


Управление ресурсами компьютера

Управление процессами и
потоками



Уровни многозадачности

- ◆ Однозадачные ОС
- ◆ Многозадачность на уровне процессов
- ◆ Многозадачность на уровне потоков
 - Потоки ядра
 - Пользовательские потоки

Процессы в современных ОС

◆ Структура процесса

- Идентификатор
- Адресное пространство
- Программа
- Данные
- Ресурсы

} – совместно используются потоками

◆ Потоки

- Работают в контексте процесса
- Имеют собственные данные (как минимум стек)

Процессы и потоки

Процессы:

- Пассивны
- Независимы
- Получают ресурсы в собственное использование
- Различные адресные пространства
- Взаимодействуют только через механизмы IPC
- Медленное переключение

Потоки:

- Активны (выполняются)
- Входят в процессы
- Разделяют ресурсы доступные процессу
- Одно адресное пространство
- Могут взаимодействовать через механизмы IPC
- Быстрое переключение

Потоки ядра (нити)

Имеют собственные:

- ◆ Идентификатор
- ◆ IP
- ◆ Стек
- ◆ Другие регистры
- ◆ Состояние
- ◆ Нити-потомки
- ◆ TLS

Разделяют:

- ◆ Адресное пространство
- ◆ Глобальные переменные
- ◆ Объекты и ресурсы ОС
- ◆ Статистическую информацию

Задачи ОС по управлению процессами и потоками

- ◆ Определить и реализовать основные свойства процессов и потоков
 - Структуры данных для хранения информации
 - Алгоритмы
- ◆ Определить, с какими объектами могут работать потоки (память)
- ◆ Управление ресурсами процессов и потоков
- ◆ Средства для создания/уничтожения/управления процессами и потоками
- ◆ Средства для разделения процессорного времени – алгоритмы планирования
- ◆ Средства синхронизации
- ◆ Механизмы защиты от взаимных блокировок
- ◆ Механизмы защиты процессов

Дескриптор процесса. РСВ. Контекст потока.

Дескриптор процесса

Общая информация о процессе:

- ◆ ID процесса
- ◆ Состояние процесса
- ◆ Ресурсы
 - Оперативная память
 - Открытые файлы
 - Сетевые соединения
 - ...
- ◆ Статистическая информация
- ◆ ...

Контекст потока (процесса)

Информация, необходимая для остановки и перезапуска потока

- ◆ Регистры CPU
- ◆ IP
- ◆ SP
- ◆ ...

Структуры управления процессами и потоками Windows

Объект ядра процесс

PCB

- Потоки
- Приоритет
- Время выполнения

Ресурсы и другое

- Таблица дескрипторов
- Виртуальная память
- Безопасность
- Отладка
- Статистика
- ...

Объект ядра поток

Контекст

- SP
- IP
- Другие регистры

Другие параметры

- Состояние
- Время создания
- Расположение стека
- Thread Environment Block
- Состояние
- Приоритет
- ...

Окружение

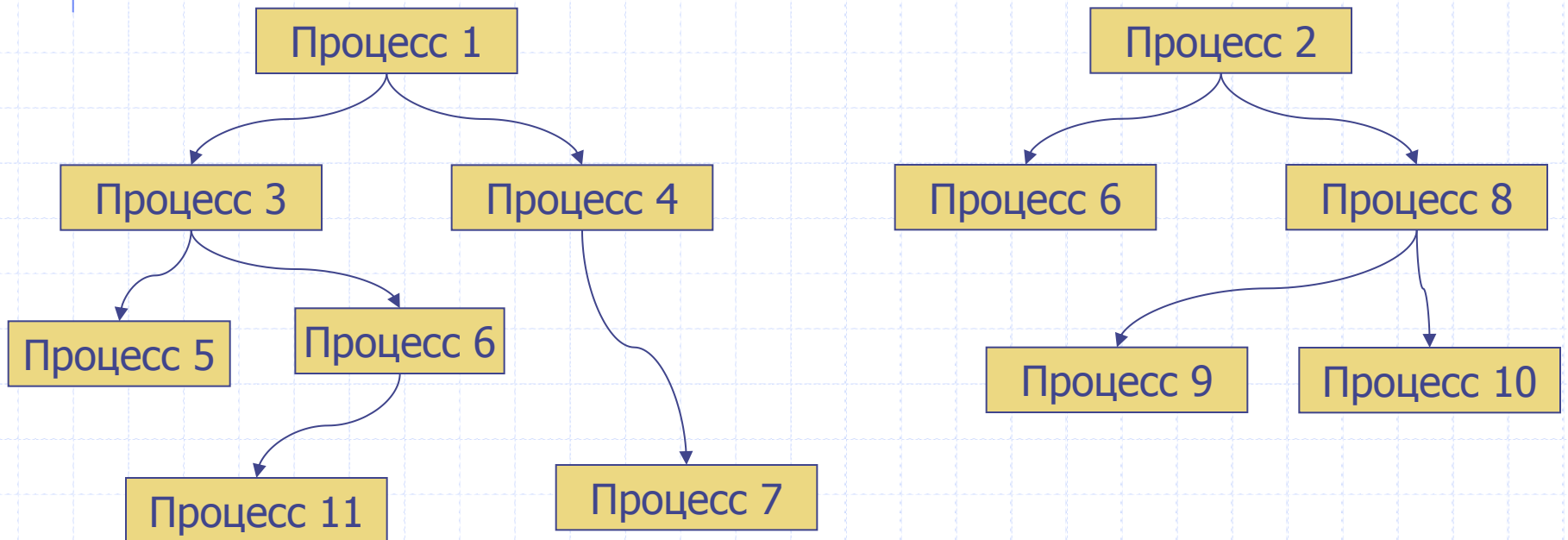
- TLS
- Исключения
- Критические секции
- ...

