Штатные средства отказоустойчивости ОСРВ QNX Neutrino

Методы обеспечения отказоустойчивости

Штатные средства отказоустойчивости ОСРВ QNX Neutrino делятся на два класса:

- Предотвращение отказа системы
  - Микроядерная архитектура
  - Дублирование функций
  - Горячая замена компонентов
  - Адаптивное квотирование
- Восстановление после отказа
  - Технология быстрой загрузки
  - Менеджер высокой готовности

Архитектуры систем реального времени

Исполнямый модуль реального времени

- Защиты памяти нет
- Приложения, драйверы и протоколы "живут" в пространстве ядра

Монолитное ядро

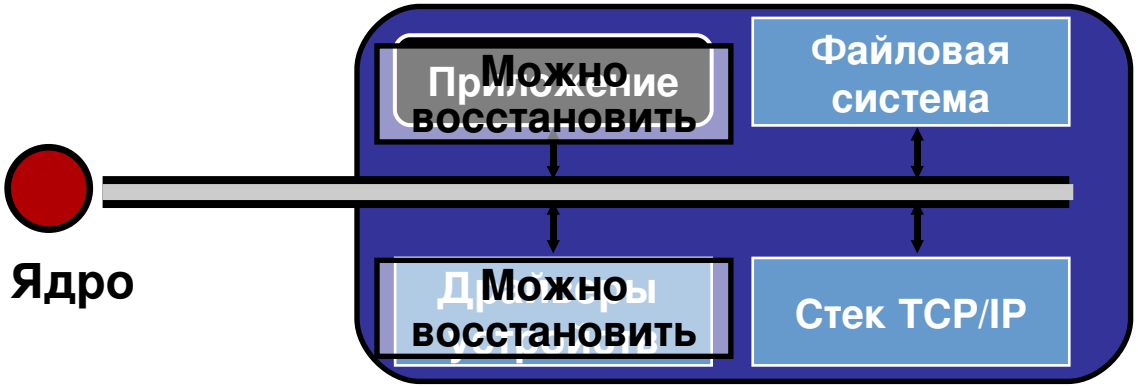
(NT / Unix / и т.п.)

- MMU, частичная защита
- Защищены только приложения

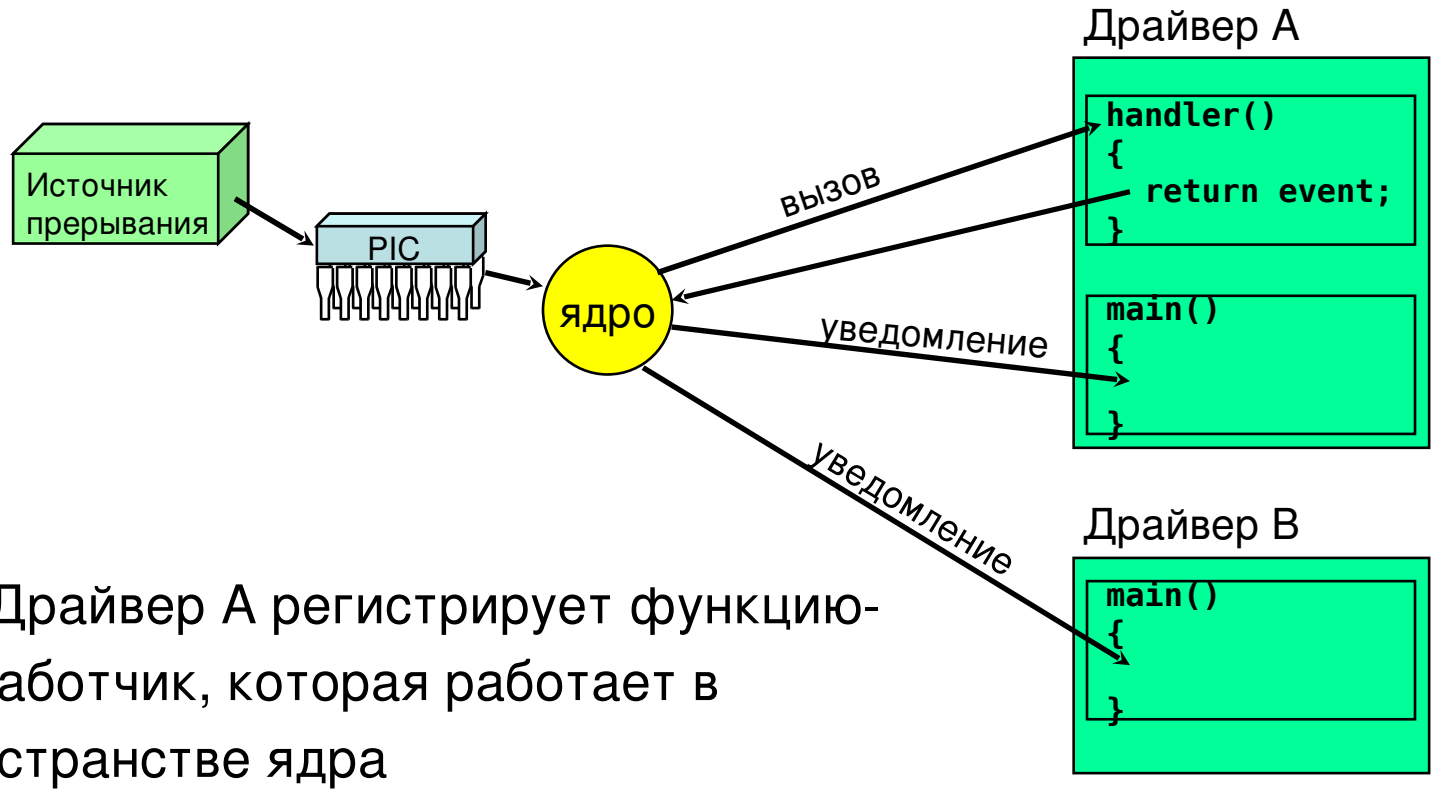
Микроядро

(QNX Neutrino)

