МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ДАЛЬНЕВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ШКОЛА ЕСТЕСТВЕННЫХ НАУК

РЕАЛИЗАЦИЯ МЕХАНИЗМА ВЗАИМОИСКЛЮЧЕНИЙ С ПОМОЩЬЮ СЕМАФОРОВ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ “СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ ”

Разработала ст.преподаватель кафедры
Информационных систем управления
Елсукова Е.А

Владивосток 2011

ЛАБОРАТОРНАЯ РАБОТА

Реализация механизма взаимоисключений с помощью семафоров

ЦЕЛЬ РАБОТЫ: Научиться использовать средства, предоставляемые операционной системой – семафоры, мониторы, очереди сообщений для организации параллельных взаимодействующих процессов.

Один из главных вопросов, на котором сосредотачивается внимание разработчиков приложений, связан с управлением процессами и потоками в многозадачной системе. Основным требованием поддержки параллельных процессов является обеспечение взаимоисключений – возможность выполнения только одного процесса с блокированием работы всех остальных процессов. Существуют 3 подхода к обеспечению взаимоисключений: программный, аппаратный механизм, средствами операционный системы (ОС). Современные ОС имеют различные собственные средства, обеспечивающие параллельные вычисления. К этим средствам относятся семафоры, мониторы, очереди сообщений, почтовые ящики, конвейеры. Семафоры являются простейшими из них.

Основной принцип обеспечения взаимоисключений заключается в том, что два или большее количество процессов могут сотрудничать посредством простых сигналов, так что в определенном месте процесс может приостановить работу до тех пор, пока не дождется соответствующего сигнала. Требования кооперации любой степени сложности могут быть удовлетворены соответствующей структурой сигналов. Для сигнализации используются специальные переменные, называющиеся семафорами. Для передачи сигнала через семафор S процесс выполняет примитив P(S), а для получения сигнала – примитив V(S). В последнем случае процесс приостанавливается до тех пор, пока не осуществится передача соответствующего сигнала.

Для достижения желаемого результата можно рассматривать семафор как переменную имеющее целое значение, над которой определены 3 операции:

1. Семафор может быть инициализирован неотрицательным значением.
2. Операция P(S) уменьшает значение семафора. Если это значение становится отрицательным, процесс, выполняющий операцию P(S) блокируется.
3. Операция V(S) увеличивает значение семафора. Если это значение не положительно, то заблокированный операцией P(S) процесс, деблокируется.

Не имеется никаких иных способов получения информации о значении семафора или изменения его, кроме перечисленных.

В листинге 1 приведено более формальное определение примитивов семафоров. Предполагается что примитивы P(S) и V(S) атомарны (не могут быть прерваны) и каждая из подпрограмм может рассматриваться как единый шаг.

Листинг **1. Определение семафорных примитивов**

```
struct semaphore {
     int count;
     queueType queue; }
```

примитив P(S)

```
void  P(semaphore S)
  { S.count - -;
    if (S.count < 0 )
 { Поместить процесс в S.queue
   Заблокировать процесс}
 }
```

примитив V(S)

```
void  V(semaphore S)
{ S.count + +;
if (S.count <= 0 )
 {Удалить процесс из S.queue
  Поместить процесс в список активных}
}
```

В листинге 2 представлена ограниченная версия семафора – двоичный семафор. Двоичный семафор может принимать значения 0 или 1. Задачи, решаемые с применением обычных семафоров, могут быть решены и с использованием лишь двоичных семафоров.

Листинг **2. Определение примитивов двоичного семафора.**

```
struct  binary semaphore {
     enum {zero, one} value;
     queueType queue; }
```

примитив P(S)

```
void  PBin(binary_semaphore
S)
 {
    if (S.value = 1 )
S.value=0;
    else
  { Поместить процесс в S.queue
    Заблокировать процесс}
 }
```

примитив V(S)

```
void  VBin(semaphore S)
{
if (S. queue.is_empty())
S.value=1;
else
 {Удалить процесс из S.queue
  Поместить процесс в список
активных}
 }
```

В листинге 3 показано простое решение задачи взаимоисключений с использованием семафора S. Пусть имеется n процессов идентифицируемых массивом Proc(i). В каждом из процессов перед входом в критический раздел выполняется вызов P(S). Если значение S становится отрицательным, процесс приостанавливается. Если же значение S равно 1,оно уменьшается до 0 и процесс немедленно входит в критический раздел; поскольку S , больше не является положительным, никакой другой процесс не может войти в критический раздел.

