Операционная система Pintos

I. О системе

Pintos — это учебная операционная система, работающая на вычислительных машинах с архитектурой Intel 80x86. Т.к. ОС Pintos была разработана исключительно для учебных целей, ее функциональные возможности сильно ограничены. В Pintos реализованы только базовые службы, такие как загрузчик, ядро, простая файловая система. Студентам предлагается самостоятельно развить данную операционную систему, последовательно добавляя в нее различные компоненты и модифицируя существующие. Основная работа будет осуществляться в таких направлениях, как обеспечение синхронизации. Студентам также предстоит ознакомиться со способами загрузки пользовательских программ и модернизировать представленный механизм.

Некоторые особенности ОС Pintos:

- Операционная система полностью написана на языке С. Исключение составляют ассемблерные вставки в реализациях загрузчика и ядра системы.
- Интерактивная работа в Pintos не предусмотрена. Пользователь обращается к системе через терминал UNIX с помощью предоставленной разработчиком ОС утилиты, запускающей Pintos в эмуляторе. При обращении к ОС в первую очередь создается конфигурационный файл эмулятора, необходимый для его запуска. Эмулятор открывается в новом окне и выводит ряд сообщений BIOS, после чего загружает и запускает Pintos.
- Pintos не поддерживает операции с числами вещественного типа. Использование типа float и double в коде небезопасно и приводит к системным ошибкам.
- В системе не реализована поддержка пользовательских многопоточных программ, поэтому термины «поток» и «процесс» в Pintos являются синонимами.
- Среда для сборки и запуска Pintos работает на базе операционной системы UNIX/Linux с использованием программ-эмуляторов. Эмуляторы являются внешним программным обеспечением и не предоставлены вместе с исходными кодами системы. Разработчиком рекомендованы такие эмуляторы, как Bochs и QEMU, при выполнении работ в учебных аудиториях рекомендован эмулятор QEMU.

Все исходные коды Pintos содержатся в архиве pintos.tar.gz. Там же находятся средства тестирования работоспособности написанных студентами программ. Каталог pintos/src рассмотрен более подробно в табл. 1.

Таблица 1. Содержимое каталога pintos/src

Каталог:	**Содержимое:**
threads/	Исходные коды ядрасистемы. Рабочая директория лабораторных работ 1, 2 и 3.
userprog/	Исходные коды загрузчика пользовательских программ. Рабочая директория лабораторных работ 4, 5.
vm/	Пустая директория, предназначена для разработки механизмов виртуальной памяти. В лабораторном практикуме не используется.
filesys/	Исходные коды файловой системы
devices/	Исходные коды программ, взаимодействующих с аппаратным обеспечением ввода-вывода (клавиатура, мышь и др. оборудование).

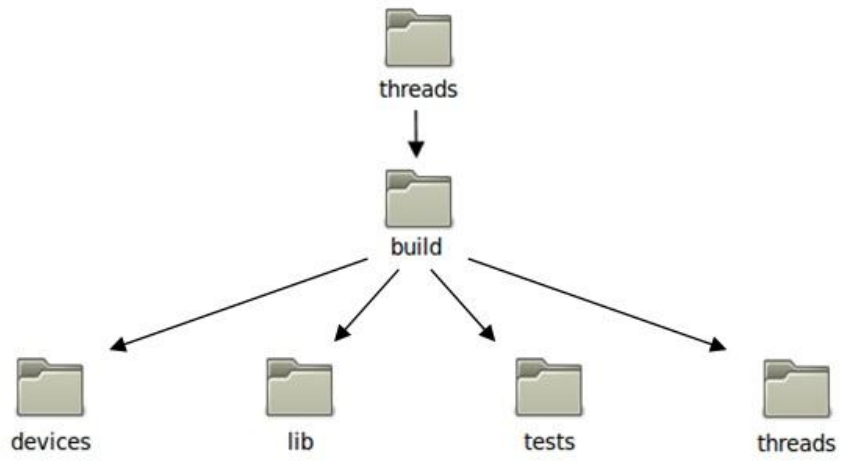
lib/	Стандартные библиотеки языка C, которые используются в коде Pintos.
lib/kernel	Библиотеки языка C, предназначенные для работы с ядром Pintos. В этом каталоге также расположены реализации некоторых структур данных, которые могут быть полезны при разработках: битовые массивы, линейные списки и хэш-таблицы
lib/user	Библиотеки языка C, предназначенные для работы с пользовательскими программами ОС Pintos
tests/	Предоставленные разработчиками системы тесты. Разрешено дополнение данной директории собственными тестами с целью оценить качество разработанных программ. Оценка результатов будет производиться по оригинальным тестам.
examples/	Примеры пользовательских программ
misc/ utils/	Программы и скрипты, предназначенные для настройки среды сборки и запуска Pintos.