- MMU, полная защита
- Защищены приложения, драйверы и протоколы

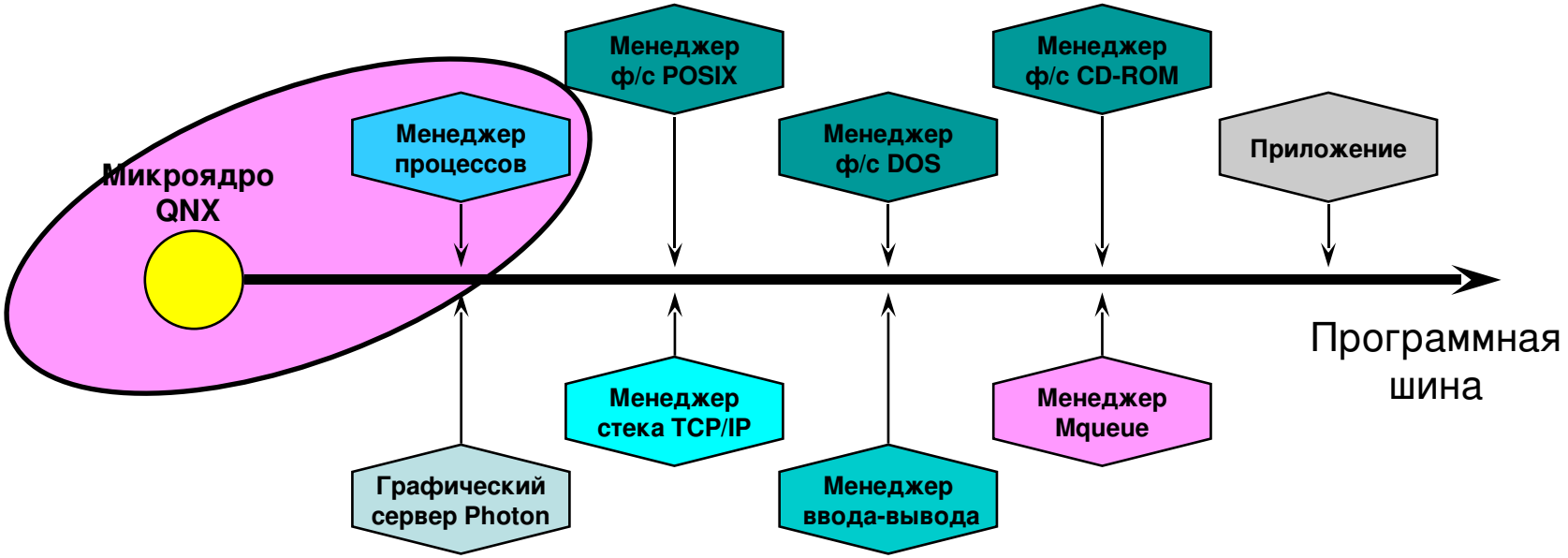


Обработка прерываний в QNX Neutrino



- Драйвер A регистрирует функцию-обработчик, которая работает в пространстве ядра
- Драйвер B регистрирует событие, которое передаётся ему ядром, и обрабатывает его в своём пространстве

