*Лабораторная работа 3*
# СИНХРОНИЗАЦИЯ ПРОЦЕССОВ

*Цель лабораторной работы – изучение примитивов синхронизации и методов работы с ними, решение классической задачи узкого моста и тестирование решения в рамках операционной системы Pintos.*

Основная рабочая директория: tests/threads/narrow-bridge.c

## Теоретические сведения

Семафоры традиционно использовались для синхронизации процессов, обращающихся к разделяемым данным. Каждый процесс должен исключать для всех других процессов возможность одновременно с ним обращаться к этим данным. Когда процесс обращается к разделяемым данным, говорят, что он находится в своем критическом участке.

Для решения задачи синхронизации необходимо, в случае если один процесс находится в критическом участке, исключить возможность вхождения для других процессов в их критические участки. Хотя бы для тех, которые обращаются к тем же самым разделяемым данным. Когда процесс выходит из своего критического участка, то одному из остальных процессов, ожидающих входа в свои критические участки, должно быть разрешено продолжить работу.

Процессы должны как можно быстрее проходить свои критические участки и не должны в этот период блокироваться. Если процесс, находящийся в своем критическом участке, завершается (возможно, аварийно), то необходимо, чтобы некоторый другой процесс мог отменить режим взаимоисключения, предоставляя другим процессам возможность продолжить выполнение и войти в свои критические участки.

Механизм семафоров, реализованный в ОС Pintos, включает в себя особый тип целочисленной неотрицательной переменной и две специальные операции, которые управляют ей:

- Операция "Down" или "P", которая ожидает, когда переменная семафора станет положительной, а затем декрементирует ее;
- Операция "Up" или "V", которая инкрементирует переменную семафора и пробуждает один ожидающий процесс, если такой существует;

Классическое определение этих операций выглядит следующим образом:
$P(S)$: $while$ $(s == 0)$ процесс заблокирован;
$S = S - 1$;
$V(S)$: $S = S + 1$;

Эта запись означает следующее: при выполнении операции P над семафором S сначала проверяется его значение. Если оно больше 0, то из S вычитается 1. Если оно меньше или равно 0, то процесс блокируется до тех пор, пока S не станет больше 0. После этого из S вычитается 1. При выполнении операции V над семафором S к его значению прибавляется 1. В момент создания семафор может быть инициализирован любым неотрицательным значением.

Подобные переменные-семафоры могут с успехом применяться для решения различных задач организации взаимодействия процессов. Операционная система Pintos обеспечивает атомарность операций P и V, используя метод запрета прерываний на время выполнения соответствующих

системных вызовов. Если при выполнении операции P заблокированными оказались несколько процессов, то порядок их разблокирования может быть произвольным, например, FIFO.

Реализация семафоров для ОС Pintos, расположенная в threads/synch.h, представлена в табл. 3.1.

**Таблица 3.1. Семафоры**

| **Объект:** | **Назначение:** |
|---|---|
| struct semaphore | Структура описывает основные параметры семафора:<br>struct semaphore<br>{<br>unsigned value; //текущее значение переменной семафора<br>struct list waiters; //список ожидающих процессов<br>}; |
| void sema_init<br>(struct semaphore *sema,<br>unsigned value) | Функция создает новый семафор с заданным значением value |
| void sema_down<br>(struct semaphore *sema) | Функция вызывает операцию"P" над семафором, ожидает, когда его значение value станет положительным, и затем декрементирует его |
| bool sema_try_down<br>(struct semaphore *sema) | Функция выполняет операцию"P" над данным семафором без ожидания. Возвращает значение "true", если операция успешно выполнена, или "false", если значение переменной семафора уже равно 0 и не может быть декрементировано |
| void sema_up<br>(struct semaphore *sema) | Функция выполняет операцию "V" над данным семафором, увеличивая на единицу значение его переменной value. Если несколько процессов ожидают семафор, то данная функция пробуждает один из них |

В ОС Pintos семафор, проинициализированный значением 0, ожидает событие, которое произойдет один раз. Например, в случае, когда некоторый процесс А вызывает другой процесс Б и ждет результатов его выполнения. Тогда процесс А может создать семафор, передать его процессу Б и выполнить операцию "P". Когда процесс Б завершит работу, он выполнит операцию "V". Семафор, инициализированный значением 1, обычно используется для контроля доступа к разделяемым ресурсам. Когда некоторый процесс начинает использовать данные, он выполняет операцию "P" и после окончания работы выполняет операцию "V". Семафоры могут быть проинициализированы и бо́льшими значениями, однако они редко используются.

