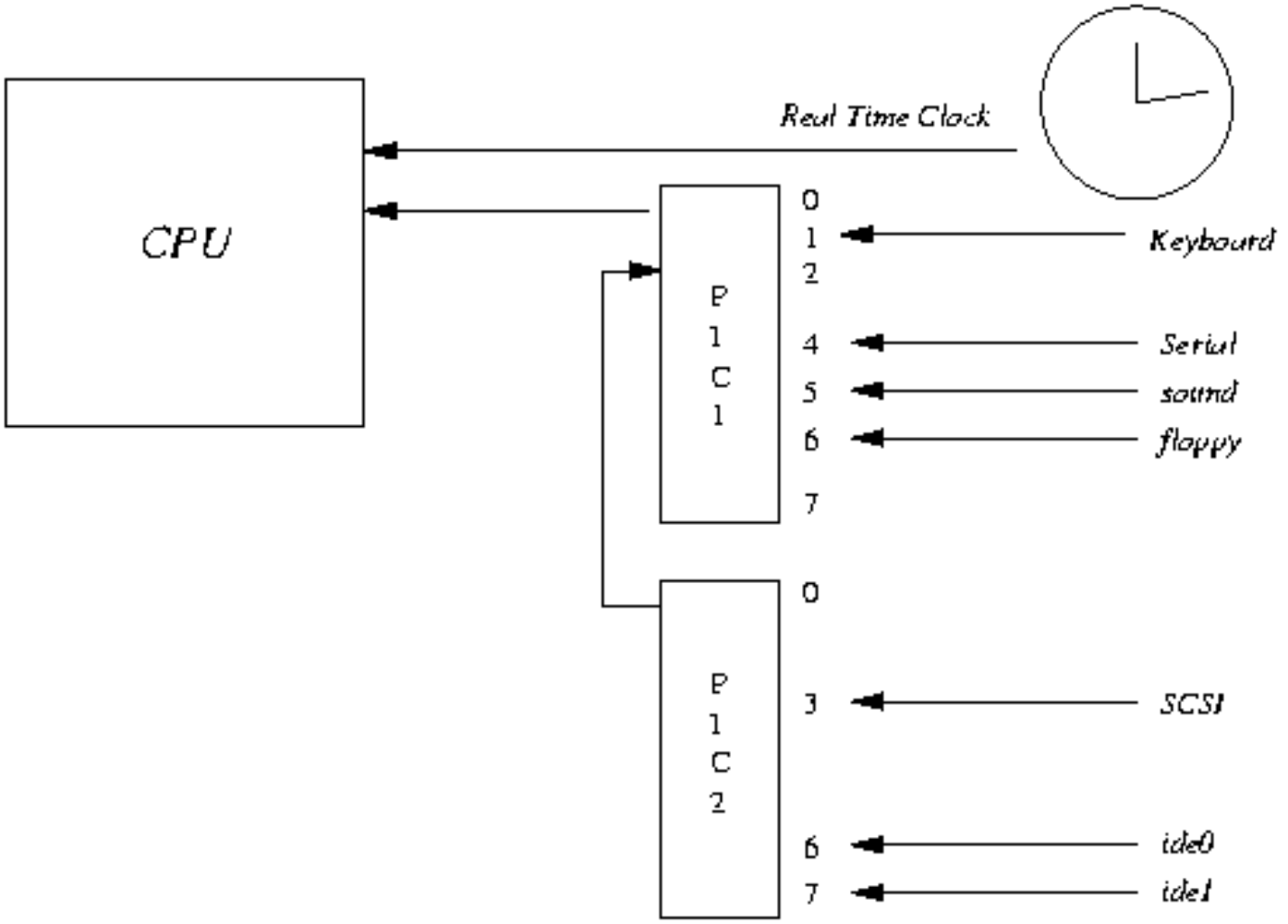
Прерывания

- Прерывание — механизм уведомления ЦП о произошедшем событии со стороны внешнего устройства
- В конце выполнения каждой инструкции ЦП проверяет поступившие сигналы о прерывании и, если прерывание поступило, начинает выполнять последовательность действий обработки прерывания
- Слово состояния процессора содержит бит IF, которое запрещает обработку маскируемых прерываний

Схема поступления прерываний в ЦП



Номер прерывания

- Каждое прерывание (как программное, так и аппаратное) имеет номер
- На i386 номер — [0;255] (8 бит)
  - 0..31 — исключения процессора и неблокируемые прерывания
  - 32..47 — прерывания от внешних устройств
  - 48..255 — программные прерывания
- Вектор обработки прерываний (Interrupt Descriptor Table) содержит информацию, необходимую для обработки всех прерываний