Микроядерная архитектура

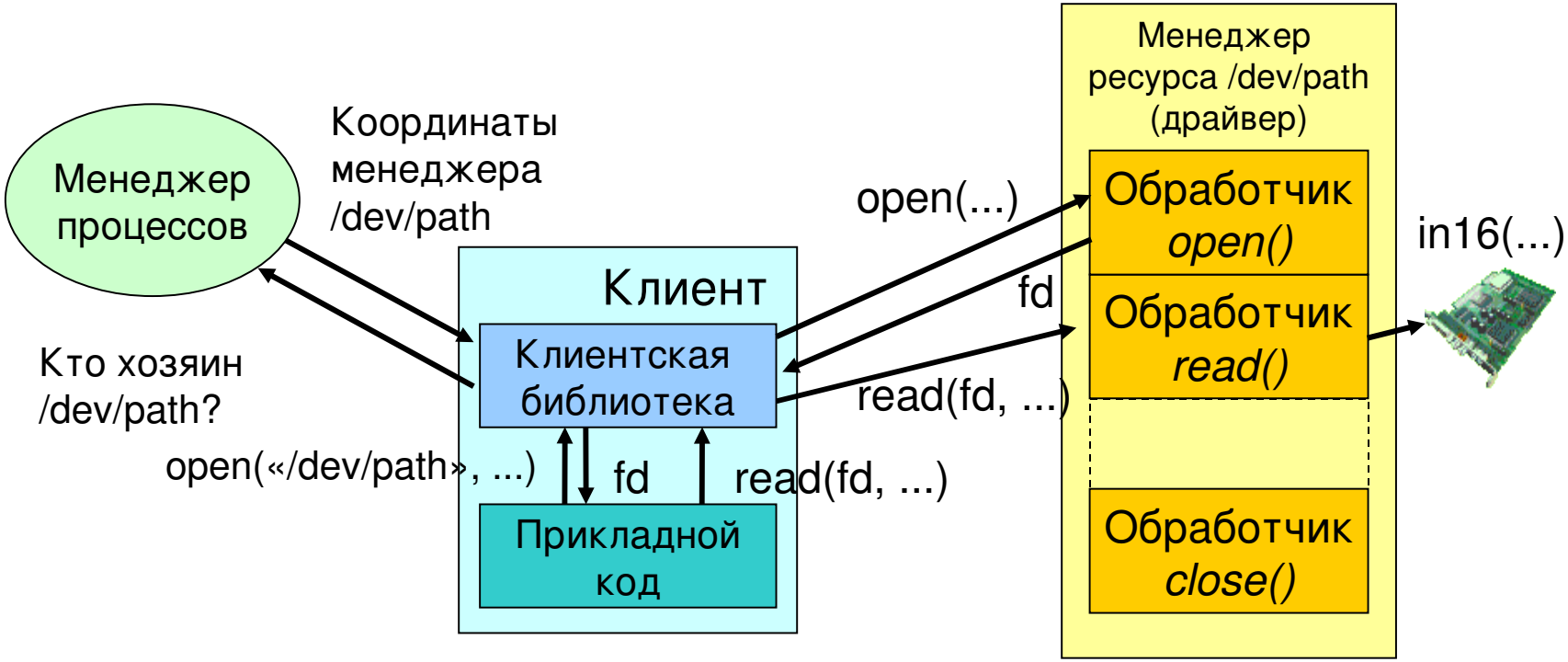


Все драйверы и приложения работают в собственных адресных пространствах

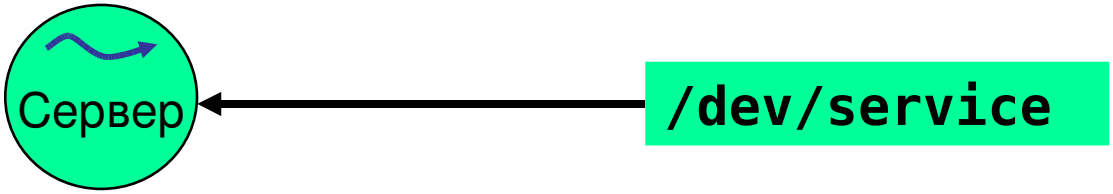
- ✓ микроядро – единственная точка отказа системы
- ✓ драйверы не вводят новые точки отказа (кроме функций-обработчиков прерываний)
- ✓ при разработке можно использовать все возможные инструменты отладки и анализа

Администраторы ресурсов

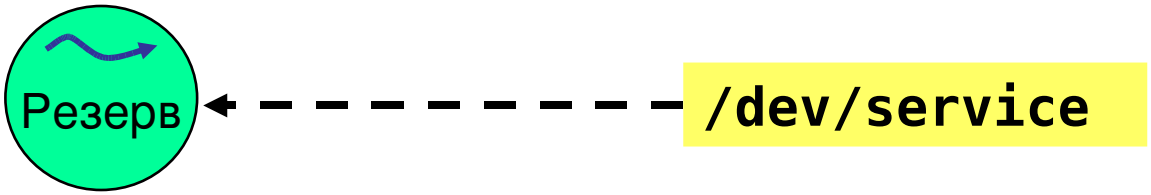
Администратор ресурса – программа, которая предоставляет некоторый сервис, создавая специальный файл и реагируя на обращения к нему



Дублирование сервисов



resmgr_attach(…,“/dev/service”,…);

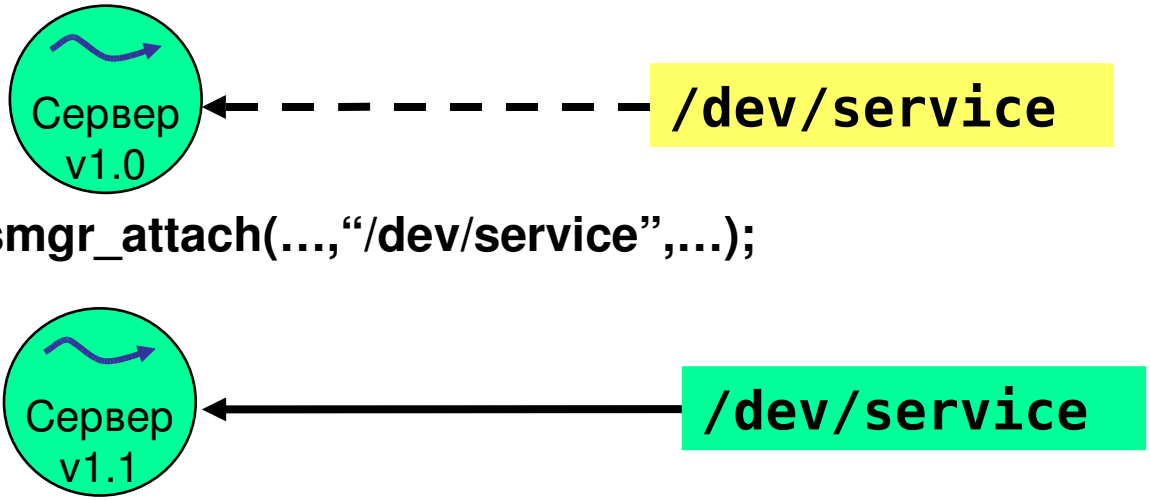


resmgr_attach(…,“/dev/service”,_RESMGR_FLAG_AFTER,…);

Один и тот же файл может обслуживаться несколькими администраторами ресурса

Если основной сервер не может отреагировать на клиентский запрос, на него реагирует резервный сервер

