Вторая работа
Планирование процессов

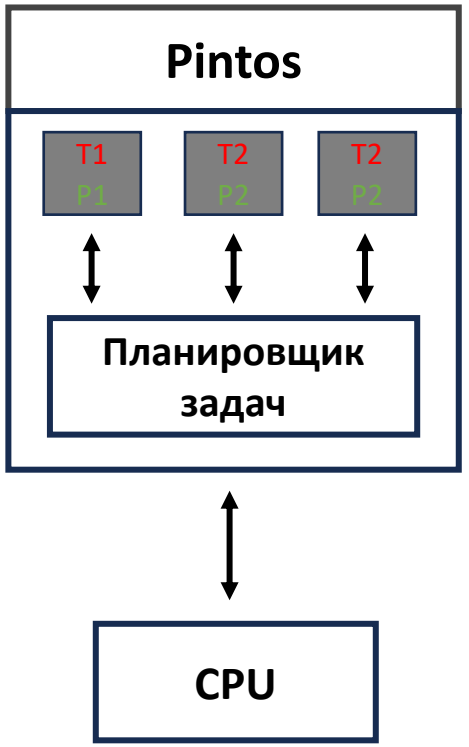
Механизм планирования

Система планирования

- Система планирования в ОС Pintos является частью системы управления процессами и отвечает за распределение разделяемых ресурсов, таких как процессорное время

```
struct thread
{
      tid_t tid;
      enum thread_status status;
      char name[16];
      uint8_t *stack;
      int priority;
      struct list_elem allelem;
      struct list_elem elem;

      #ifdef USERPROG
          uint32_t *pagedir;
      #endif
      unsigned magic;
};
```



Базовая система планирования

- В каждый момент времени в ядре ОС Pintos может выполняться **только один процесс**
- Остальные, если они существуют, становится **неактивными**

- Планировщик принимает решение, какому процессу выделить процессор
- Если готовых к выполнению процессов нет, то планировщик выбирает **специальный "пустой" процесс**, который релизован в функции idle()

- **Примитивы синхронизации** могут вносить изменения в **порядок выполнения процессов**
- Например, когда один процесс ждет результатов выполнения другого процесса или освобождения некоторых ресурсов

Функция планирования

```
static void schedule (void)
{
    struct thread *cur = running_thread ();
    struct thread *next = next_thread_to_run ();
    struct thread *prev = NULL;

    if (cur != next)
        prev = switch_threads (cur, next);
    thread_schedule_tail (prev);
}
```

Функция *schedule()* отвечает за планирование процессов

1. Записывает текущий процесс в локальную переменную cur
2. Определяет процесс, который будет выполняться следующим в локальной переменной *next*
3. Вызывает функцию *switch_threads()* чтобы осуществить переключение контекста

Остальная часть планировщика реализована в функции *thread_schedule_tail(),* которая **помечает новый процесс как выполняющийся**

Алгоритмы планирования

- ***First-Come, First-Served (FCFS)*** – выбранный процесс использует процессор до завершения своей работы. После этого для выполнения выбирается новый процесс из начала очереди.

- ***Round Robin (RR)*** –каждый процесс исполняется фиксированный квант времени, на время которого он получает процессор в свое распоряжение, после чего происходит вытеснение процесса

- ***Shortest-Job-First (SJF)*** – для исполнения выбирается процесс с минимальной длительностью исполнения среди находящихся в состоянии «готовность»

Задача1: Приоритетное планирование процессов

1. Подготовительная часть – разобраться в принципах работах планировщика
   - Как переключаются процессы
   - Как выбирается следующий процесс
   - Как меняется статус процессов
   - …

2. Реализовать приоритетное планирование
   - Переключение только при наличии процесса с большим приоритетом
   - Процессы с одинаковым приоритетом планируются по RR
   - Учитывать приоритет при входе в критическую секцию

Примитивы синхронизации

Синхронизация

- Используются для **координации доступа к общим ресурсам** между несколькими потоками или процессами
- Служат для предотвращения проблем, связанных с одновременным доступом, таких как гонка данных (race condition)

- Надлежащая синхронизация является важнейшим компонентом любой системы управления процессами
- В общем случае, любая проблема сихронизации процессов **может быть решена отключением системных прерываний**

Примитивы синхронизации

Семафор

- Поддерживают счетчик, который указывает, сколько потоков может получить доступ к ресурсу
- Когда поток хочет получить доступ, он уменьшает счетчик
- Когда поток освобождает ресурс, он увеличивает счетчик
- Если счетчик достигает нуля, все другие потоки будут блокированы

