Лабораторная работа 1
ВВЕДЕНИЕ. СИСТЕМНЫЙ ТАЙМЕР

Цель работы – изучение системы управления процессами, а также механизма работы системного таймера в ОС Pintos, анализ его недостатков и модификация его алгоритма.

Основная рабочая директория: devices/timer.c

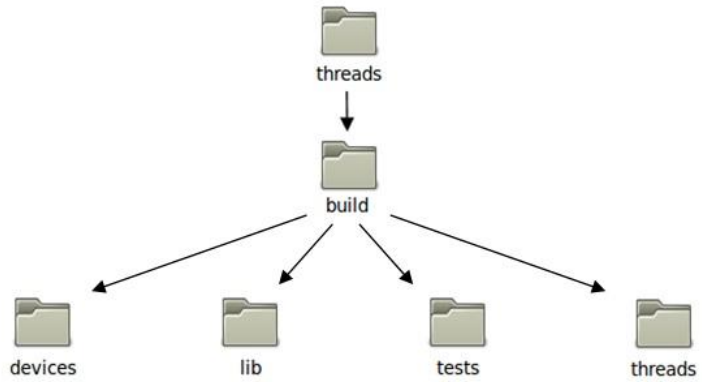
Теоретические сведения

Pintos – это учебная операционная система, которая была разработана Стэнфордским университетом для курса «Принципы построения операционных систем» в 2004 году. Она работает в рамках архитектуры 80x86 и в примитивной форме реализует основные компоненты реальных операционных систем, такие как процессы ядра, загрузка и запуск пользовательских программ, файловая система и виртуальная память. Студентам предлагается самостоятельно развить данную операционную систему, последовательно добавляя в нее различные компоненты. Основная работа будет осуществляться в таких направлениях, как обеспечение синхронизации процессов, внедрение механизмов виртуальной памяти иорганизация файловой системы. Студентам также предстоит ознакомиться со способами загрузки пользовательских программ и модернизировать представленный механизм. Вся функциональность, разработанная на каждом из этапов, будет востребована в следующих работах.

Приступая к выполнению той или иной модификации учебной операционной системы Pintos, необходимо изучить все исходные коды, относящиеся к данной задаче. Исходные коды ОС Pintos полностью открыты и написаны на языке программирования C с подробными комментариями. Названия файлов исходных кодов соответствуют расположенным внутри функциями операционной системы. Подробнее о работе с ОС Pintos см. в Приложении 2.

Рабочей директорией данной лабораторной работы будет каталог src/threads, в котором расположены исходные коды ядра системы. Если сборка системы из исходных кодов не была осуществлена ранее, запустите команду make, чтобы подготовить каталог к работе. В результате выполнения сборки будет создан подкаталог build, который содержит все необходимое для работы: наборы библиотек и тестов (см. Приложение 3), а также объектные файлы основных компонентов системы. Структура каталога threads после сборки представлена на рис. 1.1.

Рисунок 1.1. Каталог threads



Данная лабораторная работа является введением и ставит своей задачей ознакомить с механизмом работы ядра операционной системы. Основной целью является усовершенствование алгоритма

работы системного таймера, который уже существует в системе, но реализация которого не является оптимальной. Перед тем, как приступить к выполнению данной лабораторной работы, обратите особое внимание на исходные коды ядра и системы управления процессами ядра, расположенные в каталоге threads (табл. 1.1).

Таблица 1.1. Исходные коды директории src/threads

Название файла:	**Описание:**
loader.S loader.h start.S kernel.lds.S	Загрузчик операционной системы, коды первичной инициализации ядра операционной системы. Написаны на языке ассемблера с подробными комментариями.
init.c init.h	Инициализация ядра операционной системы. Особый интерес представляет функция main() в init.c , которая дает представление о том, что и в каком порядке инициализируется при запуске системы.
thread.c thread.h	Базовая система управления процессами. В thread.h определена основная структура процесса struct thread, к которой часто придется обращаться при выполнении модификаций. В thread.среализованы основные функции работы с процессами.
interrupt.c interrupt.h	Базовая реализация обработки прерываний, содержит функции, включающие и отключающие прерывания. Исходные коды intr-stubs.S/intr-stubs.h содержат реализацию прерываний на уровне оборудования.

Базовая реализация системы управления процессами в ОС Pintos содержит необходимый минимум функций, которые реализуют создание, облуживание и планирование процессов в ядре системы. Каждый процесс можно рассматривать как мини-программу, запущенную в рамках ОС Pintos.

В табл. 1.2 рассмотрены основные функции работы с процессами, расположенные в threads/thread.c.