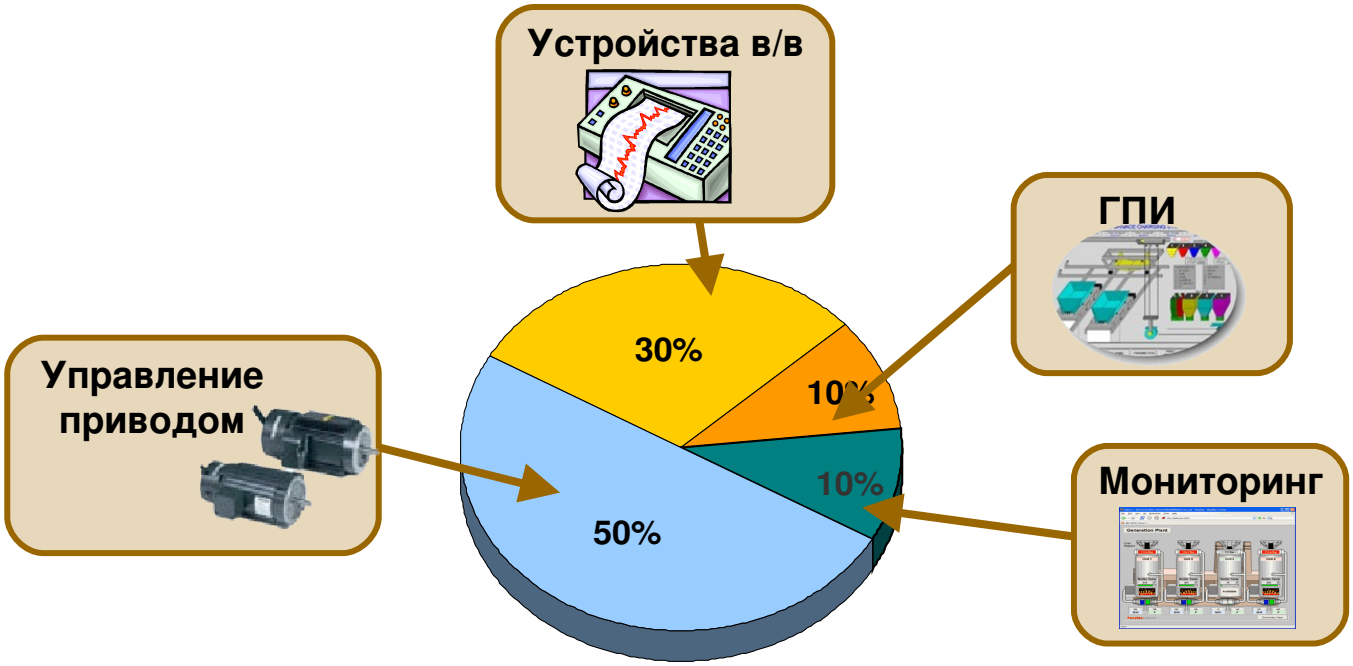
Горячая замена сервисов



Можно зарегистрировать новый сервер как основной, при этом текущий сервер автоматически становится резервным

Все запросы **`open(“/dev/service”, ...)`** немедленно начинают поступать на Сервер v1.1

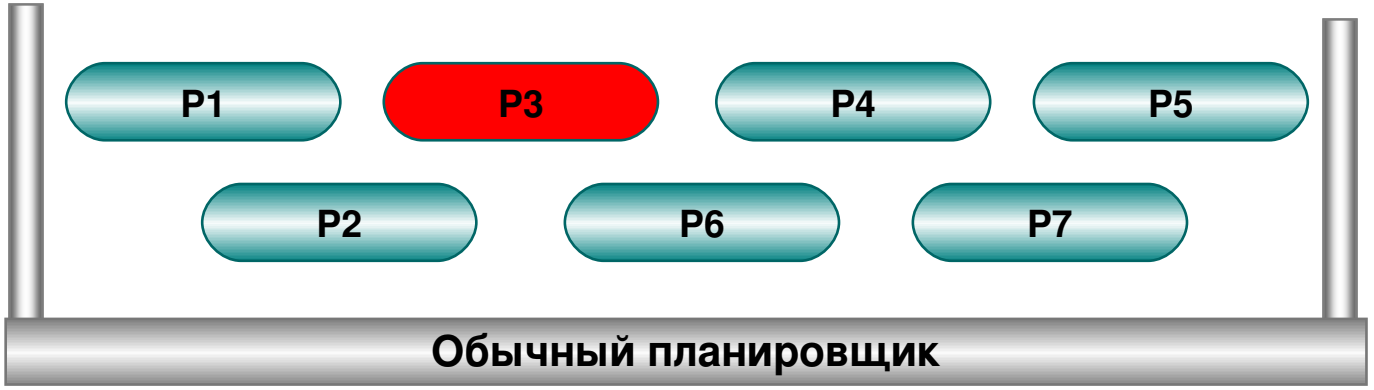
Адаптивное квотирование



Время ЦП

Адаптивное квотирование – способ гарантировать определённую долю некоторого ресурса блоку приложений

Адаптивное квотирование



Приложениям нельзя доверять:

- неудачное назначение приоритетов потокам
- возможные ошибки в работе с памятью
- DDOS-атаки