Распаковать архива ОС pintos можно с помощью команды:

```
tar –zxf pintos.tar.gz
```

В результате будет создан каталог pintos/src, содержащий указанные подкаталоги. Для сборки ОС Pintos собирается из исходных кодов с использованием применяется команда `make`, которую необходимо выполнить в каталоге threads или userprog в зависимости от лабораторной работы. Команда компилирует ядро и тесты в один исполняемый бинарный модуль. После успешного завершения сборки в каталоге появляется подкаталог build, которая содержит все бинарные файлы, объектные файлы тестов и скомпилированные библиотеки. Именно этот подкаталог build является рабочим при выполнении обращения к тестам.

Рисунок 1. Структура рабочей директории threads после сборки



II. Работа с ОС Pintos

1. Командная строка

Командная строка для запуска ОС предоставлена одноименная утилитой pintos (размещается в /usr/bin при установке) имеет следующий вид:

```
pintos [options] -- [arguments]
```

Для получении подробного описания всех опций необходимо запустить команду pintos без аргументов. Описание основных опций приведено в табл. 2.

Таблица 2. Опции запуска ОС pintos

Опция:	**Назначение:**
`--qemu`	Использовать для запуска эмулятор QEMU.
`--gdb`	Запустить эмулятор в режиме отладки.
`-T [N]`	Завершить выполнение эмулятора после N секунд.
`-m [M]`	Предоставить запускаемой в эмуляторе ОС pintos M мегабайт памяти (по умолчанию — 4)
`--put-file=HOSTFN`	Поместить файл HOSTFN на виртуальный диск.
`--get-file=GUESTFN`	Считать файл GUESTFN с виртуального диска.
`--disk=DISK`	Использовать указанный файл в качестве виртуального диска (не удалять после завершения).
`--make-disk=DISK`	Использовать указанный файл в качестве виртуального диска (удалить после завершения).

Аргументы, передаваемые запускаемому ядру ОС pintos, указываются в команде после разделителя -- (два дефиса). Набор поддерживаемых аргументов зависит от ядра pintos. В табл. 3 приведено описание поддерживаемых аргументов по умолчанию для минимального ядра pintos, собранного в каталоге threads.

Таблица 3. Аргументы запуска ОС pintos

Опция:	**Назначение:**
`-h`	Вывести перечень поддерживаемых аргументов и их описание.
`-q`	Выключить виртуальный компьютер (завершить работу) после выполнения задания.
`-r`	Перезагрузить виртуальный компьютер после выполнения задания.
`-j SEED`	Указать начальное значения для системного генератора псевдослучайных числе.
`run TEST`	Запустить тест с именем TEST

Задание лабораторной работы или индивидуального дополнительного задания может требовать расширить набор поддерживаемых аргументов запуска ядра.

Пример запуска ядра pintos на эмуляторе QEMU:

```
pintos --qemu -- run alarm-single
```

Данная команда создаст виртуальный диск в виде файла, разместит на диске загрузчик и бинарный образ ядра, запустит эмулятор QEMU с созданным виртуальным диском. Запускаемому ядру pintos будет передано имя теста для запуска (run alarm-single). После исполнения команды эмулятор с загруженной ОС pintos продолжит работу.

Для запуска pintos с целью анализа выполнения полученного вывода следует дополнить вызов аргументом ядра -q:

```
pintos --qemu -- -q run alarm-single
```

После выполнения pintos эмулятор завершается и на экран выводится результат выполнения.

2. Отладка программ

2.1. Отладочные сообщения

Наиболее простым способом анализа работы системы и поиска ошибок при выполнении программ является печать отладочных сообщений. Для этого в pintos реализована функция языка Си printf(), которая обладает высоким приоритетом и может быть вызвана из любой функции. Стратегия отладки с использованием отладочных сообщений очень проста: выявление ошибок осуществляется в процессе наблюдения за выводом сообщений и сопоставления полученного вывода с запланированным при размении вызовов printf().