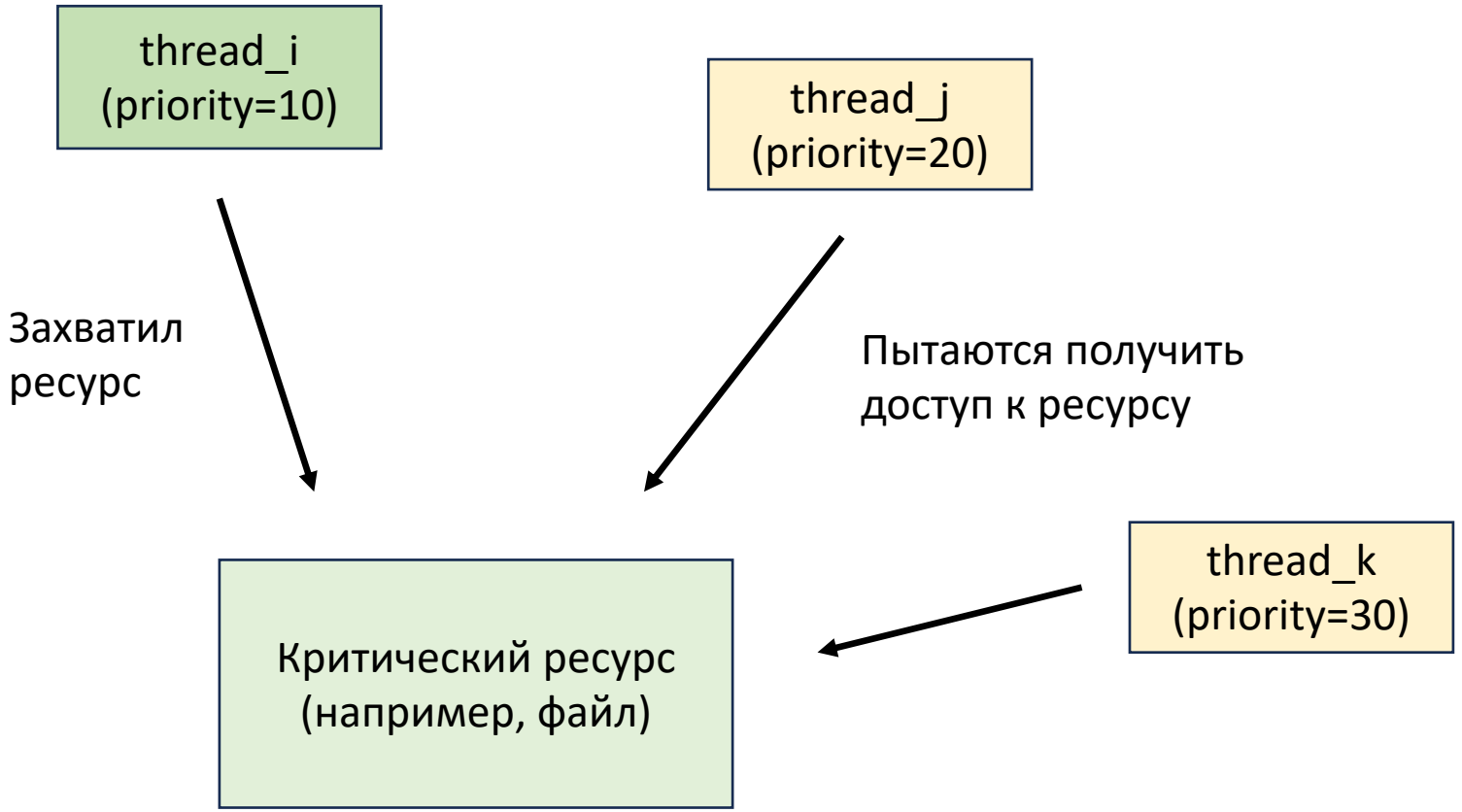
Замки (lock)

- Аналог семафора с заданным значением value, равным 1
- Гарантирует, что только один поток или процесс имеет доступ к определенному ресурсу или секции кода в данный момент времени
- Если другой поток пытается получить доступ, он будет блокирован до тех пор, пока первый поток не освободит мьютекс

Монитор

- Это логическая конструкция, представляющая собой структуру, которая содержит закрытые переменные (разделяемые ресурсы) и функции для работы с данными переменными
- Мониторы решают проблему синхронизации, используя замки и переменные-условия