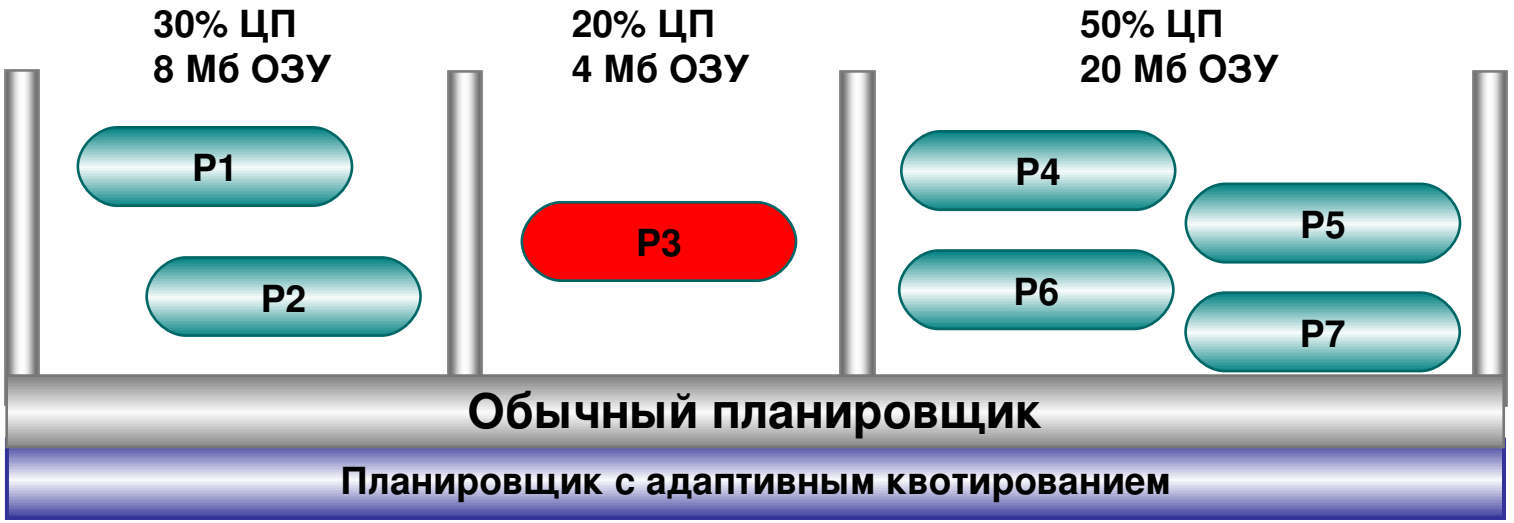
→ монополизация ресурсов

Обычный планировщик ОСРВ QNX

- гарантирует время отклика согласно приоритетам потоков
- НЕ гарантирует, что приложение вообще получит доступ к ресурсу

→ нехватка ресурсов

Адаптивное квотирование

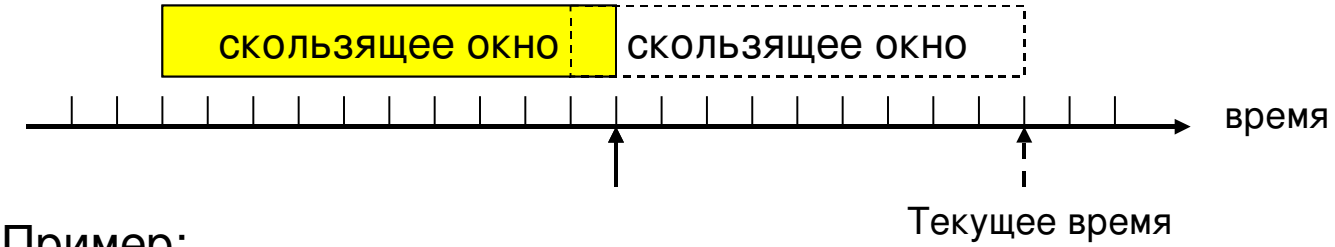


Планировщик с адаптивным квотированием

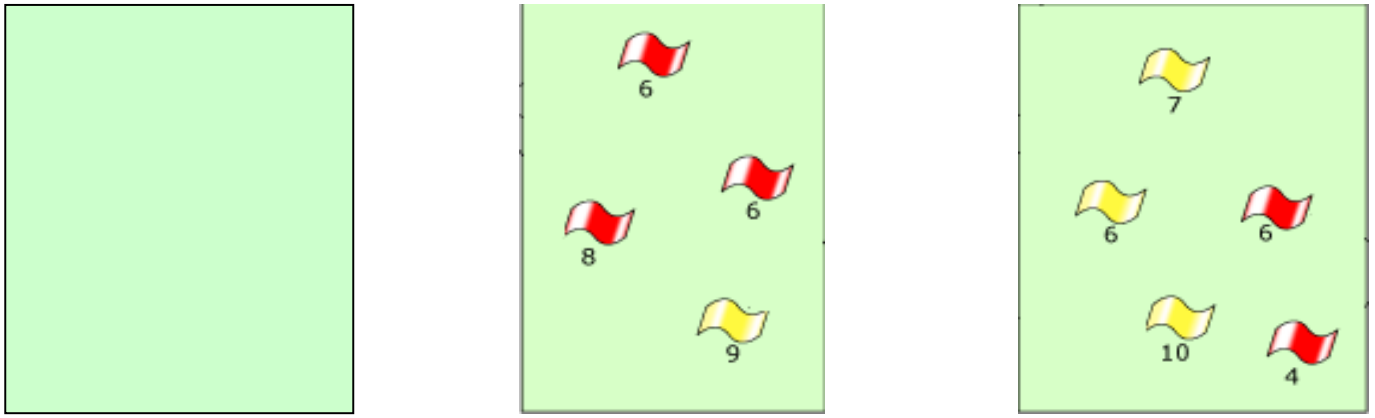
- создает «виртуальные стены» между группами приложений
- следит за расходом бюджетов
- гарантирует, что любое приложение потребит не больше ресурсов, чем выделено его разделу
- распределяет свободное время между разделами при неполной нагрузке

Адаптивное квотирование

Для оценки потребления бюджетов разделов используется алгоритм скользящего окна



Пример:



Раздел System – 70%

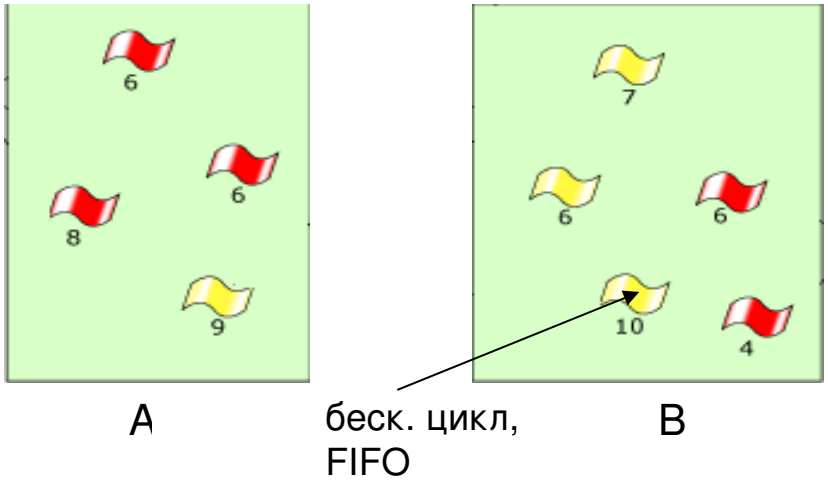
Раздел A – 20%

Раздел B – 10%

блокированы

готовы/работают

Неполная нагрузка



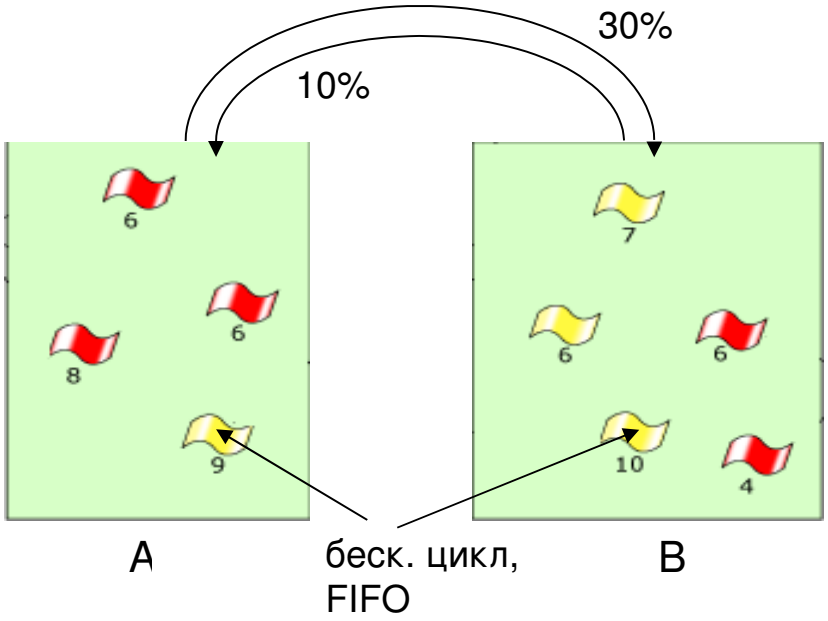
- управление передаётся самому приоритетному потоку системы (по умолчанию)

Раздел	Бюджет, %	Использ., %
System	60	0
A	30	0
B	10	100

- можно распределить свободное время пропорционально между разделами
  - функция SchedCtl() с флагом SCHED_APS_FREETIME_BY_RATIO
  - утилита ap modify -S freetime_by_ratio

Раздел	Бюджет, %	Использ., %
System	60	0
A	30	75
B	10	25

Полная нагрузка



- время делится между разделами согласно выделенным бюджетам

Раздел	Бюджет, %	Использ., %
System	60	60
A	30	30
B	10	10

Существует вероятность, что прерванный поток не выполнит свою работу вовремя…

Критически важные потоки

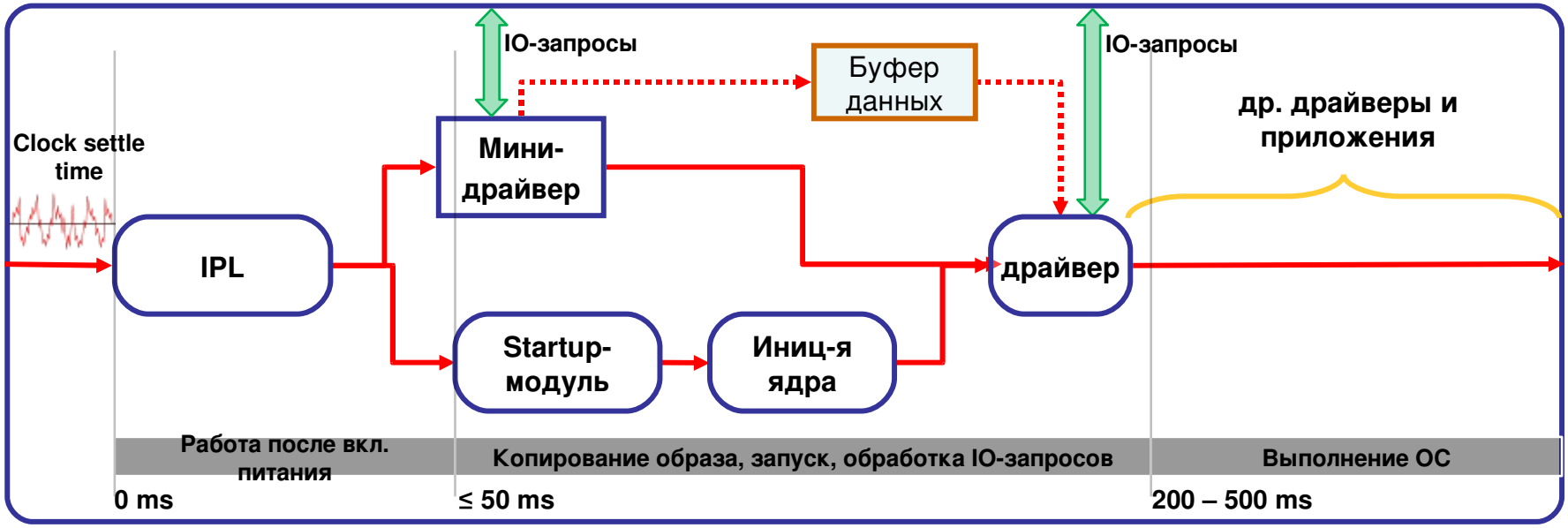
- Могут выполняться в случае, если выделенный разделу бюджет исчерпан
  - потоки ввода-вывода, обработки прерываний и т. п.
- Помечаются специальным флагом
  - С помощью функции SchedCtl()
  - Через механизм событий

```
struct sigevent my_event;
SIGEV_PULSE_INIT (&my_event, coid, 10, MY_SIGNAL_CODE, 6969);
SIGEV_MAKE_CRITICAL(&my_event);
```

  Поток, получивший такое событие, становится критическим

- Разделу необходимо выделить критический бюджет
  - время работы критических потоков в течение скользящего окна

