

Индивидуальное задание по курсу "Ядро ОС"

Манджиев Айта, 327 группа

25 декабря 2016 г.

Глава 1

Постановка задачи

В рамках решения индивидуального задания от Вас требуется реализовать потоки управления внутри процессов JOS.

Должны поддерживаться функции:

- `pthread_create()` [1];
- `pthread_join()` [2];
- `pthread_exit()` [3];
- `sched_setparam()` [4];
- `sched_setscheduler()` [5].

Должны поддерживаться политики планирования потоков `SCHED_FIFO` и `SCHED_RR` с приоритетами.

Также в рамках выполнения задачи необходимо будет разработать утилиту печати состояния потоков в процессе.

Глава 2

Описание решения

2.1 Реализация

Реализация функций работы с тредами и планировщиками ([1-5]) будет производиться как системные вызовы.

Для этого необходимо реализовать данные функции в `kern/syscall.c` и добавить их объявление в `inc/lib`, а также добавить их обработку в файле `kern/syscall.c` функции `syscall`. Также в `lib/syscall.c` необходимо добавить вызов функции `syscall` с соответствующим параметром, константы (которые передаются в системный вызов `syscall`: наподобии `SYS_yield`) нужно описать в `inc/syscall.h`.

При реализации планировщиков нужно будет изменить существующий `RR` планировщик в `kern/sched.c` (алгоритмы описаны здесь). Для `RR` планировщика будет использоваться прерывания по таймеру для процессов, которые исчерпали свой квант времени. Также необходимо будет реализовать список процессов соответствующего приоритета.

Все основные константы по работе с тредами будут описаны в файле `/inc/pthread.h` такие как: `PTHREAD_CREATE_JOINABLE`, `PTHREAD_CREATE_DETACHED`, `SCHED_RR`, `SCHED_FIFO`. Также там будет описана структура, которая будет содержать атрибуты тредов.

1. Функция [1] будет реализована путем вызова модифицированного `env_create`.
2. Функция [2] сначала проверяем является ли тред `JOINABLE`, если нет, то возвращаем ошибку. Если является, то проверяем завершился или нет, если да, то возвращаем результат и присаиваем треду `ENV_FREE` статус, если же нет, то добавляем текущий тред в глобальный список ожидания.
3. Функция [3] будет реализована путем сохранения результата в структуру `Env`, проверкой есть ли в глобальном списке треды, ожидающие завершения текущего треда и если таковые имеются возобновление их работы и запись

результата (адрес куда необходимо записывать результат будет храниться в глобальной структуре). и вызова `env_destroy`.

4. Функции [4, 5] будут реализованы путем работы со списком процессов в планировщике и замены соответствующих значений (приоритет и политика планирования) в их структурах `Env`. Для проверки можно будет печатать информацию о смене приоритета или политики планирования.

В Структуру `Struct Env` необходимо добавить:

- Флаг, указывающий чем является данный экземпляр (процесс или тред) — 0 — процесс, 1 — тред;
- Поле, отвечающее за количество тредов, порожденных процессом, так как нам нужно будет его проверять при создании треда, если мы хотим ограничить количество тредов на процесс (у тредов оно равно 0);
- Указатель на процесс, который породил данный тред(у процессов оно будет `NULL`).
- ID треда внутри своего процесса. (У процессов — 0)
- Поле, указывающее политику планирования процесса.
- Приоритет процесса.
- Поле, которое будет хранить тип треда (`JOINABLE`, `DETACHED`, `JOINABLE_FINISHED`)
- Поле типа `void*`, которое будет хранить результат отработки треда

В `kern/env.c`:

1. `env_alloc` необходимо изменить и учесть создается ли новый процесс или новый тред:
 - Настраивать виртуальную память для треда или для процесса `env_setup_vm`;
 - Учесть изменение `esp` для треда (разделим весь стек процесса на равные части, например — 6, если максимальное количество тредов — 5);
 - Изменить информационную печать (`env or thread created`).
2. `env_setup_vm`: необходимо реализовать функцию для тредов аналогичную функции `env_setup_vm`, разница в том, что мы не выделяем новую физическую страницу для директории страниц, а используем ту, которая используется в отце.

3. `env_create`:

- При создании треда не загружаем ELF образ;
- Если создается тред, то записываем `arg`, который был передан в `pthread_create`, в стек треда после этого кладется адрес возврата, который будет указывать на вход в `pthread_exit`
- Устанавливаем `env` флаг на начало функции треда.

4. `env_free`:

- Печать удаления процесса \треда;
- Если данный экземпляр является тредом, то не производить действия по удалению процесса.
- Если данный экземпляр является тредом, то проверить нет ли ожидающих его тредов в глобальном списке.
- Если экземпляр является процессом, то удалить все треды, которые он создал.
- Если данный экземпляр является тредом, то уменьшить счетчик тредов в процессе –отце.
- Если данный тред является `JOINABLE`, тогда статусу треда присвоить константу `ENV_FINISHED`, а если `DETACHED`, то присвоить статус `ENV_FREE`

5. `env_run`: Не перезагружать `cr3` регистр другой директорией страниц, если идет смена тредов внутри одного процесса.

2.2 Тесты

1. Простой тест на работоспособность тредов, например, создание у процесса нескольких тредов и печать из этих тредов проверочной информации.
2. Тест на работоспособность `join`: Один из тредов вычисляет какое-либо значение, а второй тред ожидает результата вычисления. После этого ожидающий тред печатает результат, который вычислил первый.
3. Тест на работу планировщиков: основной процесс запускает еще один с помощью `spawn`; оба процесса порождают несколько тредов с разными приоритетами, которые в свою очередь печатают информацию о себе: приоритет и политику планирования (для этого нужно будет добавить еще один системный вызов для получения информации). Так можно будет проверить очередность их выполнения.

4. Один из тредов завершает свою работу, второй пытается сделать `join`, тем самым получая результат треда, который уже завершился.
5. Завершение треда с помощью `pthread_exit` и по `return`.
6. Создание тредов с разными атрибутами и печати их состояния.
7. Тест на проверку отклоняется ли `join` в случае, если мы пытаемся сделать его к `DETACHED` треду.

2.3 Печать информации о тредах

Печать информации о тредах будет реализован в качестве команды в `monitor.c`