Таблица 1.2. Функции работы с процессами в threads/thread.c

Функция:	**Назначение:**
void thread_init (void)	Функция вызывается из main() для инициализации системы управления процессами, под которой понимается совокупность процессов и функций работы с ними. Ее основной задачей является создание главного процесса ядра ОС Pintos, который по своей архитектуре ничем не отличается от любого другого процесса системы.
void thread_start (void)	Функция вызывается из main() и запускает механизм планирования процессов. Создает «пустой» процесс, который выбирается планировщиком,когда в очереди на выполнение нет готовых процессов.
void thread_tick (void)	Функция вызывается системным таймером. Отсчитывает установленный квант и по его истечении передает управление планировщику.
tid_t thread_create (const char *name, int priority, thread_func *func, void *aux)	Функция создает и запускает новый процесс уровня ядра с именем name, заданным приоритетом *priority*, добавляет его в очередь готовых к выполнению и возвращает идентификатор TID созданного процесса или TID_ERROR, если создание процесса закончилось неудачей.
void thread_func (void *aux)	Функция, в которой задается задача, которая будет выполняться при создании нового процесса. thread_func является аргументом функции thread_create.
void thread_block (void)	Функция переводит исполняющийся процесс из состояния THREAD_RUNNING в состояние THREAD_BLOCKED. Процесс не сможет выполняться до тех пор, пока для него не будет выполнена функция thread_unblock(). Вместо использования данной низкоуровневой функции

	рекомендуется применять примитивы сихронизации.
void thread_unblock (struct thread *thread)	Функция переводит заблокированный процесс (THREAD_BLOCKED) в состояние готовности (THREAD_READY).
tid_t thread_tid (void)	Функция возвращает уникальный идентификатор исполняющегося в данный момент процесса. Аналогична операции thread_current ()->tid.
const char *thread_name (void)	Функция возвращает имя исполняющегося процесса. Аналогична операции thread_current ()->name.
void thread_exit (void) NO_RETURN	Функция завершает исполняющийся процесс. Об использовании макроса NO_RETURN см. Приложение 2.
void thread_yield (void)	Функция передает управление планировщику, который выбирает новый процесс на исполнение.

Основной структурой данных ОС Pintos, предназначенной для работы с процессами, является структура struct thread, которая определена в threads/thread.h:

```
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;
};
```

Поля данной структуры подробнее рассмотрены в табл. 1.3.

Таблица 1.3. Поля структуры struct thread

Поле:	**Назначение:**
tid_t tid	Так как в рамках ядра ОС Pintos "процесс" и "поток" являются синонимами, то аналогичным образом TID (ThreadIdentifier) является синонимом PID (Process Identifier) и представляет собой уникальный идентификатор, который присваивается процессупри его создании. TID является числом в формате int.
enum thread_status status	Статус процесса, который может быть одним из следующих: THREAD_RUNNING - процесс исполняется. В каждый момент времени в системе может исполняться только один процесс. THREAD_READY - процесс готов к выполнению, но не выполняется в данный момент. Он может быть выбран к выполнению планировщиком. THREAD_BLOCKED - процесс заблокирован. Планировщик не сможет обратиться к этом процессу, пока он не сменит состояние на THREAD_READY вызовом функции thread_unblock() или получитразрешение от примитивов сихронизации. THREAD_DYING - процесс будет уничтожен планировщиком после переключения на другой процесс

char name[16]	Имя процесса
uint8_t *stack	Каждый процесс имеет стек для того, чтобы система имела возможность отслеживать его состояние. Когда процессор переключается с одного процесса на другой, в этом поле сохраняется указатель вершины стека
int priority	Значение приоритета процесса, которое может быть установлено в рамках от PRI_MIN (0) до PRI_MAX (63), где 0 - это минимальынй приоритет процесса, 31 – значение приоритета по умолчанию. Изначально операционная система игнорирует значение приоритета процесса. Этотнедостаток нужно будет исправить впоследствии при разработке планировщика.
struct list_elem allelem	Это поле необходимо, чтобы связать процесс со списком всех процессов. Каждый процесс заносится в этот список при создании и удаляется из него при своем завершении.
struct list_elem elem	Используется для того, чтобы поместить процесс в очередь готовых к выполнению или ожидающих входа в критическую секцию процессов. Структура также описана в synch.c
unsigned magic	Устанавливает значение THREAD_MAGIC, которое служит для предупреждения переполнения стека процесса. Оно проверяется в функции thread_current(), пока выполняется процесс. Переполнение стека обычно искажает это число

Исходные коды компонентов системы, взаимодействующих с аппаратным обеспечением ввода-вывода, расположены в директории src/devices. В табл. 1.4. представлены описания некоторых из этих файлов.