Действия ЦП при поступлении прерывания на i386

- Если изменился уровень привилегий процессора, переключиться на системный стек
  - Сохранить значения регистров ss, esp
  - Загрузить регистры системного стека из дескриптора текущей задачи
- Сохранить на стеке регистры eflags, cs, eip
- Загрузить значения cs, eip из соответствующего элемента вектора обработки прерывания
- Выполняется обработчик прерывания
- После окончания работы обработчика восстанавливаются сохраненные значения регистров

Типы прерываний

- Короткое — выполняется с заблокированными прерываниями, должно завершиться очень быстро (например, таймер)
- Долгое — прерывания (кроме обрабатываемого) разрешены

Действия в начале обработки прерывания

- Сохранение всех регистров процессора (реализация на ассемблере)
- Разрешение прерываний (для долгих прерываний)
- Вызов всех зарегистрированных обработчиков по очереди (реализация на Си)
- Восстановление всех регистров (на ассемблера)
- Выход из обработчика

Действия в обработчике

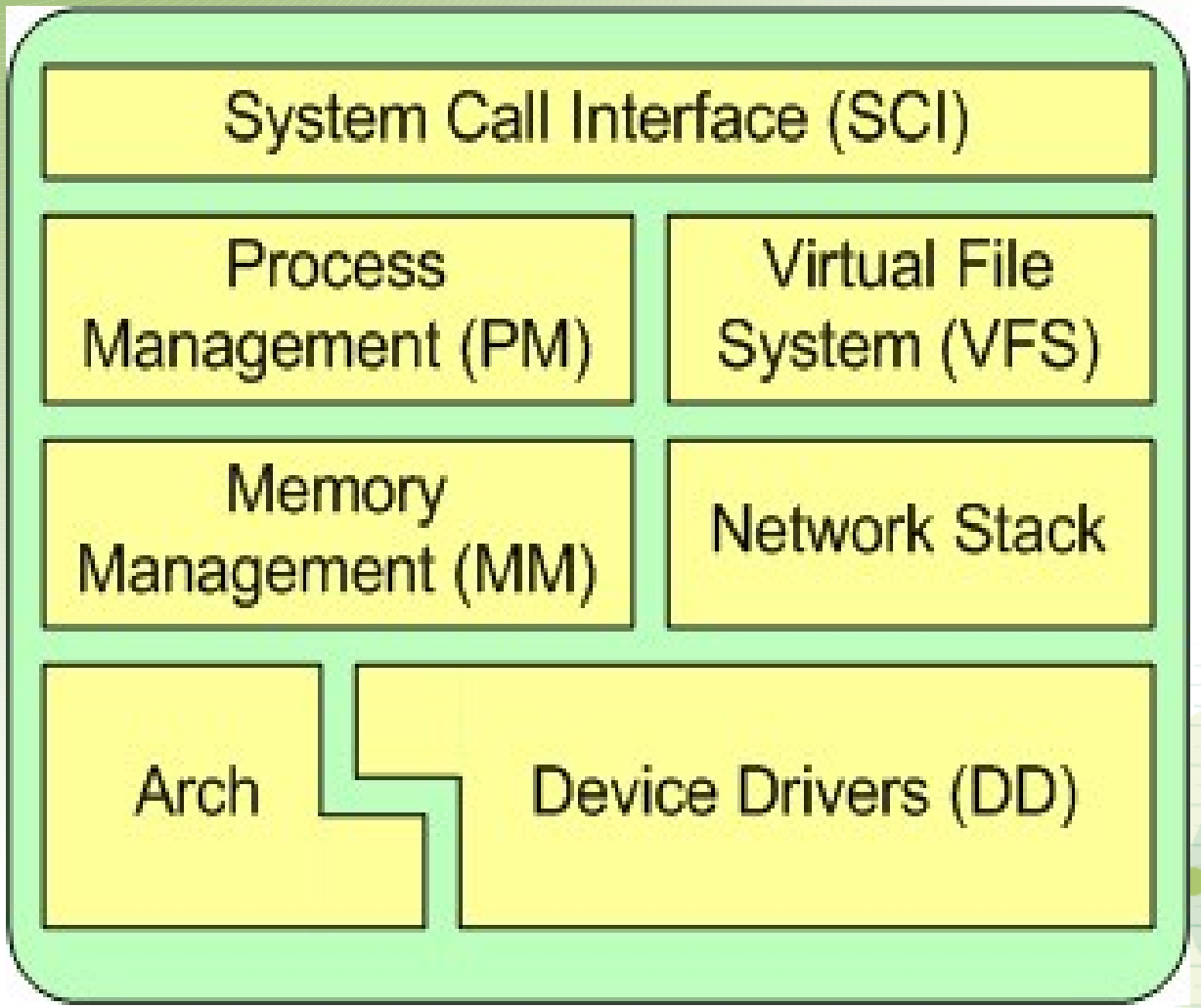
- Обработчик прерывания выполняется без пользовательского контекста и не может выполнять многие действия (засыпать, выделять память, планировать процессы)
- Обычно обработчик прерывания считывает информацию с устройства, помещает ее в буфер драйвера и планирует выполнение основной части обработчика прерывания
- Top-half — быстрая часть обработчика прерывания
- Bottom-half — основная (долгая) часть

