

Задания по Операционным системам.

Второй семестр.

1. Задачи на работу с потоками.

Блок задач на “удовлетворительно”

1.1. Создание потоков посредством POSIX API:

- a. Склонировать репозиторий `git@github.com:mrutman/os.git`. Изучите и запустите программу `threads/thread.c`. Добейтесь чтобы гарантированно выполнялись оба потока.
- b. Измените программу, добавив создание 5-ти потоков с одинаковой поточной функцией.
- c. В поточной функции распечатайте:
 - i. идентификаторы процесса, родительского процесса, потока. Для получения идентификатора потока используйте функции `pthread_self()` и `gettid()`. Сравните с тем что функция `pthread_create()` вернула через первый аргумент. Объясните результат. Почему для сравнения идентификаторов POSIX потоков надо использовать функцию `pthread_equal()`?
 - ii. адреса локальной, локальной статической, локальной константной и глобальной переменных. Объясните результат.
- d. В поточной функции попробуйте изменить локальную и глобальную переменные. Видны ли изменения из других потоков? Объясните результат.
- e. Изучите `/proc/pid/maps` для полученного процесса. Найдите в нем стеки потоков.
- f. Запустите программу из-под `strace`. Найдите системные вызовы, которые создали ваши потоки.

1.2. Потоки Joinable and Detached.

- a. Напишите программу, в которой основной поток будет дожидаться завершения созданного потока.
- b. Измените программу так чтобы созданный поток возвращал число 42, а основной поток получал это число и распечатывал.
- c. Измените программу так чтобы созданный поток возвращал указатель на строку “hello world”, а основной поток получал этот указатель и распечатывал строку.
- d. Напишите программу, которая в бесконечном цикле будет создавать поток, с поточной функцией, которая выводит на свой идентификатор потока и завершается. Запустите. Объясните результат.
- e. Добавьте вызов `pthread_detach()` в поточную функцию. Объясните результат.
- f. Вместо вызова `pthread_detach()` передайте в `pthread_create()` аргументы, задающие тип потока - `DETACHED`. Запустите, убедитесь что поведение не изменилось.

1.3. Передача параметров в поточную функцию.

- a. Создайте структуру с полями типа `int` и `char*`. Создайте экземпляр, этой структуры и проинициализируйте. Создайте поток и передайте указатель на эту структуру в качестве параметра. В поточной функции распечатайте содержимое структуры.
- b. Измените программу так чтобы поток создавался как `detached` поток. Объясните в какой области памяти нужно располагать структуру в этом случае.

1.4 Прерывание потока.

- a. Напишите программу, в которой поточная функция в бесконечном цикле распечатывает строки. Используйте `pthread_cancel()` для того чтобы ее остановить.
- b. Измените программу так чтобы поточная функция ничего не распечатывала, а в бесконечном цикле увеличивала счетчик на 1. Используйте `pthread_cancel()` для того чтобы ее остановить. Объясните результат. Что можно сделать, чтобы `pthread_cancel()` прервал поток?
- c. В поточной функции выделите память под строку "hello world" с помощью `malloc()`. Распечатывайте в бесконечном цикле полученную строку. Используйте `pthread_cancel()` для того чтобы прервать поточную функцию. Добейтесь чтобы по завершению память, выделенная под строку освобождалась. Используйте `pthread_cleanup_push/pop()`.

1.5. Обработка сигналов в многопоточной программе.

- a. Напишите программу с тремя потоками, такими что: первый поток блокирует получения всех сигналов, второй принимает сигнал `SIGINT` при помощи обработчика сигнала, а третий - сигнал `SIGQUIT` при помощи функции `sigwait()`.
- b. Можно ли установить обработчики сигнала для каждого потока?

Блок задач на "хорошо"

1.6. Разработать собственную функцию для создания ядерных потоков - аналог `pthread_create()`:

```
int mythread_create(mythread_t thread, void *(start_routine), void *arg);
```

Функция должна возвращать успех-неуспех.

Блок задач на "отлично"

1.7. Разработать собственную функцию для создания пользовательских потоков:

```
int uthread_create(uthread_t thread, void *(start_routine), void *arg);
```

Функция должна возвращать успех-неуспех.

Допускается реализация без вытеснения потока.

2. Задачи на синхронизацию.

Блок задач на "удовлетворительно"

2.1. Проблема конкурентного доступа к разделяемому ресурсу.

- a. В каталоге `sync` репозитория `git@github.com:mrutman/os.git` вы найдете простую реализацию очереди на списке. Изучите код, соберите и запустите программу

queue-example.c. Посмотрите вывод программы и убедитесь что он соответствует вашему пониманию работы данной реализации очереди. Добавьте реализацию функции queue_destroy().

- b. Изучите код программы queue-threads.c и разберитесь что она делает. Соберите программу.
 - i. Запустите программу несколько раз. Если появляются ошибки выполнения, попытайтесь их объяснить и определить что именно вызывает ошибку. Какие именно ошибки вы наблюдали?
 - ii. Поиграйте следующими параметрами:
 - 1. размером очереди (задается в queue_init()). Запустите программу с размером очереди от 1000 до 1000000.
 - 2. привязкой к процессору (задается функцией set_cpu()). Привяжите потоки к одному процессору (ядру) и к разным.
 - 3. планированием потоков (функция sched_yield()). Попробуйте убрать эту функцию перед созданием второго потока.
 - 4. Объясните наблюдаемые результаты.

2.2. Синхронизация доступа к разделяемому ресурсу

- a. Измените реализацию очереди, добавив спинлок для синхронизации доступа к разделяемым данным.
- b. Убедитесь, что не возникает ошибок передачи данных через очередь.
- c. Поиграйте параметрами из пункта 2.1:
 - i. Оцените загрузку процессора.
 - ii. Оцените время проведенное в пользовательском режиме и в режиме ядра.
 - iii. Оцените текущую заполненность очереди, количество попыток чтения-записи и количество прочитанных-записанных данных.
 - iv. Объясните наблюдаемые результаты.
- d. Часто бывает, что поток пишущий данные в очередь вынужден ожидать их (например, из сети на select()/poll()). Проэмулируйте эту ситуацию, добавив в поточную функцию писателя периодический вызов usleep(1). Выполните задания из пункта c.
- e. Измените реализацию очереди, заменив спинлок на мутекс. Прodelайте задания из пунктов b, c и d. Сравните со спинлоком.
- f. Измените реализацию очереди, добавив условную переменную. Прodelайте задания из пунктов b, c и d. Сравните со спинлоком и мутексом.
- g. Используйте для синхронизации доступа к очереди семафоры. Прodelайте задания из пунктов b, c и d. Сравните со спинлоком, мутексом и условной переменной.

Блок задач на “хорошо”

TODO

Блок задач на “отлично”

TODO

3. Пользователи в Linux. Права и привилегии

Блок задач на “удовлетворительно”

TODO

Блок задач на “отлично”

TODO

4. Мини проекты.

Блок задач на “удовлетворительно”

TODO

Блок задач на “хорошо”

TODO

Блок задач на “отлично”

TODO