Доступ к критическим ресурсам



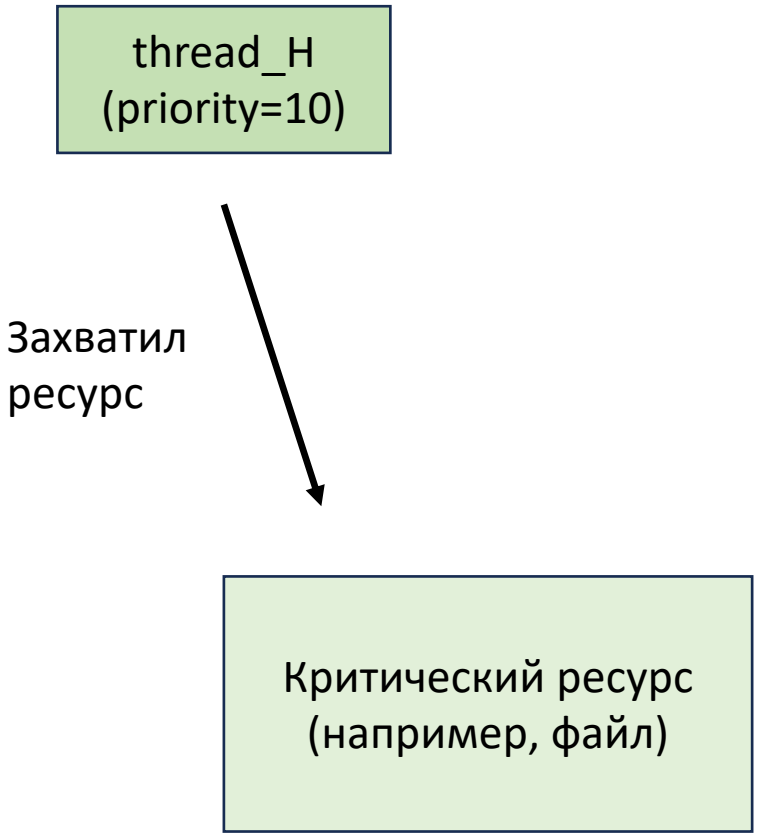
Решение - необходимо добавить поддержку приоритетов в механизмы планирования

Задача2: Поддержка приоритетов механизмами синхронизации

1. Подготовительная часть – разобраться в принципах механизмов синхронизации
   - Где реализованы
   - Как исполняются
   - Как реализованы
   - …

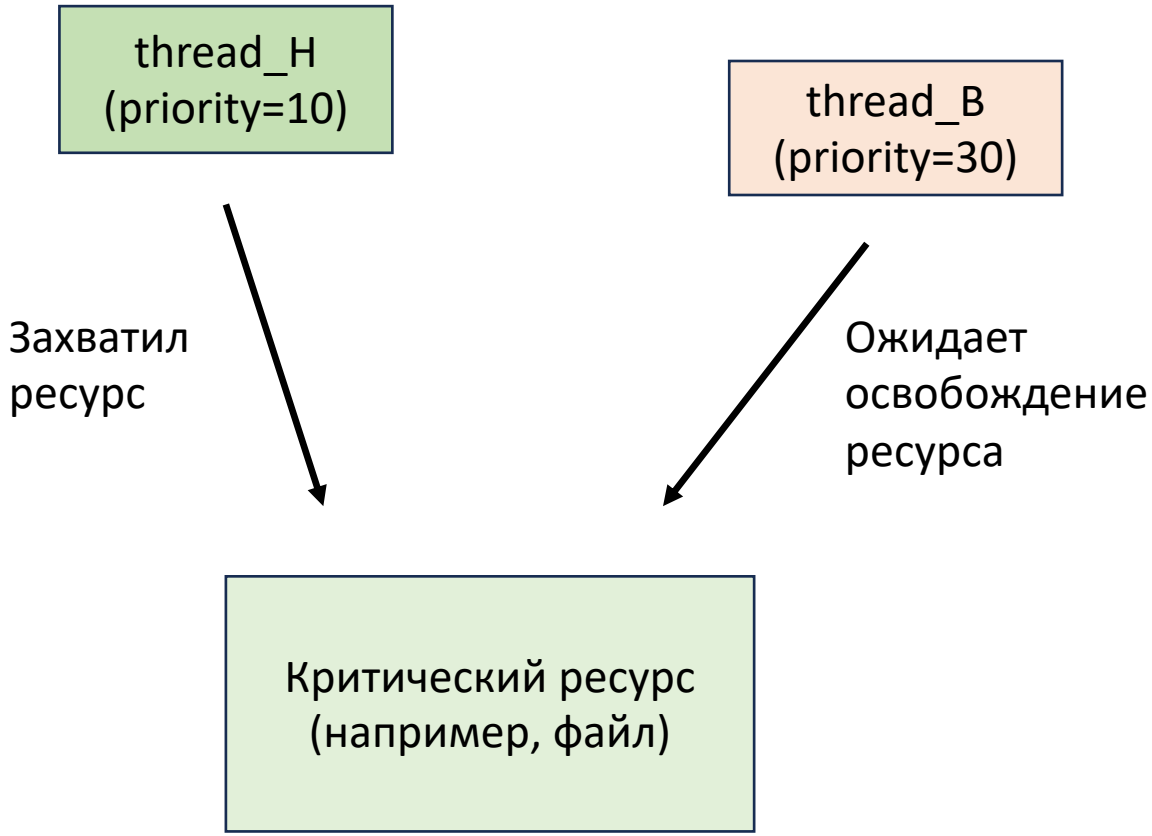
2. Внести изменения в работу примитивов синхронизации путем учета приоритетов процессов

