Лекция 9. Планирование потоков

Архитектура ОС Windows

10 декабря 2014 г.

Организация кода планирования

Определение

Диспетчер ядра: (Kernel Dispatcher) — набор процедур ядра, выполняющих обязанности планировщика задач.

Случаи необходимости планирования

- Переход потока в состояние готовности (ready);
- Переход потока из состояния выполнения (run);
- Изменение приоритета потока;
- Изменение аффинности потока так, что он больше не может работать на текущем процессоре.

Базовый приоритет потока

№		№	Описание
0			поток, обнуляющий страницы
1	–	15	варьируемые (динамические) уровни
16	–	31	уровни реального времени

Таблица 1: уровни приоритета потоков

Приоритеты

IDLE_PRIORITY_CLASS
BELOW_NORMAL_PRIORITY_CLASS
NORMAL_PRIORITY_CLASS
ABOVE_NORMAL_PRIORITY_CLASS
HIGH_PRIORITY_CLASS
REALTIME_PRIORITY_CLASS

Таблица 2: классы приоритета процессов

THREAD_PRIORITY_IDLE
THREAD_PRIORITY_LOWEST
THREAD_PRIORITY_BELOW_NORMAL
THREAD_PRIORITY_NORMAL
THREAD_PRIORITY_ABOVE_NORMAL
THREAD_PRIORITY_HIGHEST
THREAD_PRIORITY_TIME_CRITICAL

Таблица 3: уровни приоритета потоков

Установка приоритета

Win32 SetPriorityClass(), SetThreadPriority()

```
BOOL WINAPI SetPriorityClass(
  __in    HANDLE hProcess,
  __in    DWORD  dwPriorityClass
);

BOOL WINAPI SetThreadPriority(
  __in    HANDLE hThread,
  __in    int    nPriority
);
```

Приоритеты

Класс/Уровень	Idle	Below normal	Normal	Above normal	High	Realtime
Idle: −16	1	1	1	1	1	16
Lowest: −2	2	4	6	8	11	22
Below normal: −1	3	5	7	9	12	23
Normal: 0	4	6	8	10	13	24
Above normal: +1	5	7	9	11	14	25
Highest: +2	6	8	10	12	15	26
Time critical: +16	15	15	15	15	15	31

Таблица 4: отображение классов приоритетов процессов/уровней приоритетов потоков на приоритеты

Уровни прерываний

Рис. 1: уровни запроса прерываний в архитектуре x64

Уровни прерывания

Правила изменения уровней прерывания

- Прерывания с более высоким уровнем вытесняют с низким.
- На время исполнения прерывания IRQL процессора повышается.
- Поток ядра может повышать приоритет процессора явно (`KeRaiseIrql()`/`KeLowerIrql()`) или неявно (захват объектов синхронизации ядра.)

Назначение уровней прерываний

Уровень	Использование
Высокий	Останов системы при помощи `KeBugCheckEx()`
Профилирование	Профилирование ядра
Ошибка питания	Не используется
Межпроцессорный	Запрос другому процессору, например, обновление кэша TLB, завершение или сбой системы
Часы	Системные часы для получения времени или назначение времени выполнения потокам
Синхронизация	Работа планировщика и синхронизации
Устройство k	Приоритезация прерываний устройств

Таблица 5: предустановленные уровни прерываний

Назначение уровней прерываний (окончание)

Уровень	Использование
Исправленная машинная проверка	Извещение ОС о серьёзном, но исправленном состоянии оборудования или ошибке процессора или прошивки
DPC/диспетчеризация, APC	Генерируются программно ядром и драйверами
Пассивный	Не является уровнем

Таблица 6: предустановленные уровни прерываний (окончание)

Программные прерывания

Использование программных прерываний

- Начало диспетчеризации потоков;
- Обработка прерываний, не критичных по времени;
- Обработка завершения таймера;
- Асинхронное выполнение процедуры в контексте потока;
- Асинхронные операции ввода/вывода.

Прерывания DPC

Определение

Прерывания диспетчеризации/отложенного вызова процедур: (*Deferred Procedure Call, DPC*) — используется для откладывания действия до завершения текущей операции (синхронизация доступа к общим структурам ядра).

Использование прерывания

- Планирование потоков;
- Завершение таймера (освобождение ожидающих потоков);

Прерывания APC

Определение

Прерывания асинхронного вызова процедур: (*Asynchronous Procedure Call, APC*) — используется для пользовательским и системным кодом для исполнения в контексте заданного пользовательского потока.

Возможности в отличие от DPC

- Получение ресурсов (объектов);
- Ожидание дескрипторов;
- Вызов ошибок доступа к страницам;
- Вызов системных функций.