## Порядок выполнения работы

1. Условие задачи:

В одном немецком городе издавна существует узкий мост, движение транспорта по которому ограничено следующими правилами:

1. Движение по мосту в каждый момент времени возможно **только в одном направлении**;
2. Если **три автомобиля** попытаются пересечь мост одновременно, он разрушится под их весом;
3. Автомобили «**Скорой помощи**» должны проезжать по мосту **вне очереди**.

2. Изучить файлы threads/synch.h и threads/synch.c: в них представлены функции для работы с семафорами в ОС Pintos, которые необходимо использовать при выполнении данной лабораторной работы.

3. Изучить содержимое теста narrow-bridge.c, narrow-bridge-test.c, расположенного в src/tests/threads. Данный тест моделирует задачу управления узким мостом для различного числа автомобилей и различных направлений их движения. По умолчанию при вызове процедуры test_narrow_bridge() создаются процессы для указанного количества автомобилей и проводится ожидание завершения всех созданных процессов.

4. Внести изменения в код данного теста в файле narrow-bridge.c, реализуя задачу узкого моста. В операционной системе Pintos будем представлять один автомобиль одним процессом, который, выполняет следующую процедуру:

```
void one_vehicle(enum car_priority prio, enum car_direction dir)
{
    arrive_bridge(prio, dir);
    cross_bridge(prio, dir);
    exit_bridge(prio, dir);
}
```

Здесь параметр dir указывает направление, в котором будет пересечен мост, и принимает значение dir_left или dir_right. Параметр prio указывает на приоритет данного автомобиля и принимает значение car_normal, если это обычный автомобиль, или car_emergency, если это машина «Скорой помощи». Естественно, автомобиль с высоким приоритетом должен как можно скорее пересечь мост.

4.1 Продумайте систему семафоров, которая будет соответствовать установленным правилам и регулировать движение по мосту без образования заторов (голодания процессов) и разрушения моста. Для инициализации объектов синхронизации и глобальных переменных следует реализовать код в функции narrow_bridge_init().

4.2 Процедура cross_bridge является промежуточной и осуществляет печать нескольких отладочных сообщений, сигнализирующих о въезде и выезде автомобиля с моста.

4.3 Реализуйте процедуры arrive_bridge() и exit_bridge(). В процедуре arrive_bridge должна оцениваться возможность пересечения автомобилем моста: текущая дорожная ситуация должна быть проанализирована и въезд на мост должен быть заблокирован до тех пор, пока автомобиль не сможет безопасно пересечь мост. Процедура exit_bridge должна соответственно, снимать установленные ограничения на въезд.

Убедитесь, что в вашем решении:
— Не производится отключение прерываний. Взаимоисключение процессов при обращении к общим областям памяти необходимо производить с помощью объектов синхронизации: семафоров и замков.
— Не производится явное обращение к полям value и waiters семафоров, а также holder замков. Обращаться к этим полям можно только при отключенных прерываниях, но в данной работе синхронизация таким способом не производится.
— Не вызываются функции thread_block и thread_unblock. Данные функции являются внутренними и вызываются ядром ОС pintos в моменты "захвата" и "освобождения" объектов синхронизации. В данной работе задачу необходимо решать с помощью примитивов синхронизации, а не явного переключения состояний процессов.
— Для решения задачи не используются механизмы приоритетного планирования процессов в ядре ОС, т.е. все процессы, моделирующие автомобили, имеют одинаковый приоритет. При этом очередность проезда моста машинами с разным приоритетом определяется не приоритетом процессов, моделирующих автомобили, а реализуется логическими условиями и(или) порядком захвата/освобождения семафоров.