Листинг 3. Взаимоисключения с использованием семафоров.

```
/*  программа взаимного исключения */
const  int n= /*  количество процессов */
 semaphore  S=1;
 void  Proc (int i)
{  while (True)
   { P (S);
     /*  критическая секция */;
     V (S);
    /*  остальной код i-го процесса */;
   }
}
void main ()
{
   parbegin
     Proc(1), Proc(2), …., Proc(n);
   parend;
 }
```

Следующим шагом является введение такой системы управления, в которой группа процессов является равноправной, причем каждый из них выполняется с собственной скоростью, зависящей в частности, от его взаимодействия с внешними объектами и другими процессами. Такая модель носит название МОДЕЛЬ АСИНХРОННЫХ

ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ, а режим работы операционной системы, в котором она осуществляется, называется, соответственно, РАЗДЕЛЕНИЕМ ВРЕМЕНИ.

Задача «Читатели — писатели»

Важной и часто встречающейся задачей, решение которой требует синхронизации, является задача «Читатели — писатели». Эта задача имеет много вариантов. Определить ее можно следующим образом. Имеются данные, совместно используемые рядом процессов. Данные могут находиться в файле в блоке основной памяти или даже в регистрах процессора. Имеются несколько процессов, которые только читают эти данные (*Читатели*), и несколько других, которые только записывают данные (*Писатели).*
При этом должны удовлетворяться следующие условия.

- Любое число читателей могут одновременно читать файл.
- Записывать информацию в файл в определенный момент времени может только один Писатель.
- Когда Писатель записывает информацию в файл, ни один Читатель не может его читать.

Пример использования – работа с библиотечным каталогом. Другим типичным примером служит система автоматизированной продажи билетов. Процессы «*Читатели*» обеспечивают нас справочной информацией о наличии свободных билетов на тот или иной рейс. Процессы «*Писатели*» запускают с пульта кассира, когда он оформляет для нас тот или иной билет. Имеется большое количество как «*Читателей*», так и «*Писателей*».

Наиболее характерная область использования этой задачи в вычислительной системе — при построении систем управления файлами. Два класса процессов имеют доступ к некоторому ресурсу (области памяти, файлам). «*Читатели*» — это процессы, которые могут параллельно считывать информацию из некоторой общей области памяти, являющейся критическим ресурсом. «*Писатели*» — это процессы, записывающие информацию в эту область памяти, исключая при этом и друг друга и процессы «*Читатели*».

Широко распространены следующие два условия::

1.Приоритетное чтение: Устанавливается приоритет в использование критического ресурса процесса *Читатели*. Это означает, что если хотя бы один *Читатель* пользуется ресурсом, то он закрыт для использования всем *Писателям* и доступен для использования всем *Читателям*. При появлении запроса от *Писателя* необходимо закрыть дальнейший доступ всем тем процессам *Читателям*, которые выдадут запрос на критический ресурс после него.

Пример решения задачи представлен в листинге 5.

При решении данной задачи используются:

- Семафор `W` – обеспечивает взаимоисключение, для доступа к разделяемой области памяти;
- Семафор `R` – отслеживает корректное обновление `NumR`, предохраняет от выполнения операций `P(W), V(W);`
- переменная `NumR` – предназначена для подсчета текущего числа процессов типа «Читатели», находящихся в критическом интервале.

Листинг 5. Решение задачи «Читатели — писатели» с приоритетом «читателей» в доступе к критическому ресурсу

```
Main ()
R,W semaphore=1;
int NumR= 0

Reader()                          Writer()
{ while (true)                     { while (true)
   P(R); NumR++;                      P(W);
```

```
      if numR=1 then P(W);
      V(R);
      Read_Data;
      P(R);
      NumR--;
      if NumR=0 then V(W);
      V(R);
   }
   parbegin
     Reader(); Writer();
   Parend.
```

```
      Write_Data;
       V(W);
   }
```

2.Приоритетная запись: Когда «Писатель» выполняет операцию V(W), неясно, какого типа процесс войдет в критический интервал. Чтобы гарантировать получение процессами «читателями» наиболее свежей информации, необходимо при постановке в очередь готовности использовать дисциплину обслуживания, учитывающую более высокий приоритет «писателей». Однако этого оказывается недостаточно, ибо если в критическом интервале продолжает находиться, по крайней мере, один «Читатель», то он не даст обновить данные, но и не воспрепятствует вновь приходящим процессам «читателям» войти свою критическую секцию. Необходим дополнительный семафор. Пример правильного решения этой задачи приведен в листинге 6.