Мгновенная активация устройств



- Во время перезапуска сначала стартует минидрайвер
  - «подхватывает» данные устройства/шины раньше, чем заработает ядро
- После запуска ядра стартует полноценный драйвер
  - принимает собранные данные от минидрайвера

Отказоустойчивые файловые системы

ОСРВ QNX Neutrino включает в себя набор файловых систем для решения широкого круга задач:

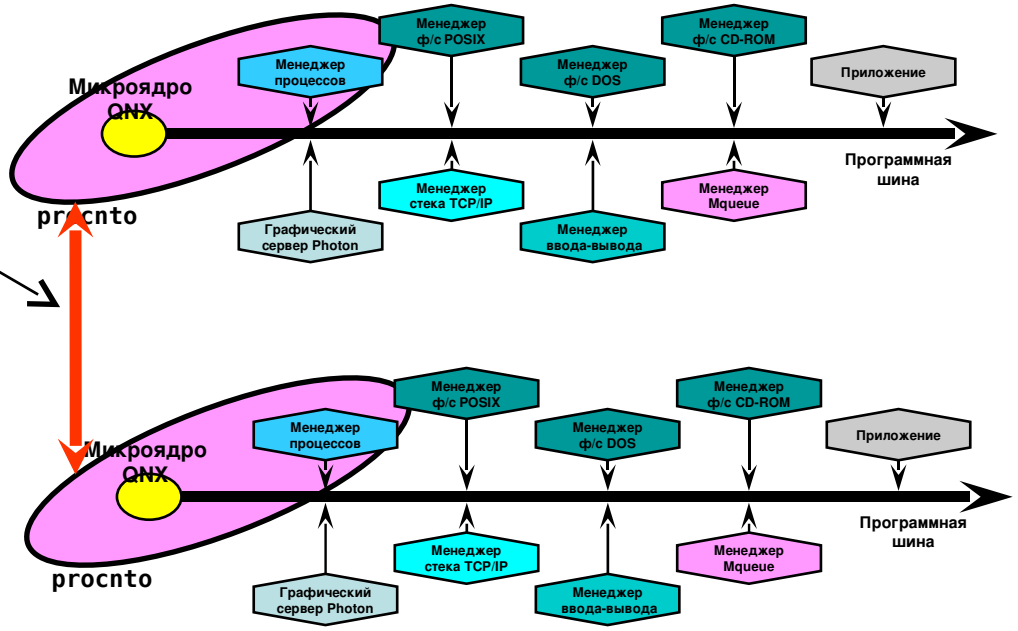
- ОЗУ-резидентные (Image, Shmem, io-blk, devb-ram)
- Для ППЗУ (FFS)
- Дисковые (QNX4, FAT32, NTFS, HFS+, ISO 9660, UDF)
- Сетевые (NFS, SMB)
- Виртуальные (inflator, пакетная)

Файловые системы с повышенной отказоустойчивостью:

- ✓ Power-Safe filesystem (QNX6) – для жёстких дисков
- ✓ Embedded transaction filesystem (ETFS) – для Flash-памяти

Распределённая вычислительная среда

Технология QNX TDN
на базе протокола Qnet
(4-й уровень ISO OSI)



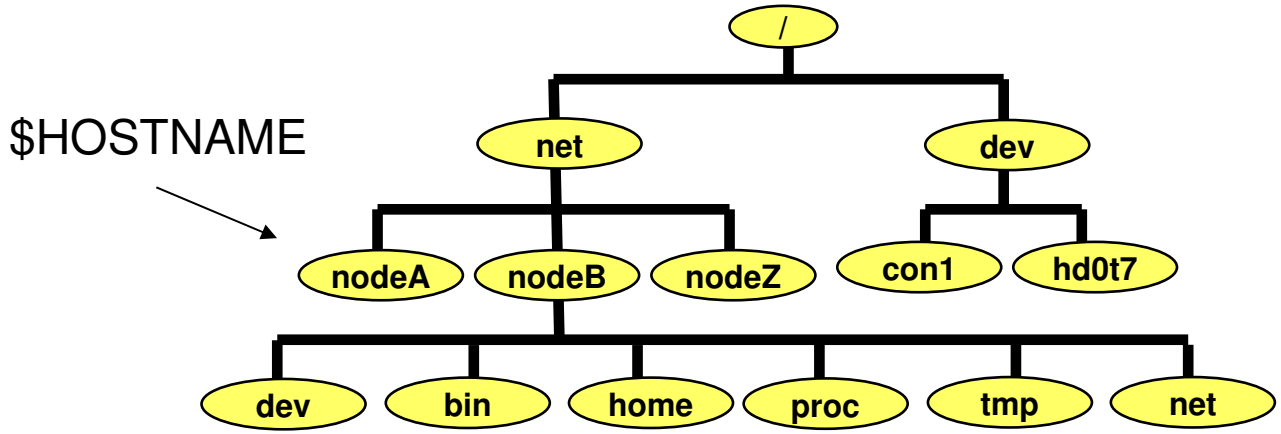
Qnet прозрачно «транслирует» действия программ на удалённые узлы:

- передача сообщений
- доступ к файлам
- доступ к устройствам
- работа с графической оболочкой

Благодаря Qnet можно дублировать приложения, сервисы и устройства, помещая их на разные узлы сети Qnet

