменную в 0, был прерван. За время его приостановки другой процесс занял ресурс, вошел в свою критическую секцию, но также был прерван, не завершив работы с разделяемым ресурсом. Когда управление было возвращено первому процессу, он, считая ресурс свободным, установил признак занятости и начал

выполнять свою критическую секцию. Таким образом, был нарушен принцип взаимного исключения, что может привести к нежелательным последствиям.

Во избежание таких ситуаций в системе команд машины можно иметь единую **команду "проверка-установка"**, или же реализовывать системными средствами соответствующие программные примитивы, которые бы запрещали прерывания на протяжении всей операции проверки и установки. Гарантируется, что операция "проверка-установка" неделима – другой процесс не может обратиться к слову в памяти, пока команда не выполнена. Процессор, выполняющий такую команду, блокирует шину памяти, препятствуя обращениям к памяти со стороны остальных процессоров, кроме того, работа процесса не может быть прервана в ходе выполнения данной команды.

2.3.3. Алгоритм Петерсона.

В 1981 году Петерсон (G. L. Peterson) придумал алгоритм взаимного исключения, представленный ниже и состоящий из двух процедур, написанных на С.

```
#define N 2                                   /* Количество процессов */
int turn;                                     /* Чья сейчас очередь? */
int interested[N];                            /* Все переменные изначально равны 0
                                              (FALSE) */
void enter_region(int process):               /* Процесс 0 или 1 */
        {
        Int other;                                  /* Номер второго процесса */
        other = 1 - process:                        /* "противоположный" процесс */
        interested[process] = TRUE;                 /* Индикатор интереса */
        turn = process:                             /* Установка флага */
        while (turn == process && interested[other] == TRUE)   /* Пустой цикл */;
        }
void leave_region(int process)     /* Процесс, покидающий критическую область */
        {
        interested[process] - FALSE; /*Индикатор выхода из критической области*/
        }
```

Прежде чем обратиться к совместно используемым переменным (то есть перед тем, как войти в критическую область), процесс вызывает процедуру enter_region со своим номером (0 или 1) в качестве аргумента.

Поэтому процессу при необходимости придется подождать, прежде чем входить в критическую область. После выхода из критической области процесс вызывает процедуру leave_region, чтобы обозначить свой выход и тем самым разрешить другому процессу вход в критическую область.

Пусть процесс 0 вызывает enter_region, задает элементы массива и устанавливает переменную turn равной 0. Поскольку процесс 1 не заинтересован в попадании в критическую область, происходит возврат из процедуры. Теперь, если процесс 1 вызовет enter_region, ему придется подождать, пока interested [0] примет значение FALSE, а это произойдет только в тот момент, когда процесс 0 вызовет процедуру leave_region при покидании критической области.

Например, оба процесса вызвали enter_region практически одновременно. Оба запомнят свои номера в turn. Но сохранится номер того процесса, который был вторым, а предыдущий номер будет утерян. Предположим, что вторым был процесс 1, отсюда значение turn равно 1. Когда оба процесса дойдут до конструкции white, процесс 0 войдет в критическую область, а процесс 1 останется в цикле и будет ждать, пока процесс 0 выйдет из нее.

Реализация критических секций с вышеописанных механизмов имеет существенный недостаток: в течение времени, когда один процесс находится в критической секции, другой процесс, которому требуется тот же ресурс, будет выполнять рутинные действия по опросу блокирующей переменной, бесполезно тратя процессорное время.

2.4. Классические проблемы межпроцессного взаимодействия.

Существует четыре основных типа проблем межпроцессного взаимодейсвтия:

Проблема монопольного использования двух или нескольких ресурсов. Для работы процессу нужно монопольно завладеть двумя ресурсами. Обладание только одним из них недостаточно, поэтому следует

избегать ситуации, когда каждый работающий процесс завладевает лишь одним ресурсом, не давая завершиться другому и не имея возможность завершиться самому (ситуация взаимоблокировки, но в данном случае она разрешается за счёт дополнительных вставок в код программ, а не средствами операционной системы)

Проблема использования буфера на одну запись. Программа – производитель пишет в буфер, программа – потребитель, читает из него. Производитель не должен писать в буфер, пока информация из него не будет прочитана потребителем, потребитель не должен читать, пока буфер пуст.

