Домашнее задание 3.
Потоки, планирование и синхронизация.

Я

March 15, 2016

Contents

1 Основное задание

В этом домашнем задании вам необходимо научиться создавать/завершать/планировать потоки исполнения, а также реализовать простую синхронизацию потоков исполнения.

С появлением потоков, вам придется внести изменения в уже написанный код, так чтобы он стал потокобезопасным. Скорее всего вам понадобится исправить алокатор страниц и алокатор объектов фиксированного размера (SLAB).

Для всех заданий должны быть предусмотрены тесты, т. е. создание потоков, использование взаимного исключения, переключение потоков и завершение потоков - все эти функции должны быть использованы в тестах.

1. Реализовать примитив взаимного исключения потоков.

2. Реализовать планировщик потоков и организовать вытесняющую многозадачность. Т. е. другими словами, реализовать функцию, которая приостанавливает текущий поток исполнения и вместо него ставит на процессор другой поток исполнения (если таковой имеется), и вызвать эту функцию из обработчика прерывания таймера, после того как поток выработает свой квант времени.

3. Реализовать функции управления потоками. Конкретный интерфейс остается на ваше усмотрение, главное чтобы можно было создавать новые потоки, завершать потоки и дожидаться завершения других потоков.

Задания 2 и 3 очень тесно связаны друг с другом - действительно довольно бессмысленно создавать потоки, если вы не умеете передавать им управление. Поэтому, задание 3 без выполненного задания 2 не засчитывается. Обратное, впрочем, возможно, т. е. можно выполнить задание 2, не выполнив до конца задание 3.

2 Дополнительные задания

1. Реализация блокирующего примитива взаимного исключения (то, что обычно называют mutex-ом). В отличие от примитива взаимного исключения в основном задании, если mutex был захвачен, когда поток попытался выполнить lock на нем, этот поток

должен быть погружен в "сон" до тех пор, пока держатель mutex-а не выполнит на нем unlock[1].

2. Реализовать условную переменную (conditional variable, для примера смотрите pthread_cond_t) и соответствующие функции для работы с ней:

   - wait - ожидать, пока кто-нибудь не посигналит на условной переменной;
   - signal - посигналить одному из потоков ожидающих на условной переменной;
   - broadcast - посигналить всем потокам ожидающим на условной переменной;

Как и для основного задания, для дополнительного задания должны быть какие-нибудь тесты.

3 Примитив взаимного исключения

Примитив взаимного исключения потоков должен предоставлять как минимум функции lock и unlock, которыми можно оградить использование разделяемых ресурсов. Кроме очевидных гарантий взаимного исключения, требуется, чтобы вызовы lock/unlock могли быть вложенными, т. е. такое использование допустимо:

```
lock(lock_descriptor0)
    lock(lock_descriptor1)

    unlock(lock_descriptor1)
unlock(lock_descriptor0)
```

Поддерживать следующий вариант можно, но совершенно не обязательно:

```
lock(lock_descriptor0)
    lock(lock_descriptor1)

unlock(lock_descriptor0)
    unlock(lock_descriptor1)
```

Параметры[2] остаются на ваше усмотрение.

[1] Имена функций и названия структур остаются на ваше усмотрение, но семантика погружения потока в сон должна оставаться.

[2] Да и имена функции lock/unlock.

При реализации lock и unlock, настоятельно рекомендуется[1] помнить, что вы работаете с однопроцессорной системой, т. е. настоящего параллельного исполнения нет.

4 Функции управления потоками

От функций управления потоками ожидается как минимум следующая функциональность:

- создание потока и выделение всех необходимых ему ресурсов;
- завершение потока и освобождение его ресурсов;
- ожидание завершения некоторого потока;

4.1 Создание потока исполнения

Поток исполнения должен исполнять какой-то код, поэтому разумно, что в функцию создания потока исполнения будет передан указатель на функцию, с которой начнется исполнение потока - этакий main. Опционально[2] можно предусмотреть передачу некоторого параметра функции, которую будет исполнять поток.

Кроме того, так как потоков может быть несколько, полезно иметь возможность обращаться к потокам исполнения, т. е. иметь некоторый идентификатор потока исполнения, по которому его можно найти.

