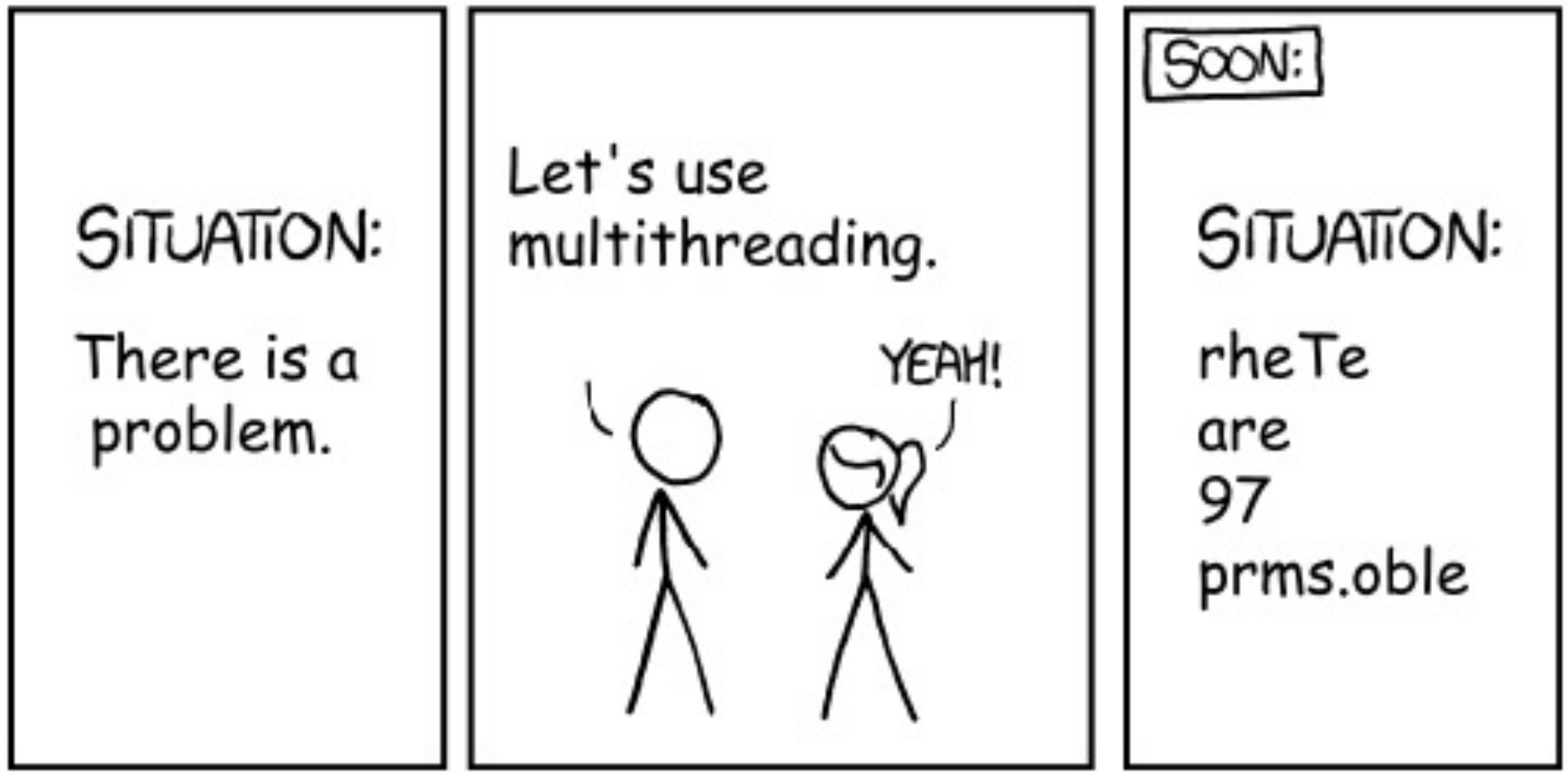
Конструирование ядра операционных систем (V)

Базовые примитивы синхронизации


SITUATION:
There is a problem.
Let's use multithreading.
YEAH!
SOON:
SITUATION:
rheTe
are
97
prms.oble

План

- Необходимость синхронизации доступа и язык C
- Блокирующие примитивы синхронизации
- Неблокирующие примитивы синхронизации
- Примеры ошибок при параллельном доступе
- Практическая часть лабораторной работы

Возможности платформы (1/2)

1. Можно ли параллельно считывать *обычную глобальную переменную* на языке C? На ассемблере x86_64 с помощью инструкции mov?
2. Можно ли параллельно записывать *обычную глобальную переменную* переменную на языке C? На ассемблере x86_64 с помощью инструкции mov?
3. Можно ли параллельно считывать и записывать?