Листинг 6. Решение задачи «Читатели — писатели» с приоритетом в доступе к критическому ресурсу первых с дополнительным семафором

```
Main ()
R,W,S semaphore=1;
int NumR= 0

Reader()
{ while (true)
   P(S); P(R); NumR++;
   if numR=1 then P(W);
   V(S); V(R);
   Read_Data;
   P(R);
   NumR--;
   if NumR=0 then V(W);
   V(R);
}
parbegin
  Reader; Writer;
Parend.
```

```
Writer()
{ while (true)
   P(S); P(W);
   Write_Data;
   V(S);  V(W);
}
```

ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ СЕМАФОРОВ

Задача 1. Моделирование работы кухни ресторана

В вычислительной системе моделируется работа кухни ресторана, где отец и 3 дочери готовят пиццу.

Приготовление пиццы требует трех составляющих: теста, соуса, сыра. Одна дочь должна непрерывно поставлять тесто, вторая – соус, третья – сыр. Приготовление пиццы происходит следующим образом: первая дочь формирует из теста лепешку для пиццы, после чего вторая дочь смазывает лепешку соусом, а третья – посыпает сыром. Отец берет подготовленную дочерьми пиццу и ставит ее в духовку.

СОВЕТ: Оформите модель приготовления пиццы в виде 4 процессов: для отца и для каждой из дочерей.

1. Используемые семафоры:

   SemT, SemSouce, SemChees, SemPech – обеспечивают синхронизацию работы соответствующих процессов;

   Все используемые в программе семафоры обобщенного типа: могут принимать значения в диапазоне от –N ÷ N , где N – количество запущенных процессов.

2. Решение задачи

```
void main ()
   Semaphore: SemT, SemSouce, SemChees, SemPech;
   initSemaphore:  SemT=0; SemSouce=0, SemChees=0, SemPech=0;
   process                 process                 process
   D1Testo;                D2Souce;                D3Cheese;
   begin                   begin                   begin
                             P(SemT)                 P(SemChees)
  /*Изготовление              /*смазываем              /*посыпаем
   лепешки теста*/         лепешку  соусом         лепешку сыром*/
     V(SemT);              */                        V(SemPech);
   end;                                              end;
                           V(SemSouce);
                           end;
   process FaPech;
   begin
   P(SemPech);
  /* ставим пиццу в печь*/
   end;

   parbegin
      D1Testo; D2Souce; D3Cheese; FaPech;
   Parend
```

Задача 2. Моделирование движения по мосту

В вычислительной системе моделируется движение самосвалом от карьера к заводу и обратно по дороге со старым мостом. Движение по мосту может осуществляться в обоих направлениях, но на нем не может быть одновременно более трех машин, иначе мост рухнет.

СОВЕТ: В виде отдельного процесса оформить движение самосвала.

1. Используемые семафоры:

   SemBridge – обеспечивает взаимоисключение при работе с разделяемым ресурсом – мостом (в данной задаче).

   В программе используется семафор обобщенного типа: SemBridge может принимать значения в диапазоне от –N ÷ N , где N – количество запущенных процессов.

2. Решение задачи

```
void main ()
   I: количество самосвалов в системе, i=1..n;
   Semaphore:SemBridge;
   initSemaphore: SemBridge =3;

   process MoveCar(i);
   begin
     /*Движение по дороге */
    P(SemBridge);
     /*Движение по мосту*/
   V(SemBridge);
     /*Движение по дороге */
   end;

   parbegin
      MoveCar(1); MoveCar(2);.. MoveCar(i)…; MoveCar(n);
   Parend
```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Решить задачу с использованием семафоров
    a. Опишите назначение и укажите тип (обобщенный, двоичный) используемых семафоров.
    b. Выделите в программе критические секции и поясните, почему этот фрагмент является критической секцией.

Варианты задач

1. Задача о парикмахерской

В парикмахерской 3 кресла, 3 парикмахера, зал ожидания, в котором 4 клиента могут разместиться на диване остальные стоя. Правила пожарной безопасности ограничивают общее количество клиентов внутри помещения 20 людьми. Предполагаем, что всего мастерская должна обслужить 50 клиентов.