Таким образом функция создания потока может иметь примерно такую сигнатуру:

```
pid_t create_thread(void (*fptr)(void *), void *arg);
```

где

- pid_t - идентификатор созданного потока, или например, код ошибки, если не удалось создать поток
- fptr - главная функция потока (его main)
- arg - аргумент, который будет передан в fptr, когда поток начнет свое исполнение.

[1] Это очень-очень-очень упростит вам жизнь.

[2] И это довольно полезно.

С каждым потоком исполнения связан набор ресурсов, например, поток исполнения может иметь свою таблицу страниц[1], или некоторый дескриптор[2]. Набор ресурсов зависит от фантазии разработчика, но каждый поток исполнения обязан иметь свой собственный стек. Таким образом функция создания потока как минимум должна выделить для него стек.

Но самая содержательная часть этой функции заключается в том, что она должна инициализировать ресурсы потока таким образом, чтобы ему можно было передать управление, т. е. чтобы планировщик мог переключиться с другого потока на поток созданный этой функцией. Как именно этого добиться, естественно, зависит от того, как ваш планировщик переключает потоки.

Пример реализации переключения потоков вы можете найти в лекции, напомню, что для сохранения состояния потока используется стек этого потока. И, когда планировщик переключается на новый поток, он "восстанавливает" его состояние со стека. Но что если это совсем новый поток, который еще никогда не выполнялся? Планировщик, еще не снимал его с процессора и значит не сохранял его состояние на стек. Соответственно функция создания потока должна инициализировать стек потока таким образом, чтобы планировщик смог "восстановить" из него состояние вновь созданного потока.

4.2 Завершение потока исполнения и ожидание завершения

Потоки не работают бесконечно, поэтому вы должны предоставить функцию для завершения потока (аналог функции exit). Процессор, однако, ничего не знает о потоках - это абстракция нашей ОС, он просто исполняет инструкции. Отсюда следует, что когда поток завершается, на его место должен встать кто-то другой. Поэтому завершение потока предполагает, что произойдет переключение на какой-то другой поток исполнения, а ваша обязанность гарантировать, что такой поток исполнения всегда будет. Другими словами сложность этой части задания в том, что вы должны создать специальный поток, который выполняется, когда больше некому выполняться[3].

[1]Что в нашем случае не обязательно, и все потоки могут пользоваться общей таблицей страниц, или использовать свою таблицу, в которой есть отображение для общей памяти всех потоков.

[2]Структура, которая описывает состояние потока, его ресурсы и прочее.

[3]Такой поток обычно называют idle.

Другая сложность связанная с завершением потоков заключается в том, что после завершения потока необходимо освободить ресурсы, которые он занимает (как минимум стек). Более того, сделать это непосредственно в функции завершения потока нельзя[1]. Самый очевидный вариант - поручить освобождение ресурсов потока тому, кто занял его место, т. е. следующему потоку на процессоре.

Наконец, если поток может завершиться, значит кто-то может ждать, пока он завершится. Вам нужно предусмотреть возможность подождать завершения определенного потока[2].

5 Планирование и вытесняющая многозадачность

Вам не требуется придумывать какой-то особенный алгоритм планирования - хватит простейшего Round Robin. Т. е. давайте поработать всем потокам по кругу. Таким образом выполнение этой части задания сводится к реализации одной сравнительно простой функции, назовем ее schedule[3].

Эта функция выбирает следующий поток исполнения и переключается с текущего потока на выбранный поток исполнения. Переключение состоит из сохранения состояния текущего потока и восстановления состояния следующего потока. Пример переключения вы можете найти в лекции.

Если вы реализовали и проверили работу функции schedule, то для организации вытесняющей многозадачности вам остается только вызвать эту функции из обработчика прерывания таймера. Естественно, что вызывать ее на каждое прерывание слишком сурово, поэтому вам необходимо предусмотреть возможность отсчитывать время, которое поток занимает процессор и дать потоку поработать хотя бы несколько миллисекунд[4].

Учтите, что поток, на который вы переключитесь из обработчика таймера может ничего не знать об этом таймере, поэтому послать EOI контроллеру прерываний вы должны до того, как вызывать функцию schedule.

---

[1] Почему?

[2] Семантика схожая с pthread_join.

[3] Вы можете называть ее как вам угодно - но помните о чувстве юмора, точнее о его отсутствии.

[4] Если он, конечно, не захочет сам отдать процессор. Как он может это сделать?

6 Вопросы и ответы