2.2. Использование макросов

Использование при написании кода макросов позволит отловить ошибки еще до того, как они повлекут за собой негативные последствия. Например, они могут быть использованы для проверки переданных в функцию аргументов:

```
ASSERT(arg_value != NULL)
```

Макрос ASSERT определен в lib/debug.h и имеет следующий синтаксис:

```
ASSERT(логическое выражение)
```

Логическое выражение внутри скобок может принимать значение *истина* или *ложь*. Если оно принимает значение *ложь*, то выполнение кода будет остановлено. Сообщение об ошибке будет содержать само логическое выражение и данные (имя функции и номер строки, в которой расположен данный макрос) в виде трассировки, которые помогут определить местонахождение и причину ошибки.

Pintos также предлагает разработчикам использовать приведенные в табл. 4 макросы (они также определены в lib/debug.h)

Таблица 4. Макросы Pintos

Макрос:	**Применение:**
`UNUSED`	Позволяет сообщить компилятору, что данный параметр может быть не использован внутри функции. Это позволяет отказаться от ряда предупреждений при компиляции. Пример: `void func(void* param UNUSED);`
`NO_RETURN`	Используется при создании прототипа функции, чтобы указать компилятору на отсутствие возвращаемого значения. Пример: `int main(void) NO_RETURN;`
`NO_INLINE`	Добавляется к прототипу функции, чтобы запретить компилятору inline подстановку. Пример: `static void NO_INLINE func(int *args);`

2.3. Использование отладчика gdb

Для комплексной отладки программ в ОС UNIX используется отладчик gdb (The GNU debugger). Все IDE в Linux-системе с возможностью отладки обычно имеют текстовый режим с gdb в качестве основы. Если gdb не установлен в системе, то для его установки необходимо выполнить команду:

```
sudo apt-get install gdb
```

Для того чтобы начать отладку, необходимо запустить Pintos с опцией отладки --gdb:

```
pintos --qemu --gdb --run alarm-single
```

Теперь необходимо связать gdb с эмулятором. Для этого необходимо открыть новый терминал, перейти в каталог build и выполнить команду запуска отладчика:

```
cd pintos/src/threads/build
pintos-gdb kernel.o
```

В запустившемся отладчике gdb необходимо подключиться к эмулятору с запущенным pintos:

```
(gdb) target remote localhost:1234
```

Для работы с pintos можно использовать полный функционал gdb. Список наиболее часто используемых команд приведен в табл. 5.

Таблица 5. Часто используемые команды gdb

Команда	**Применение:**
`break function`	Установить точку установа на функции с имененем function. Например, можно использовать команду break main, чтобы установить точку останова на функции main() ядра, вызываемой при

	его старте.
`break file:line`	Установить точку останова на конкретной строке кода
`break *address`	Установить точку останова на указанном адресе. Необходимо использовать префикс *0x*, чтобы указать адрес шестнадцатеричным числом.
`c`	Продолжить выполнение до следующей точки останова или нажатия комбинации клавиш Ctrl+C
`step`	Выполнить одну строку исходного кода и остановиться после ее выполнения. Если встретится вызов функции — остановка будет произведена *внутри* вызванной функции. Дополнительным параметром можно указать число — сколько шагов выполнить за один прием.
`next`	Тоже, что и step, но не выполняется остановка внутри вызываемых функций, вызываемые функции рассматриваются как один оператор выполнения. Дополнительным параметром можно указать число — сколько шагов выполнить за один прием.
`p expression`	Вычисляет заданное в expresson выражение и выводит его значение. Если выражение содержит обращение к некоторй функции, данная функция будет вызвана.
`bt`	Выводит текущую трассировку (содержание стека вызовов). Команда представляет собой аналог вызова утилиты backtrace.
`p/a address`	Выводит на экран имя функции или переменной расположенной по указанному адресу.
`disassemble function`	Производит дисассемблирование функции function.