Распределённая вычислительная среда

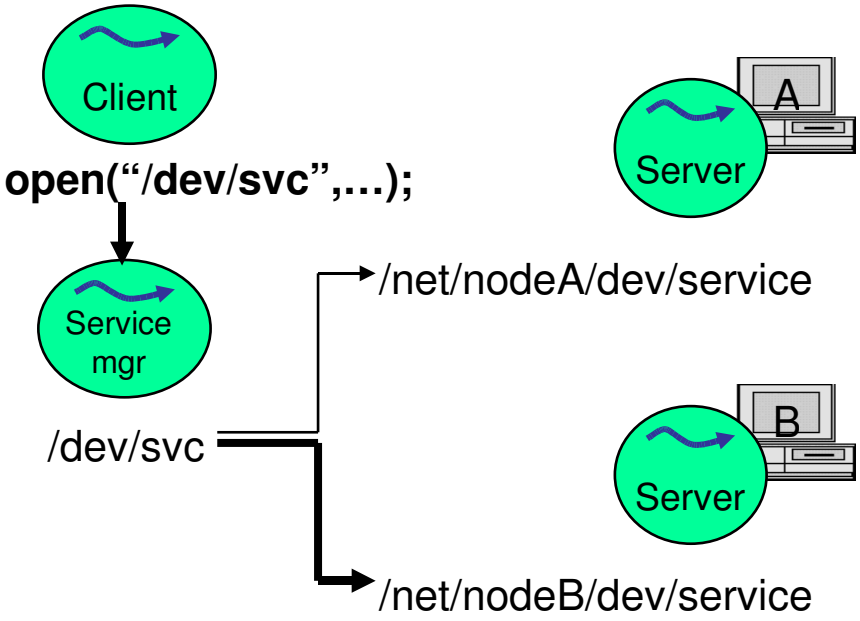
Менеджер Qnet регистрирует префикс /net на своём узле



Запуск программы
в распределенном режиме:

```
on –n nodeB \
–f nodeC \
–t /net/nodeD~preferred:en1/dev/tty0 \
/net/nodeE~exclusive:en2/bin/ls
```

Распределение нагрузки между серверами

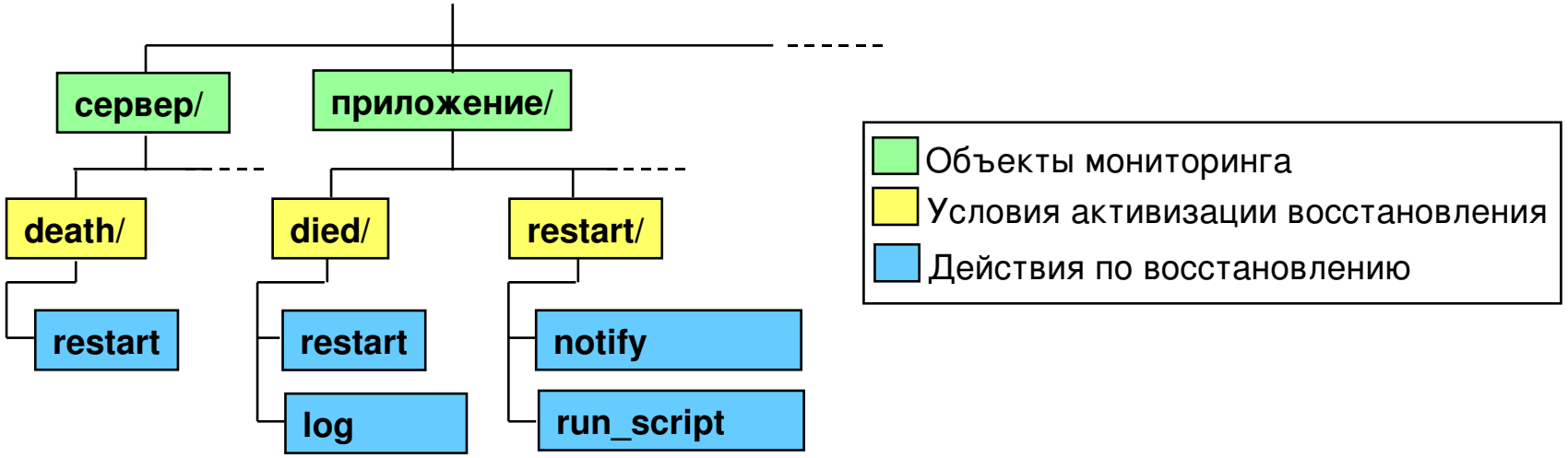


Код перенаправления клиентского запроса:

```
int io_open (resmgr_context_t *ctp, io_open_t *msg,
        RESMGR_HANDLE_T *handle, void *extra)
{
   eflag = msg->connect.eflag;
   ftype = msg->connect.file_type;
   memset(&msg->link_reply, 0, sizeof(msg->link_reply));
   _IO_SET_CONNECT_RET(ctp, _IO_CONNECT_RET_LINK);
   msg->link_reply.file_type = ftype;
   msg->link_reply.eflag = eflag;
      msg->link_reply.nentries = 0;
   msg->link_reply.path_len = strlen(new_path) + 1;
   strcpy(((char *)msg + sizeof(msg->link_reply)), new_path);
   retlen  = sizeof(msg->link_reply) +
     msg->link_reply.path_len;
   return _RESMGR_PTR(ctp, msg, retlen);
}
```

Новый запрос **open("/dev/svc", ...)** поступит на один из узлов

Автоматическое восстановление процессов



Автоматизация восстановления соединений

Цель механизма – «развязать» логику приложения и логику восстановления соединений

Пример кода “восстановителя” соединения:

```
fd = ha_open( “/dev/device”, O_RDONLY, recover_read_fd, “/dev/device”, 0 );
…
int recover_read_fd( int old_fd, void *hdl )
{
  fname = (char *) hdl;
  delay(100); // Здесь выполняются необходимые действия
  new_fd = ha_reopen( old_fd, fname, O_RDONLY );
  return new_fd;
}
```

Заключение

- Технологии QNX включают в себя широкий спектр средств повышения отказоустойчивости ПО
  - снижение вероятности отказа
  - ускорение восстановления после отказа
- Штатная комплектация QNX Neutrino включает в себя высокоэффективные средства отказоустойчивости ПО
- Имеются дополнительные коммерческие технологии высокой готовности
  - FastBoot
  - модули среды исполнения для Adaptive Partitioning, High Availability Manager, …