Использование APC

Использование прерываний APC

- Исполнительной системой для завершения операции в контексте потока (запись результатов асинхронного ввода/вывода в АП);
- Подсистемой окружения для приостановки/завершения потока, имитации сигналов POSIX, ...;
- Драйверами устройств для передачи полученных данных в поток после перепланировки.
- Функциями Windows API: `ReadFileEx()`, `WriteFileEx()`, ... — функция по завершению операции (вызывается для потока в alertable wait state: `WaitForMultipleObjectsEx()`, `SleepEx()`), ...

Управление повышением приоритета

Win32 SetProcessPriorityBoost(), SetThreadPriorityBoost()

```
BOOL WINAPI SetProcessPriorityBoost(
  __in    HANDLE hProcess,
  __in    BOOL   bDisablePriorityBoost
);

BOOL WINAPI SetThreadPriorityBoost(
  __in    HANDLE hThread,
  __in    BOOL   bDisablePriorityBoost
);
```

Случаи повышения приоритетов

Динамический приоритет

- После завершения операции ввода/вывода (величина определяется драйвером устройства).
- По окончании ожидания событий и семафоров (на 1 уровень).
- После выхода из состояния ожидания на объекте ядра активного (с активным окном) потока (не отключается функцией `SetThreadPriorityBoost()`).
- После пробуждения из-за активности GUI (на 2 уровня).
- Из-за нехватки процессорного времени (инверсирование приоритета).

Правила повышения приоритетов

Динамический приоритет

- Всегда $\geqslant$ базового.
- Изначально $=$ базовому.
- *Повышение приоритета* (priority boost) может выполняться для потоков с базовым приоритетом $\in [0, 15]_{\mathbb{Z}}$.
- Величина динамического приоритета также всегда $\in [0, 15]_{\mathbb{Z}}$.
- Процесс с `NORMAL_PRIORITY_CLASS` с главным окном на переднем плане повышает класс приоритета $\geqslant \forall$ фонового процесса (возврат при переходе в фон).
- После повышения приоритет уменьшается на 1 после каждого завершения кванта времени, до понижения до базового.

Предотвращение голодной смерти

Определение

Инверсирование приоритета: (*priority inversion*) — повышение до 15 динамического приоритета готовых к исполнению (состояние “ready”) потоков, ожидающих $\sim$ 4 с.

Пример

Поток	Приоритет	Состояние
1	высокий	ожидает общего ресурса с потоком 2
2	низкий	исполняет код критической секции
3	средний	исполняется

Предотвращение голодной смерти (окончание)

Правила работы диспетчера настройки баланса

- Проверяет за каждый запуск очередные 16 готовых потоков.
- За 1 проход повышается приоритет не более 10 потоков, далее просмотр прекращается, возобновляется при следующем запуске.

Повышение приоритета в режиме пользователя

Категория	Приоритет	Описание
Высокая	23 – 26	потоки обработки профессионального аудио, приоритет выше остальных потоков кроме критически важных
Средняя	16 – 22	часть мультимедийного приложения переднего плана
Низкая	8 – 15	остальные потоки
Истощённая	1 – 7	исчерпавшие своё использование процессора, продолжат выполнение только после завершения остальных потоков высшего приоритета

Таблица 7: уровни приоритета Multimedia Class Scheduler Service (MMCSS)

Состояния потоков

Состояния потоков в планировщике

Готов: (*ready*) — ожидает выполнения.

Простаивает: (*standby*) — выбран следующим для исполнения на конкретном процессоре.

Выполняется: (*running*) — выполняется, пока на него переключён контекст.

Ожидает: (*waiting*) — самостоятельно начинает ожидание на синхронизирующем объекте или его вынуждает к этому подсистема окружения.

Переходное состояние: (*transition*) — готов к выполнению, но его стек ядра выгружен из памяти.

Завершён: (*terminated*) — после завершения выполнения.

Инициализирован: (*initialized*) — в начале после создания.

Состояния потоков (продолжение)

Состояния потоков в планировщике (окончание)

Готов, отложен: (*deferred ready*) — (Windows Server 2003 и выше) выбран для выполнения на процессоре, но ещё не запланирован.