— Исключены параллельные чтение и(или) запись значений глобальных переменных из нескольких процессов. Любые обращения к глобальным областям памяти из нескольких процессов должны быть синхронизированы с помощью примитивов синхронизации.

— Отсутствует "активное ожидание". Процесс, моделирующий автомобиль, стоящий в очереди, должен иметь состояние THREAD_BLOCKED, т.е. ожидать какого-либо примитива синхронизации (семафора) и пробуждаться только когда подошла его очередь пересекать мост.

— Изменения внесены только в файл narrow-bridge.c, причем в прототипы функций narrow_bridge_init, arrive_bridge и exit_bridge не добавлено каких-либо дополнительных параметров.

```
void cross_bridge(enum car_priority prio, enum car_direction dir)
{
    msg("Vehicle: %4s, prio: %s, direct: %s, ticks=%4llu",
          thread_current()->name,
          prio == car_emergency ? "emer" : "norm",
          dir == dir_left ? "l -> r" : "l <- r",
          (unsigned long long) timer_ticks ());

    timer_sleep(10);
}
```

Ожидание в конце данной процедуры симулирует проезд автомобиля по мосту. Если на мосту находится одновременно несколько автомобилей, распечатываемое значение ticks для них будет равно или незначительно отличаться. Если значение отличается на указанное в timer_sleep значение, то такие автомобили проезжают мост не одновременно.

5. Запустите тесты в командной строке Pintos и проанализируйте полученные результаты. Количество автомобилей указывается через аргументы командной строки ядра pintos в следующем порядке: автомобилей слева, автомобилей справа, автомобилей «скорой помощи» слева, автомобилей «скорой помощи» справа:

```
pintos --qemu --  -q run "narrow-bridge 0 0 0 0"
pintos --qemu --  -q run "narrow-bridge 0 0 0 1"
pintos --qemu --  -q run "narrow-bridge 0 4 0 0"
pintos --qemu --  -q run "narrow-bridge 0 0 4 0"
pintos --qemu --  -q run "narrow-bridge 3 3 3 3"
pintos --qemu --  -q run "narrow-bridge 4 3 4 3"
pintos --qemu --  -q run "narrow-bridge 7 23 17 1"
pintos --qemu --  -q run "narrow-bridge 40 30 0 0"
pintos --qemu --  -q run "narrow-bridge 30 40 0 0"
pintos --qemu --  -q run "narrow-bridge 23 23 1 11"
pintos --qemu --  -q run "narrow-bridge 22 22 10 10"
pintos --qemu --  -q run "narrow-bridge 0 0 11 12"
pintos --qemu --  -q run "narrow-bridge 0 10 0 10"
pintos --qemu --  -q run "narrow-bridge 0 10 10 0"
```

Особое внимание обратите на ситуации, когда количество желающих пересечь мост автомобилей велико и неравномерно с левой и правой стороны моста.

6. Получите у преподавателя индивидуальное дополнительное задание и реализуйте его в указанном файле. Проанализируйте полученное решение и результаты работы, приведите их в отчете.

## Содержание отчета

В отчете необходимо указать следующие данные:

1. Описание разработанного алгоритма синхронизации и обоснование его эффективности:
   - по какому алгоритму в разработанном решении работают примитивы синхронизации и причина выбора данного алгоритма;
   - как определяется количество автомобилей, пересекающих мост в один момент времени;
   - каким образом разработанное решение гарантирует, что автомобили «Скорой помощи» пересекут мост вне очереди;
2. Исходный код реализованных функций narrow_bridge_init(), arrive_bridge() и exit_bridge() с подробными комментариями
3. Диаграмма взаимодействия процессов на «узком мосте» в зависимости от их количества и состояний примитивов синхронизации. Показать, как осуществляется ограничение доступа к разделяемому ресурсу.
4. Полученный в результате выполнения программы вывод.
5. Результаты выполнения дополнительного индивидуального задания: описание алгоритма и его реализации, диаграмма состояния и взаимодействия процессов, результаты выполнения программ и анализ полученных результатов.