Возможности платформы (2/2)

1. Операции чтения в C (и на ассемблере) thread-safe.
2. Операции записи в C non-thread-safe.
3. Некоторые операции записи могут быть thread-safe как в C (stdatomic.h), так и на ассемблере.

NB: Как только у вас появляется запись и любая другая операция, вам необходима синхронизация.

Критическая секция

- *Критическая секция* — участок в программе, в котором происходит обращение к некоторому разделяемому ресурсу.
- В момент времени внутри критической секции может находиться не более одного потока. Такой поток часто называют *владельцем критической секции*.
- Другой поток не может попасть в критическую секцию до момента, когда владелец критической секции выйдет из неё, т.е. *освободит критическую секцию*.
- Критическая секция может быть реализована аппаратно на уровне операции с разделяемым ресурсом (atomics).

Операции критической секции

- Захватить (lock) — поток ожидает освобождения критической секции и становится новым владельцем.
- Освободить (unlock) — владелец критической секции уступает право владения другому потоку.
- Выбор нового владельца может происходить по алгоритму для обеспечения справедливости (fairness).
  - Ticket Lock — en.wikipedia.org/wiki/Ticket_lock
  - ABQL — en.wikipedia.org/wiki/Array_Based_Queuing_Locks

Блокирующая синхронизация

В роли критической секции выступает внешняя сущность, например, ядро операционной системы:

- В момент захвата критической секции генерируется прерывание (обработка которого не может быть прервана) и осуществляется проверка состояния критической секции.
  - Если секция свободна, то поток становится новым владельцем и получает управление обратно.
  - Если секция занята, то поток не получает управления до момента освобождения критической секции и выбора его новым владельцем.
- В момент освобождения критической секции генерируется прерывание, которое обновляет состояние критической секции на свободное и при необходимости устанавливает нового владельца.

Примеры (1/2)

- *Мьютекс* — базовый примитив, типичные операции: lock, unlock, is_locked, try_unlock.
- *Рекурсивный мьютекс* — позволяет выполнять операции lock и unlock внутри критической секции владельцем.

```
void worker (void) {
    lock (&recursive_mutex);
    if (extra_condition) {
        lock (&recursive_mutex);
        /* work */
        unlock (&recursive_mutex);
    }
    unlock (&recursive_mutex);
}
```

Примеры (2/2)

- *Семафор* — примитив со счётчиком, который можно описать как критическая секция с N владельцами.
  - При создании семафор инициализируется с n <= N свободными слотами.
  - Операция lock() уменьшает n на 1, если n > 0 и захватывает блокировку. Иначе происходит ожидание достижения n > 0.
  - Операция unlock() увеличивает n на 1.
- *Рекурсивные семафоры* позволяют не учитывать одного и того же владельца повторно.
- *Бинарные семафоры* — синоним мьютекса.

Неблокирующая синхрон. (lock-free)

- Чтение и запись атомарной переменной происходит внутри критической секции на аппаратном уровне.
- Для создания примитивов синхронизации есть дополнительные операции с атомарными переменными:
  - Compare & Swap (CAS) — x86, SPARC: изменение атомарной переменной в случае совпадения её значения с переданным.
  - Load Link / Store Conditional (LL/SC) — ARM, MIPS, PowerPC:
    - Load Link — считывает значение атомарной переменной.
    - Store Conditional — обновляет значение атомарной переменной, если она не изменилась с момента операции Load Link.