Механизм разделения приоритета



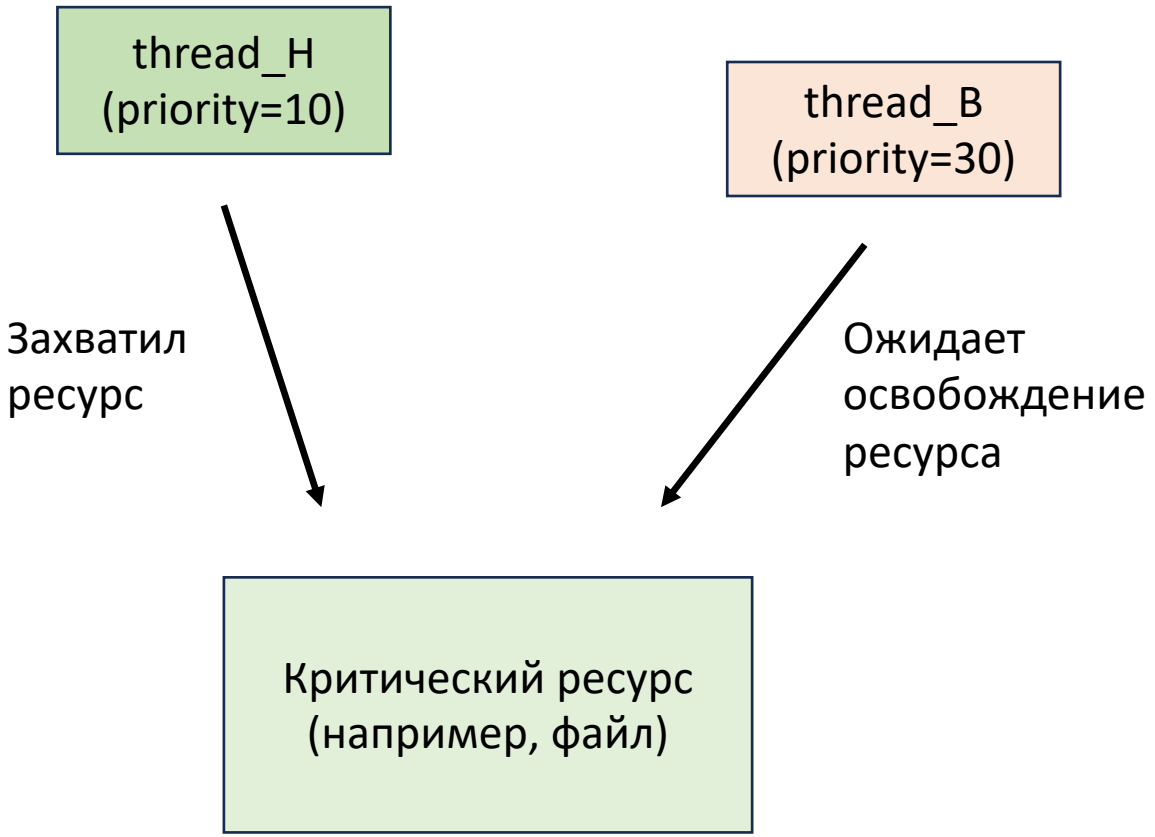
Выполняется процесс *thread_H*

Механизм разделения приоритета



Выполняется процесс *thread_H*
Процесс *thread_B* заблокирован

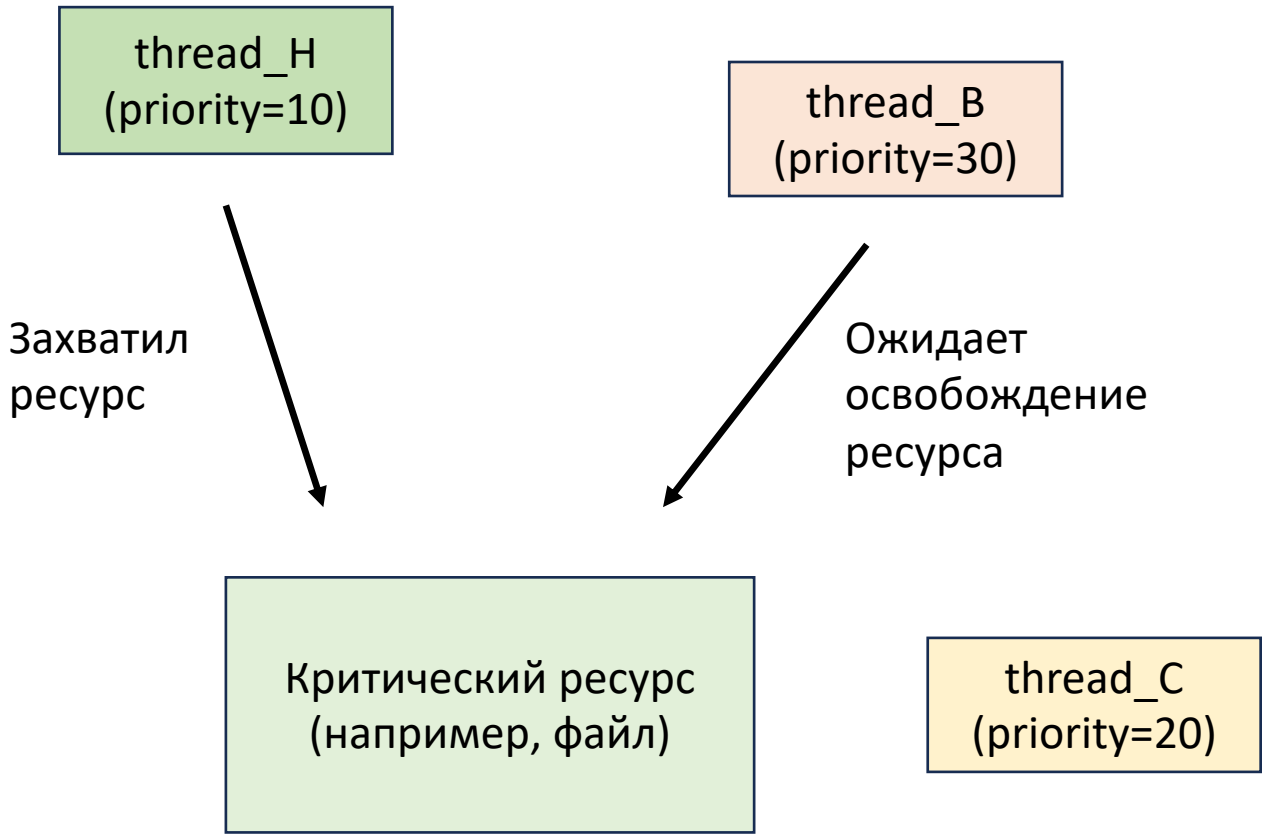
Механизм разделения приоритета



Процесс *thread_H* остановлен
Процесс *thread_B* заблокирован
Процесс *thread_C* исполняется

Процесс *thread_H* никогда не получит процессорного времени

Механизм разделения приоритета



thread_H ← *thread_B*

thread_H начинает исполняться
thread_H освобождает ресурс

thread_H ← исходный приоритет

thread_B начинает исполняться
thread_B захватит ресурс

Задача3: Реализация механизма жертвования

- Когда процесс пытается захватить ресурс
  - Проанализировать ситуацию
  - Пожертвовать приоритет всем нуждающимся процессам

- Когда процесс освобождает ресурс
  - Вернуть исходный приоритет