Проблема использования бесконечно большого буфера. Программа – производитель пишет в буфер, программа – потребитель, читает из него. Потребитель не должен читать из пустого буфера.

Проблема использования буфера на несколько записей. Аналогична проблеме с буфером на одну запись, но требует от производителя учёта записей в буфере.

3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1. Задание на лабораторную работу.

Необходимо в соответствии с индивидуальным вариантом задания (см. табл. 1) написать программы двух типов, содержащую потоки, предусмотрев при этом в каждой механизмы предотвращения гонок.

Программы выполняют обращения к файлам с заранее заданным периодом (рекомендуется 1 раз в 10 – 20 секунд), определяемым числом, вводимым пользователем при запуске программы, но необходимо предусмотреть средства, позволяющие пользователю принудительно выполнять обращения к файлам до истечения указанного времени (например по нажатию определённой кнопки на панели окна приложения).

Так как сами обращения к файлам достаточно быстротечны, то для того, чтобы процессы ожидания освобождения ресурсов были наглядны, в

код программы необходимо вести операторы, формирующие задержку (порядка 5 секунд) при работе с файлами.

Для визуализации состояния программы (простаивает, работает с ресурсом, ожидает освобождения ресурса) предусмотреть средства индикации в рабочем окне программы.

3.2. Индивидуальные варианты заданий.

Таблица 1. – Варианты заданий.

Номер варианта	Действия «Программы 1»	Действия «Программы 2»	Примечания
1.	Записать в конец любых двух файлов из трёх* заданную заранее текстовую стоку	Записать в конец любых двух файлов из трёх заданную заранее текстовую стоку	Программа открывает оба файла одновременно, а не по очереди
2.	Писать в конец рабочего файла текстовую строку	Удалять из начала рабочего файла текстовую строку	Недопустимо удаление строки из пустого файла
3.	Писать в конец рабочего файла текстовую строку	Удалять из начала рабочего файла текстовую строку	В файле не менее 2 и не более 6 строк
4.	Писать в рабочий файл строку	Выводить записанную первой программой строку на экран	Не писать в файл, если его ещё не прочитали, не читать, если ничего не записали
5.	Писать в конец рабочего файла текстовую очередную строку из заранее подготовленного файла	Инвертировать (записать задом наперёд) текстовую строку	Писать только незаписанную строку. Инвертировать только последнюю неинвертированную строку.
6.	Писать в конец рабочего файла текстовую очередную строку из заранее подготовленного файла	Конкатенировать две строки файла	Писать только незаписанную строку. Конкатенировать только две последние неконкатенированные строки.
7.	Записать в конец любых двух файлов из трёх заданную заранее текстовую стоку	Записать в начало любых двух файлов из трёх заданную заранее текстовую стоку	Программа открывает оба файла одновременно, а не по очереди

* Файлы выбираются случайно.

Номер варианта	Действия «Программы 1»	Действия «Программы 2»	Примечания
8.	Писать строку к конец одного файла из двух	Менять местами строки двух файлов	Не менять местами строки файлов, если хотя бы в одном их файлов все строки уже поменяны
9.	Писать строку в конец одного файла из двух	Удалить первые строки из обоих файлов	Не удалять строки, если хотя бы один из файлов пуст
10.	Писать строку к конец двух файлов	Удалить первую строку из одного файла из двух	Не писать строки, если в хотя бы одном из файлов строк больше 5
11.	Дописывать строку в файл только в нечётные позиции	Дописывать строку в файл только в чётные позиции	Не дописывать строку, если последняя позиция не удовлетворяет по чётности
12.	Дописывать строку в файл	Удаляет 6 первых строк из файла, если число строк больше 6 и кратно 3	
13.	Дописывать строку в файл только в позиции, номера которых кратны трём	Дописывать строку в файл только в позиции, номера которых не кратны 6	
14.	Писать две строки в конец файла	Удалить три последние строки из файла	Не писать строки, если в файле более 7 строк
15.	Дописывать строку в файл	Дописывать строку в предпоследнюю позицию файла	
16.	Дописывать строку в файл	Дописывать к строке, записанной 1–й программой определенную последовательность	Вторая программа на выполняет своих действий, если в файле нет необработанных строк
17.	Писать две строки в начало файла	Удалить первую и третью строки файла	Не удалять, если это невозможно
18.	Писать две строки в начало файла (в первую и третью позиции)	Удалить две строки из конца файла	Не писать строки, если в файле более 9 строк