Трудности атомарного доступа (1/2)

- Что если в атомарную переменную было записано текущее значение между LL/SC?
- Что если между операциями LL/SC произошло переключение контекста, но не было записи?
- Что если значение переменной было изменено на иное, а затем возвращено обратно (проблема ABA)? en.wikipedia.org/wiki/ABA_problem

Трудности атомарного доступа (2/2)

```
int atomic_variable = 0;
int global_variable = 0;

void thread1 (void) {
    global_variable = 1;
    atomic_write (&atomic_variable, 1);
}

void thread2 (void) {
   while (atomic_read (&atomic_variable) != 1) { }
   printf (“global_variable is %d\n”, global_variable);
}
```

Что выведет данный код?

Понятие барьера памяти (fence)

- Барьер определяет, что операции с неатомарной памятью должны быть выполнены в определённой точке:
  - Компиляторные барьеры запрещают перестановку операций компилятором после атомарной операции.
  - Процессорные барьеры запрещают перестановку операций на уровне процессора. Также они делают результаты операций видимыми для других ядер процессора.
- В языке C барьеры добавляются автоматически перед атомарными операциями для зависимых переменных в зависимости от memory order. Для seq_cst это происходит всегда.

Примеры

- *Спинлок* — базовый примитив, типичные операции: lock, unlock, is_locked, try_unlock.
  Реализуется поверх оперативной памяти с горячим ожиданием (busy wait) разблокировки.
- *Адаптивный* мьютекс — гибрид мьютекса и спинлока, в котором горячее ожидание заменяется переключением контекста через определённый промежуток времени.
- *Wait-free* алгоритмы — написание кода без горячего ожидания посредством использования read/write операций.

Потокобезопасность, реентерабельнсть

- *Потокобезопасность* — код работает корректно, если он вызывается из нескольких потоков одновременно.
- *Реентерабельность* — код работает корректно, если он может быть выполнен частично, прерван и выполнен отдельно, а затем завершён с места прерывания.

```
// Thread-safe but not reentrant :-)
_Thread_local int tmp;
int get_number (void) {
    tmp = rand ();
    return tmp;
}
```

Реентерабельнсть и рекурсия

```
void puts (const char *str) {
    if (!IO_available ())
        panic (“I/O console unavailable”);
    IO_write (str);
}

_Noreturn void panic (const char *reason) {
    puts (reason);
    halt ();
}
```

Ошибки класса TOCTOU

- *TOCTOU (Time Of Check Time Of Use)* — ситуация, когда проверка целостности ресурса разделена по времени с использованием ресурса, при этом ресурс находится в недоверенной среде, например, на диске.

```
int update_firmware (const char *filename) {
    if (!verify_signature (filename))
        return -1;
    // В этот момент нарушитель подменяет файл на диске.
    return upload_firmware (filename);
}
```

Взаимоблокировка (deadlock)

- *Взаимоблокировка* — ситуация, когда два процесса находятся в состоянии ожидания получения доступа к ресурсу, при этом ни один не может продолжить работу.

```
int get_number1 (void) {
    lock (&lock1); lock (&lock2);
    int tmp = rand ();
    unlock (&lock2); unlock (&lock1);
    return tmp;
}
int get_number2 (void) {
    lock (&lock2); lock (&lock1);
    int tmp = rand ();
    unlock (&lock1); unlock (&lock2);
    return tmp;
}
```

Динам. взаимоблокировка (livelock)

- *Динамическая взаимоблокировка* — ситуация, когда система выполняет работу, но не функционирует.

Пример: два пешехода идут по узкому коридору с разных сторон. Если пешеход встретил кого-то на своём пути, он возвращается к началу коридора и начинает идти по нему заново. При условии, что пешеходы двигаются с одной скоростью, ни один из них не выйдет из коридора.

Ссылки

- http://svr-pes20-cppmem.cl.cam.ac.uk/cppmem — интерактивная модель памяти языков C (см. также Lab1-MEM1, Lab1-MEM2)
- https://cwe.mitre.org/data/definitions/367.html https://wiki.sei.cmu.edu/confluence/display/c/FIO45-C.+Avoid+TOCTOU+race+conditions+while+accessing+files TOCTOU и примеры реализации атак

Спасибо за внимание!

Вопросы?