Состояния потоков (окончание)

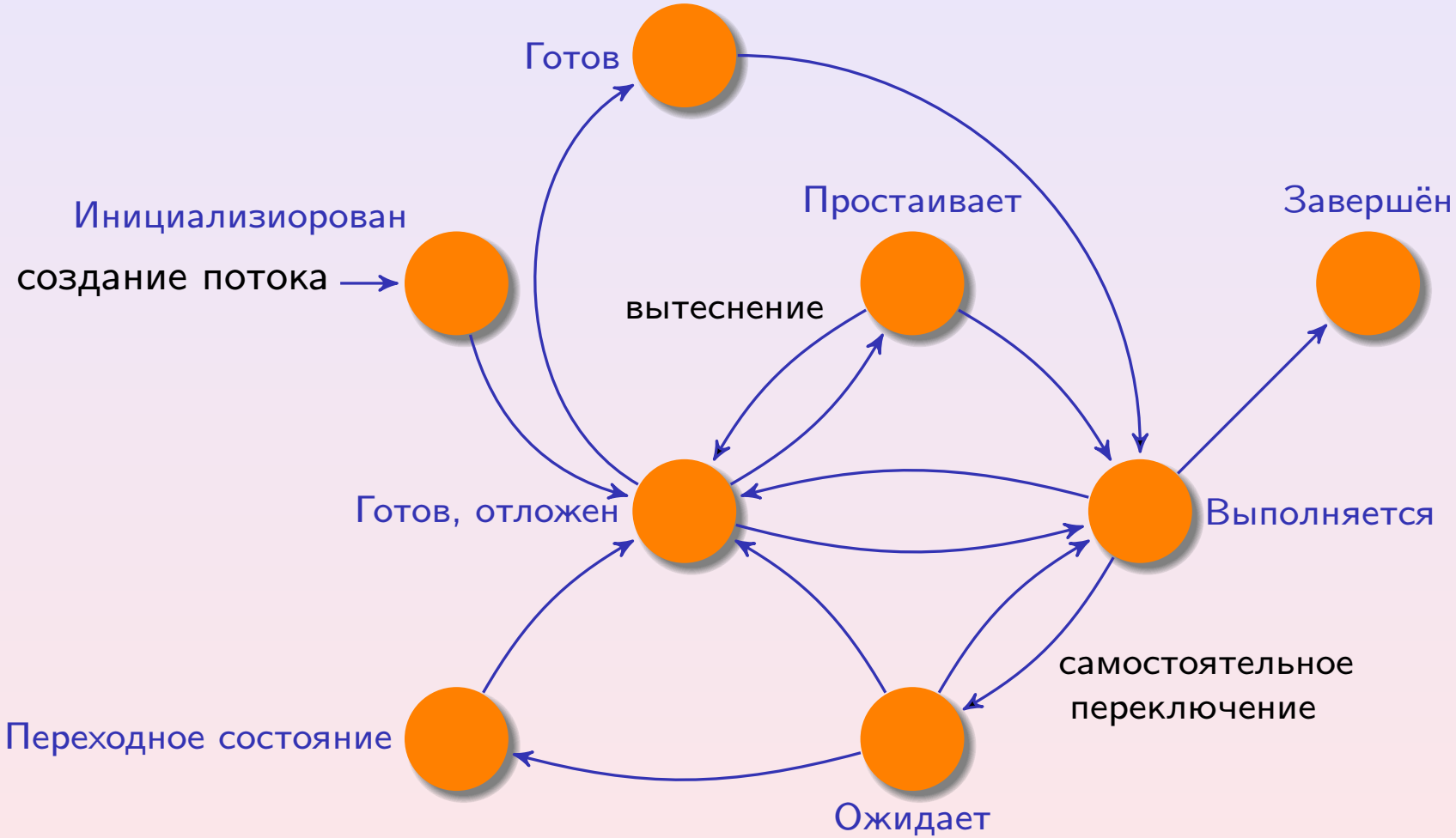

Рис. 2: граф переходов между состояниями потока

Принятие решения по переключению контекста

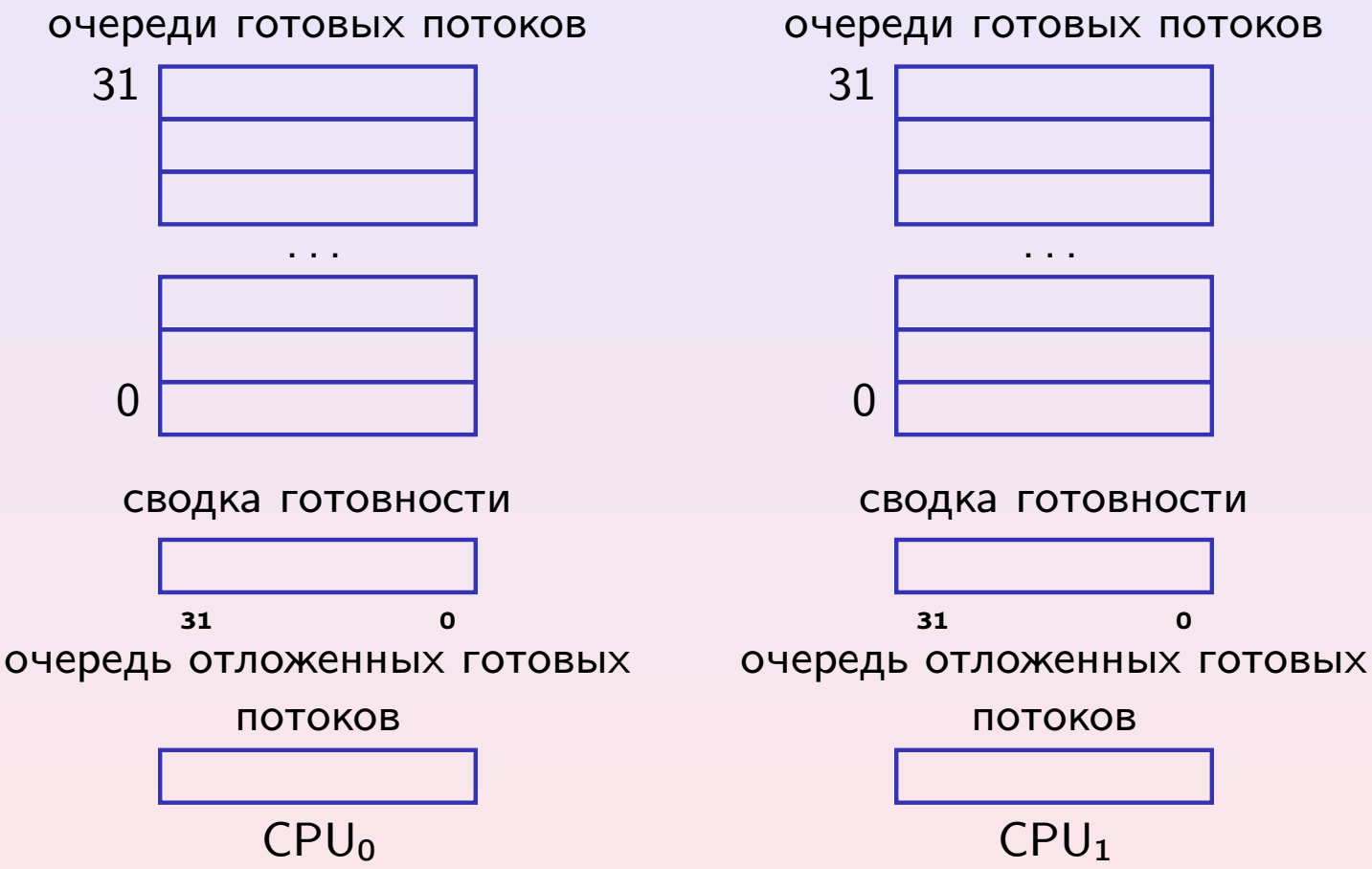

Рис. 3: база данных диспетчера ядра

Правила переключения контекста

Алгоритм переключения

1. Сохранить контекст выполнявшегося потока.
2. Поместить поток в конец (начало) очереди для его приоритета.
3. Найти очередь наивысшего приоритета с готовыми к выполнению потоками.
4. Удалить поток из начала очереди, загрузить его контекст и исполнить.