Таблица 1.4. Исходные коды директории src/devices

Название файла:	**Описание:**
kbd.c kbd.h	Драйвер клавиатуры
ide.c ide.h	Поддержка чтения и записи секторов IDE дисков
timer.c timer.h	Системный таймер, который по умолчанию отсчитывает 100 тактов в секунду
vga.c vga.h	Драйвер VGA, который отвечает за вывод текста на экран
partition.c partition.h	Анализаторразбиения диска на разделы

Системный таймер является частью системы управления процессами. Функция timer_sleep(), реализованная в devices/timer.c, приостанавливает выполнение текущего процесса на заданное число тактов таймера *ticks*, в цикле while() проверяя текущее значение ticks и вызывая функцию thread_yield():

```
void timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();
  ASSERT (intr_get_level () == INTR_ON);

  /*АКТИВНОЕ ОЖИДАНИЕ*/
  while (timer_elapsed (start) < ticks)

    thread_yield ();
}
```

Базовая (т.е. предоставленная разработчиками системы) реализация системного таймера содержит в себе цикл активного ожидания. *Активное ожидание* – это состояние процесса, при котором он многократно проверяет истинность некоторого условия, например, такого как доступность некоторых ресурсов или состояние других процессов. Очевидно, что такая реализация не только не выполняет никакой полезной работы, но и расходует ресурсы системы: использование активного ожидания приводит к бесполезному расходованию процессорного времени и, как следствие, снижению общей производительности системы. Кроме того, при некоторых стратегиях планирования времени ЦП в целом корректный алгоритм с активным ожиданием может привести к тупику.

Примером может служить стратегия планирования с приоритетами, когда процесс, имеющий больший приоритет, загружается на выполнение и переходит в состояние активного ожидания, в то время как процесс с меньшим приоритетом захватил необходимый ресурс, но был выгружен до того, как освободил его. Поскольку процесс с большим приоритетом не блокирован, а готов к продолжению выполнения, переключение процессов не происходит, и процесс, владеющий ресурсом, никогда не сможет его освободить. Активное ожидание используется только тогда, когда есть уверенность, что время ожидания будет небольшим. Блокировка, использующая активное ожидание называется *спин-блокировкой*.

Порядок выполнения работы

1. Изучить работу системного таймера. Для этого необходимо ознакомиться с исходными кодами таймера, представленными в devices/timer.h и devices/timer.c, а также с функциями работы с процессами, которые описаны в threads/thread.h и threads/thread.c.
2. Составить таблицу функций системного таймера (см. описания функций в devices/timer.h) с описанием аргументов, которые принимают эти функции, и действий, которые эти функции выполняют.
3. Изобразить блок-схему текущего алгоритма работы функции timer_sleep() системного таймера в ОС Pintos.
4. Разработать новый алгоритм работы системного таймера, который позволит избежать активного ожидания. Алгоритм должен отвечать следующим требованиям:
   a) Если система находится в режиме ожидания (нет других запущенных потоков), то вызвавший функцию timer_sleep() процесс должен быть разбужен точно по истечении заданного времени;
   b) Процесс не должен мешать другим процессам, которые исполняются в момент его пробуждения. Для решения этой проблемы рекомендуется создать очереди готовых к исполнению и заблокированных таймером процессов, а также использовать функции, которые переводят процессы из состояния в состояние.
   c) Процессы, находящиеся в такой очереди должны храниться в отсортированном виде. Очередь должна быть реализована в виде отсортированного динамического списка или массива.
   d) Значение аргумента функции timer_sleep() измеряется в единицах таймера, не в миллисекундах. Значение по умолчанию определено в макросе TIMER_FREQ (devices/timer.h)

и равно 100 единицам в секунду. Не рекомендуется изменять это значение, так же как и вносить изменения в функции timer_msleep(), timer_usleep() и timer_nsleep();

5. Реализовать разработанное решение. Для этого нужно внести соответствующие изменения в исходные коды devices/timer.c и, если это потребуется, в devices/timer.h, threads/thread.h и threads/thread.c. Если для программной реализации решения требуется добавить новый заголовочный файл или файл с исходным кодом на языке программирования C, необходимо поместить его в нужную директорию и модифицировать Makefile, соответствующий этой директории, указав в нем имя нового файла.
6. Изобразить блок-схему разработанного алгоритма.
7. Оценить реализованное решение, используя встроенную систему тестирования (см. Приложение 3). Должны быть успешными следующие тесты:
   - alarm-single
   - alarm-multiple
   - alarm-simultaneous
   - alarm-zero
   - alarm-negative
8. Выполнить дополнительные индивидуальные задания, выданные преподавателем.

Содержание отчета

В отчете необходимо указать следующие данные:

1. Таблица функций системного таймера.
2. Блок-схемы алгоритма работы системного таймера до и после модификации.
3. Описание внесенных модификаций в код ОС pintos с подробными комментариями.
4. Выводы тестов, полученные при их запуске.
5. Результаты выполнения дополнительных индивидуальных заданий.