Пример отладочного сеанса:

```
pintos --qemu --gdb -- -q run alarm-single
```

В другом терминале:

```
cd pintos/src/threads/build
pintos-gdb kernel.o
(gdb) target remove localhost:1234
Remote debuggin using localhost:1234
0x0000fff0 in ?? ()
(gdb) break main
Breakpoint 1 at 0xc00201b3: file ../../threads/init.c, line 78
(gdb) break test_alarm_single
Breakpoint 2 at 0xc00294c6: file ../../tests/threads/alarm-wait.c, line 17
(gdb) c
Continuing.

Breakpoint 1, main() at ../../threads/init.c:78
78 {
(gdb) step
82      bss_init();
(gdb) step
bss_init () at ../../threads/init.c:150
150     memset (&_start_bss, 0, &_end_bss - &_start_bss);
(gdb) next
```

```
main () at ../../threads/init.c:85
85       argv = read_command_line ();
(gdb) next
86       argv = parse_options(argv);
(gdb) next
90       thread_init ();
(gdb) c

Breakpoint 2, test_alarm_single () at ../../tets/threads/alarm-wait.c:17
17     {
(gdb) step
18        test_sleep (5,1);
(gdb) step
test_sleep (thread_cnt=thread_cnt@entry=5, iteration=iterations@entry=1)
  at ../../tests/threads/alarm-wait.c:52
52     {
(gdb) bt
#0  test_sleep (thread_cnt=thread_cnt@entry=5, iteration=iterations@entry=1)
  at ../../tests/threads/alarm-wait.c:52
#1 0xc00294d8 in test_alarm_single () at ../../tests/threads/alarm-wait.c:18
#2 0xc00290ad in run_test(name=0xc0007d45 "alarm-single")
  at ../../tests/threads/tests.c:56
#3 0xc00201a0 in run_task(agv=0xc0036624 <argv+4>)
  at ../../threads.init.c:290
#4 0xc002065c in run_actions(agv=0xc0036624 <argv+4>)
  at ../../threads.init.c:340
#5 main () at ../../threads/init.c:133
(gdb) p thread_cnt
$2 = 5
(gdb) p iterations*10
$3 = 10
(gdb) p iterations*10+50
$3 = 60
(gdb) p/a 0xc00294d8
$5 = 0xc00294d8 <test_alarm_single+18>
(gdb) next
60     ASSERT(!thread_mlfqs)

...

(gdb) quit
Detaching from program …/kernel.o, Remote target
Ending remote debugging.
```

Альтернативой использования gdb из командной строки является графическая надстройка ddd (Data Display Debugger). Чтобы выполнять отладку в графическом режиме, необходимо запустить находясь в каталоге build программу с параметрами:

```
ddd --gdb --debugger pintos-gdb
```

2.4. Утилита backtrace

Когда происходит ошибка "kernel panic", ядро ОС Pintos выводит ряд трассировок, которые позволяют определить, какие функции выполнялись в момент обнаружения ошибки. Адреса функций в трассировке указаны в шестнадцатеричном виде, поэтому Pintos предоставляет утилиту backtrace, которая позволяет перевести шестнадцатеричные числа в удобочитаемый формат.

Например, если запустить ядро pintos с указанием теста, которого не существует:

```
pintos --qemu -- -q run test
```

Будет получен следующий вывод:

```
...
Boot complete.
Executing 'test':
Kernel PANIC at ../../tests/threads/tests.c:60 in run_tets(): no test named "test"
Call stack: 0xc00283989 0xc002a2ee 0xc00207d1 0xc0020771.
...
```

С помощью утилиты backtrace можно получить по указанным адресам имена функций. Для этого утилиту необходимо вызывать из каталога build и в качестве аргументов передать ей имя файла с бинарным образом pintos, а также интересующие адреса:

```
backtrace kernel.o 0xc00283989 0xc002a2ee 0xc00207d1 0xc0020771
0xc0028389: debug_panic (.../../lib/kernel/debug.c:38)
0xc002a2ee: intrff_stub (.../../threads/intr-stubs.S:0)
0xc00207d1: run_task (.../../threads/init.c:292)
0xc0020771: run_actions (.../../threads/init.c:341)
```

Трассировка не является оптимальным средством отладки, так как часто может быть искажена из-за повреждения стека или произведенных компилятором оптимизаций. Тем не менее, в общем случае информации, предоставляемой трассировкой достаточно, чтобы определить местонахождение основных ошибок.

В дополнение к утилите, Pintos предлагает несколько функций, определения которых расположены в lib/debug.h. Это функции debug_backtrace(), которая может быть размещена в любом участке кода и будет выводить текущую трассировку, и debug_backtrace_all(), которая выводит трассировки всех исполняющихся в данный момент процессов.

III. Архивация результатов модификаций

Для архивации результатов работы необходимо выполнить очистку каталога от всех бинарных файлов и выполнить упаковку исходных текстов архиватором:

```
cd pintos/src
make clean
cd ../../
tar –zcf pintos_result.tar.gz pintos
```

В последней команде pintos_result.tar.gz – имя нового файла с архивом, pintos — существующий каталог исходного кода pintos с вашими модификациями. В результате будет создан архив pintos_result.tar.gz, содержащий результаты работы. Данный архив может быть перенесен на любой компьютер с настроенной средой для работы с ОС pintos.