Выбор кванта времени

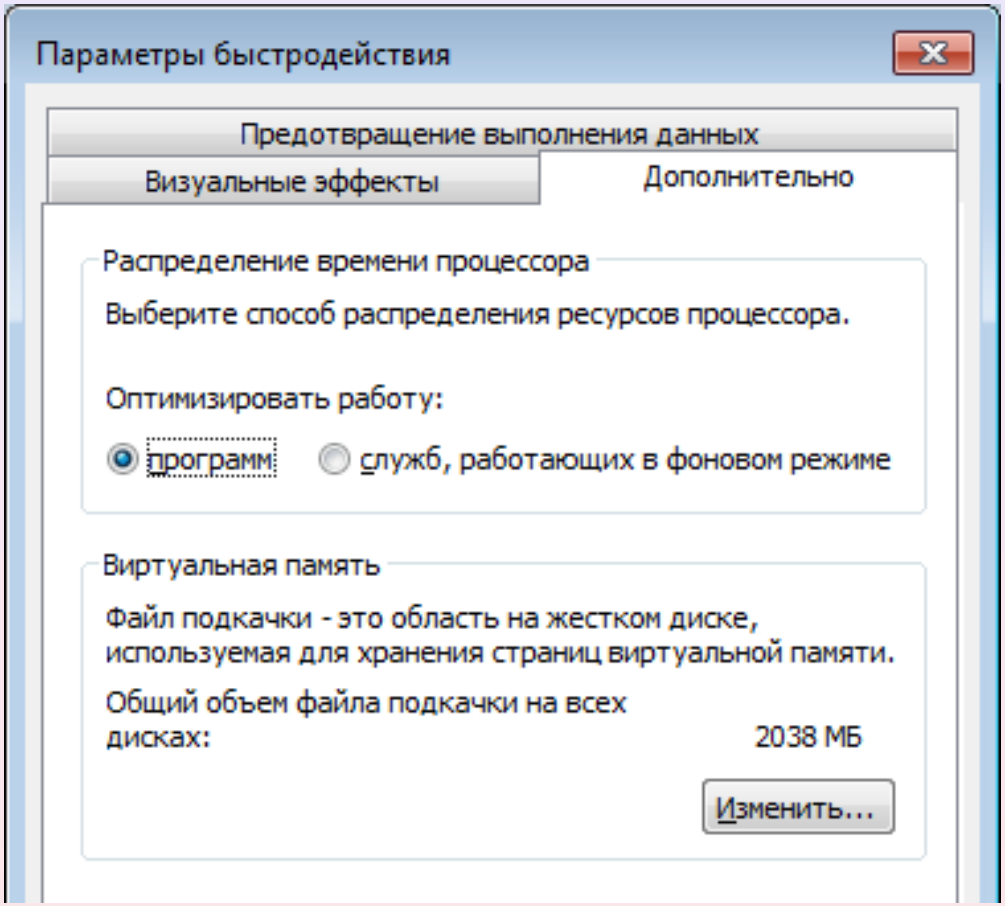

Рис. 4: выбор базового кванта времени (2 или 12 тактов таймера)

Вычисление кванта времени

количество тактов на 1 квант =

скорость процессора, Hz (тактов/секунду) $\times$ количество секунд на 1 квант

квантовая цель =

квантовое значение сброса $\times$ количество тактов на 1 квант

Изменение кванта времени

Правила изменения квантов

- При настройке «оптимизировать работу программ» кванты потоков процесса с активным окном, классом приоритета выше `IDLE_PRIORITY_CLASS` увеличиваются в 3 раза.

Вытеснение потока

Ситуации, приводящие к вытеснению

- Добровольное освобождение процессора (квант уменьшается на 1 для приоритетов < 14, для $\geqslant 14$ сбрасывается);
- Готовность к исполнению другого потока с более высоким приоритетом (вытесненный поток помещается в *начало* очереди готовых потоков $\sim$ приоритета);
- Исчерпание кванта времени (приоритет может снизиться, перемещается в конец очереди, состояние «выполняется» $\rightarrow$ «готов», вычисляется новый квант); нет потоков равного приоритета или выше $\Rightarrow$ потоку выдаётся следующий квант;
- Завершение потока.

Поток простоя

Выполнение потока в составе System Idle Process

- Выполняется на каждом процессоре.
- Выполняется только в отсутствие других потоков (не имеет приоритета).
- Выполняет некоторые системные функции (проверяет, выбран ли какой-либо поток для выполнения на данном процессоре и организует его диспетчеризацию, вызывает процедуру из HAL для простоя, включает/отключает прерывания, . . .)

Установка аффинности

Win32 SetProcessAffinityMask(), SetThreadAffinityMask()

```
BOOL WINAPI SetProcessAffinityMask(
  __in    HANDLE    hProcess,
  __in    DWORD_PTR dwProcessAffinityMask
);

DWORD_PTR WINAPI SetThreadAffinityMask(
  __in    HANDLE    hThread,
  __in    DWORD_PTR dwThreadAffinityMask
);
```

Планирование на многопроцессорных системах

Правила

- При наличии простаивающих процессоров выбирается в первую очередь идеальный, если невозможно — предыдущий, затем — тот, на котором выполняется планирование. Иначе — первый из простаивающих (с учётом Hyperthreading, NUMA).
- При отсутствии простаивающих процессоров проверяется приоритет выполняемого (running) или простаивающего (standby) потока на идеальном процессоре. Ниже $\Rightarrow$ вытеснение.
- Иначе поток помещается в очередь готовых (ready) $\sim$ приоритета на идеальном процессоре.
- Если очередь потоков для данного процессора пуста, запускается поток простоя, который проверяет очереди готовых потоков на других процессорах (с учётом Hyperthreading, NUMA).