Студенты формируют бригады по два человека в каждой. Бригада выполняет вариант, указанный преподавателем.

3.3. Обеспечение монопольного использования разделяемого ресурса

Гонки при доступе к разделяемому ресурсу предотвращаются за счёт использования механизма блокирующих переменных. Блокирующей переменной может служить содержимое какого либо служебного файла. Обращаясь к нему, программа читает его запись. Если она нулевая, то записывает ненулевую и закрывает файл, устанавливая тем самым блокирующую переменную и получая монопольный доступ к ресурсу. Если запись ненулевая, то программа периодически читает файл, ожидая его обнуления, которое выполняет только программа, завершающая работу с разделяемым ресурсом

3.4. Указания к выполнению работы.

При работе с фалами программы входят в критические секции, связанные с разделяемыми ресурсами (файлами и переменными, хранящими параметры работы с файлами). Для предотвращения гонок использовать внутренние механизмы работы с критическими секциями среды Delphi/

Рекомендуется выбирать расширение рабочих файлов «.htm», чтобы иметь возможность быстро отслеживать изменения, вносимые в них программой с помощью браузера. Чтобы иметь возможность контролировать работу программ, строки, добавляемые ими, не должны повторяться.

3.5. Содержание отчёта

1) Вариант задания.
2) Листинг программ.
3) Вид главного окна программ.
4) Временная диаграмма работы программ, отражающая их состояния в течение 3 – 4 минут работы (см. пример в табл. 2)

Таблица 2. – Образец временной диаграммы.

Время с начала работы	Состояние потока 1	Состояние потока 2	Содержимое рабочих файлов

4. КОНТРОЛЬНЫЕ ВОПРОСЫ.

1) Дать определение понятию «критическая секция».

2) Что такое разделяемый ресурс? Привести несколько реальных примеров использования программами разделяемых ресурсов.

3) Для чего необходимо предусматривать средства устранения гонок процессов?

4) Использование переменных блокировки при обращении к разделяемым ресурсам.

5) Альтернативные использованию блокирующих переменных механизмы устранения проблемы гонок процессов.

6) Основные преимущества и недостатки использования блокирующих переменных при устранении проблемы гонок процессов.

7) Алгоритм Питерсона.

8) Перечислить основные типы проблем межпроцессного взаимодействия и охарактеризовать каждый их них

9) Охарактеризовать тип проблемы межпроцессного взаимодействия, решаемой в ходе выполнения данной лабораторной работы

5. БИБЛИОГРАФИЧЕСКИЙ СПИСОК.

1) Таненбаум Э., Вудхал А. Операционные системы: разработка и реализация. – СПб.: Издательский дом «Питер», 2006.

2) Олифер В.Г., Олифер Н.А.. Сетевые операционные системы. - СПб.: Издательский дом «Питер», 2003.

3) Ахо В., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. - М.: Издательский дом «Вильямс», 2001.

4) Вильямс А. Системное программирование в Windows 2000 для профессионалов. - СПб.: Издательский дом «Питер», 2001.

5) Гордеев А.В., Молчанов А.Ю.. Системное программное обеспечение – Спб.: Питер, 2001. – 736с. илл.

6) Гордеев, А.В., Кучин, Н.В. Проектирование взаимодействующих процессов в операционных системах: Учебное пособие. – Л.: ЛИАП, 1991. - 72 с.

7) Кэнту, М. Delphi 5 для профессионалов. – СПб.: Издательский дом «Питер», 2001.