Клиент не может войти в парикмахерскую, если она полностью заполнена другими клиентами. Оказавшись внутри, клиент либо присаживается на диван или стоит, если диван занят. Когда мастер освобождается, к нему отправляется наиболее долго ожидающий клиент с дивана; если имеются стоящие клиенты, то тот из них кто ожидает дольше других, присаживается на диван. По окончании стрижки принять плату может любой парикмахер, но так как кассир всего лишь один плата принимается в один момент времени только от одного клиента. Рабочее время парикмахера разделяется на стрижку, принятие оплаты о клиента и сон в кресле в ожидании очередного клиента.

2. Прогулка по парку Юрского периода

Парк Юрского периода состоит из музея динозавров и безвольерного зоопарка. Имеется M пассажиров и N одноместных машин. Пассажиры некоторое время проводят в музее, а затем на машине посещают зоопарк. Если имеется свободная машина, в ней размещается один пассажир, который совершает поездку по парку в течение некоторого (случайного) промежутка времени. Если все машины заняты, то пассажиры, желающие посетить парк, ожидают, пока машины не освободятся. Если есть свободные машины, но нет ожидающих их пассажиров, то в состоянии ожидания находятся машины. Используйте семафоры для синхронизации M процессов пассажиров и N процессов машин.

3. Моделирование производственной линии

Необходимо смоделировать производственную линию, производящую Виджеты. Каждый Виджет состоит из Детали C и Модуля 1. Модуль 1 состоит из Детали A и Детали B. Изготовление Детали A требует 2 секунды, Детали B - 3 секунды и Детали C - 4 секунды.

4. Подготовка к Новому году

Дед Мороз спит в своей избушке на Северном полюсе и может быть разбужен только 1) 5 сестрами Снегурочками, вернувшимися из отпуска из Крыма, или 2) несколькими Снеговиками, у которых возникли проблемы при изготовлении игрушек. Чтобы дать Деду Морозу поспать, Снеговики будят его только в том случае, когда проблемы возникают у 3 из них. Когда трое Снеговиков решат свои вопросы, все остальные Снеговики должны ждать пока не вернется побывавшая у него тройка. Если проснувшийся Дед Мороз обнаруживает у дверей избушки и Снеговиков и последнюю из Снегурочек, то он просит Снеговиков обождать со своими проблемами до окончания празднования Нового года, поскольку куда важнее ехать поздравлять малышей. (Снегурочки не спешат возвращаться из Крыма и остаются там до последнего момента.) Прибывшая последней Снегурочка идет будить Деда Мороза пока остальные занимаются макияжем в своих комнатах.

5. Производитель и Потребитель

Напишите две программы, Производитель и Потребитель, такие, что Производитель заполняет буфер в разделяемой памяти, а Потребитель читает его. Производитель должен помещать новые данные в буфер только после того, как Потребитель прочитает его.

Совет: Для синхронизации можно использовать два семафора (эффективнее всего - набор из двух семафоров). Один из семафоров нужно ассоциировать с записью новых данных в буфер. Другой должен быть ассоциирован с чтением данных Потребителем.

6. Один производитель и несколько потребителей буфера

Напишите программу-производитель, которая помещает текст в буфер, размещенный в сегменте разделяемой памяти. Напишите программу-потребитель, которая будет читать из этого буфера. Может существовать несколько копий потребителя. Производитель может обновлять буфер только после того, как все потребители считали его содержимое.

Совет: Нужны два семафора, но их значения будут меняться в диапазоне от 0 до количества потребителей.

7. Несколько читающих процессов и один эксклюзивный процесс записи в разделяемую память

Предположим, что исполняется N процессов. Часть из них читает, а остальные пишут в разделяемый сегмент памяти. Несколько читающих процессов могут работать с буфером одновременно. Если один из пишущих процессов выполняет запись, все остальные процессы должны ждать. Кроме того, если пишущий процесс хочет обновить данные, он должен ждать, пока все читающие процессы закончатся. Для обеспечения такого взаимного исключения надо использовать семафоры.

Совет: Один из способов решения состоит в использовании набора из трех семафоров со следующими значениями:

- индекс 0 счетчик процессов, выполняющих чтение
- индекс 1 двоичный семафор, гарантирующий, что только один процесс может писать в буфер
- индекс 2 используется для блокировки чтения на время записи или ожидания записи.

Список используемой литературы

1. Столлингс В. Операционные системы – М: Издательский дом “Вильямс”, 2010. – 848с.
2. Бэкон Д., Харрис Т. Операционные системы: Параллельные и распределенные системы. – СПб.:Питер, Киев: Издательская группа BHV, 2009. – 800с.