Пример

```
void short_do_tasklet (unsigned long);
DECLARE_TASKLET(short_tasklet,short_do_tasklet, 0);

void tl_interrupt(int irq, void *dev_id,
                  struct pt_regs *regs)
{
  do_gettimeofday(tv_head);
  incr_tv(&tv_head);
  tasklet_schedule(&short_tasklet);
  bh_count++;
}
```

Общая архитектура ядра



Классификация ядер ОС

- Монолитное ядро
- Микроядро
- Гибридное ядро

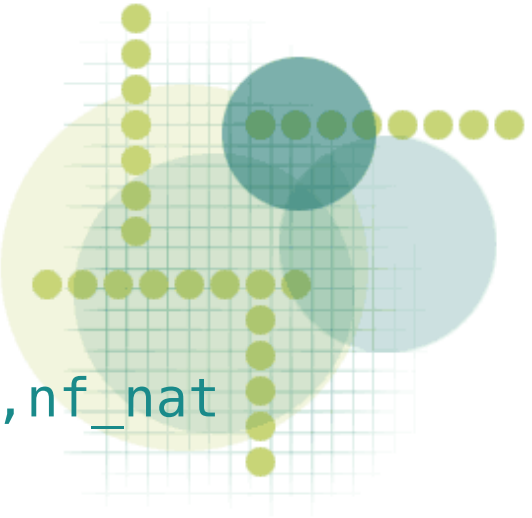
Монолитное ядро

- Все ядро находится в одном адресном пространстве, использует общие структуры данных и взаимодействует с помощью вызовов функций
- (+) простота взаимодействия компонент
- (+) отсутствие переключений контекста
- (-) ошибка в драйвере может обрушить все ядро

Загружаемые модули

- Повышение гибкости монолитных ядер
- Части ядра (драйвера устройств, ФС и прочее) могут подгружаться в адресное пространство и (иногда) выгружаться

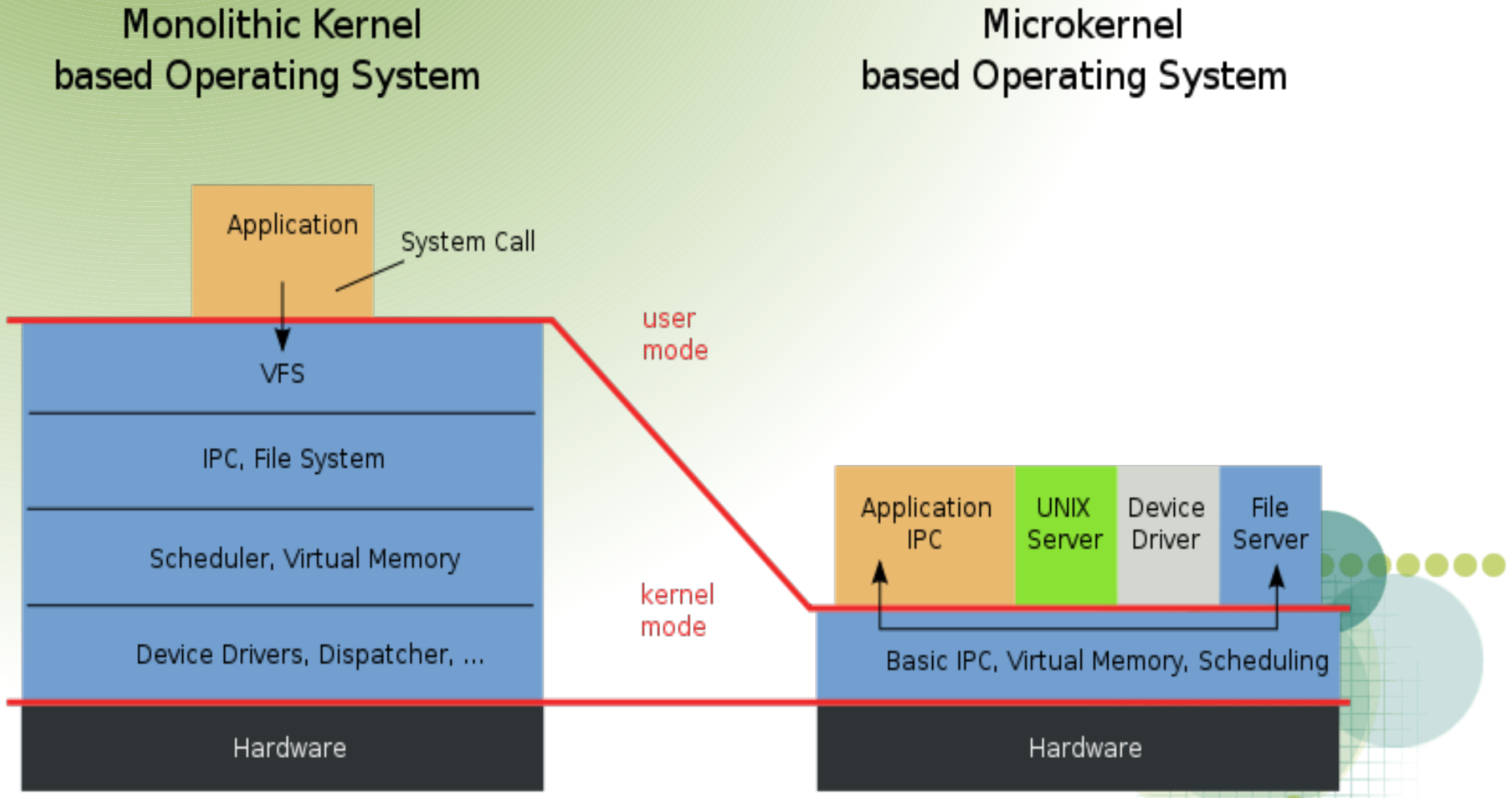
```
Module                  Size  Used by
tun                    14596  0
ipt_REJECT              6912  2
xt_tcpudp               6784  12
xt_state                6016  14
iptable_filter          6656  1
iptable_nat             9224  1
nf_nat                 19736  1 iptable_nat
nf_conntrack_ipv4      12296  17 iptable_nat,nf_nat
```



Микроядро

- В ядре оставляется необходимый минимум: планирование, управление памятью, IPC
- Все остальные компоненты ядра выносятся в серверы, работающие в изолированных адресных пространствах
- Компоненты взаимодействуют друг с другом с помощью механизма посылки сообщений

Сравнение ядер



Достоинства и недостатки

- (+) Высокая степень изоляции компонент ядра
- (+) Возможность рестарта сбойного драйвера ядра
- (+) Возможность планирования компонент ядра и переключения выполнения между компонентами ядра
- (-) Накладные расходы на передачу сообщений
- (-) Отсутствие общих структур данных и необходимость реализации сложных протоколов взаимодействия компонент

Микроядерные системы

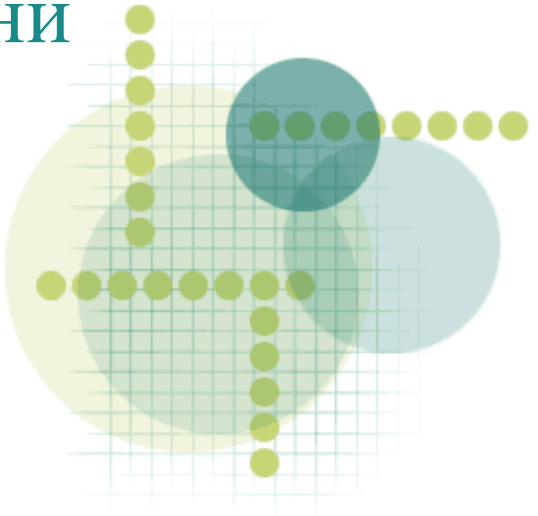
- GNU Hurd
- Mach
- L4
- QNX
- MINIX

Гибридные системы

- В основе ядра лежит микроядро
- Над микроядром надстраиваются необходимые компоненты ядра
- Микроядро и остальные компоненты работают в одном адресном пространстве и в привилегированном режиме
- Пример: MacOS (сервер 4.4BSD, надстроенный над ядром Mach), Windows NT+

ОС реального времени

- Цель: гарантировать предсказуемость поведения ядра во всех ситуациях
- Удовлетворить директивным требованиям на время реакции на события
- Предоставить процессам возможности для реализации систем реального времени



Задачи, решаемые в ядре

- Минимизация латентности прерываний (то есть от подачи сигнала прерывания до начала работы обработчика)
- Минимизация времени переключения процессов
- Предсказуемость поведения подсистемы управления памятью (отключение MMU или блокировка страниц в ОЗУ)
- Возможность переключения контекста в произвольный момент времени (в том числе в ядре)

Kernel preemption

- Традиционно ядра Unix не являлись прерываемыми в ядре
- Процесс, выполняющийся в режиме ядра, не может быть снят с процессора и заменен другим процессом
- Это неприемлемо с точки зрения реального времени, так как вносит непредсказуемые задержки
- Современные версии ядер Unix в большой мере допускают перепланирование процессов в ядре

Подходы к реализации

- Микроядро + все процессы, работающие в одном адресном пространстве (QNX)
- Настройка над ОС общего назначения, в которой ОС общего назначения исполняется как процесс (RTLinux)