Создание процесса

- ◆ Когда происходит создание процессов?
 - При запуске системы
 - Динамически

Динамическое создание процесса

Процесс, инициировавший создание нового процесса называется процессом-родителем (parent process). Созданный процесс – ребёнком (child process).



Действия при создании процесса

- ◆ Создаётся PCB процесса
- ◆ Процесс получает уникальный идентификатор
- ◆ Выделяются ресурсы для нового процесса
 - Процесс может получить часть ресурсов от процесса-родителя и использует их совместно
 - Процесс получает новые ресурсы от ОС
- ◆ Устанавливаются начальные значения контекста процесса, включая регистры процессора
 - Дублируется контекст процесса-родителя (Unix, Windows)
 - Запускается новая программа (Windows, VAX/VMS)
- ◆ Состояние процесса изменяется на «готовность»

Создание процесса в Windows

```
BOOL CreateProcess(  
    LPCTSTR lpApplicationName, // имя выполняемого файла  
    LPCTSTR lpCommandLine,     // командная строка  
    LPSECURITY_ATTRIBUTES lpProcessAttributes,  
        // дескриптор защиты процесса  
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  
        // дескриптор защиты потока  
    BOOL bInheritHandles,  
        // режим наследования дескрипторов  
    DWORD dwCreationFlags,     // флаги  
    LPVOID lpEnvironment,     // блок переменных окружения  
    LPCTSTR lpCurrentDirectory, // текущая директория  
    LPSTARTUPINFO lpStartupInf,  
        // информация об окне приложения  
    LPPROCESS_INFORMATION lpProcessInformation  
        // информация о процессе и потоке  
);
```

Создание процесса в Unix

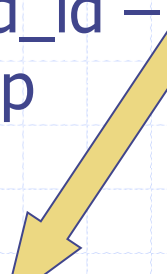
При создании нового процесса создаётся его копия, отличающаяся от родителя только следующим:

- ◆ Дочерний процесс имеет уникальный id
- ◆ Дочерний процесс имеет другой id родителя
- ◆ Дочерний процесс получает копию дескрипторов системных объектов, ссылающихся на те же объекты
- ◆ Счётчики использования ресурсов сбрасываются
- ◆ Все интервальные таймеры сбрасываются

Создание процесса в Unix

```
pid_t child_id;  
child_id=fork();
```

Родитель, `child_id` –
идентификатор
созданного
ребёнка



```
if(child_id>0)  
    printf("child created");  
else if(child_id==0)  
    printf("I am child");
```

Ребёнок,
`child_id = 0`



```
if(child_id>0)  
    printf("child created");  
else if(child_id==0)  
    printf("I am child");
```

Создание процесса в Unix

Родитель

```
pid_t child_id;  
char *argv[];  
char *env[];  
...  
if((child_id=fork())==0)  
    execve("child",argv,env);  
else if(child_id>0)  
    printf("child created");
```

Ребёнок

```
if((child_id=fork())==0)  
    execve("child",argv,env);  
//загрузка child  
int main(argc,argv,env) ...
```

Состояния потоков

◆ Выполнение

Поток обладает всеми необходимыми ресурсами и выполняется процессором.

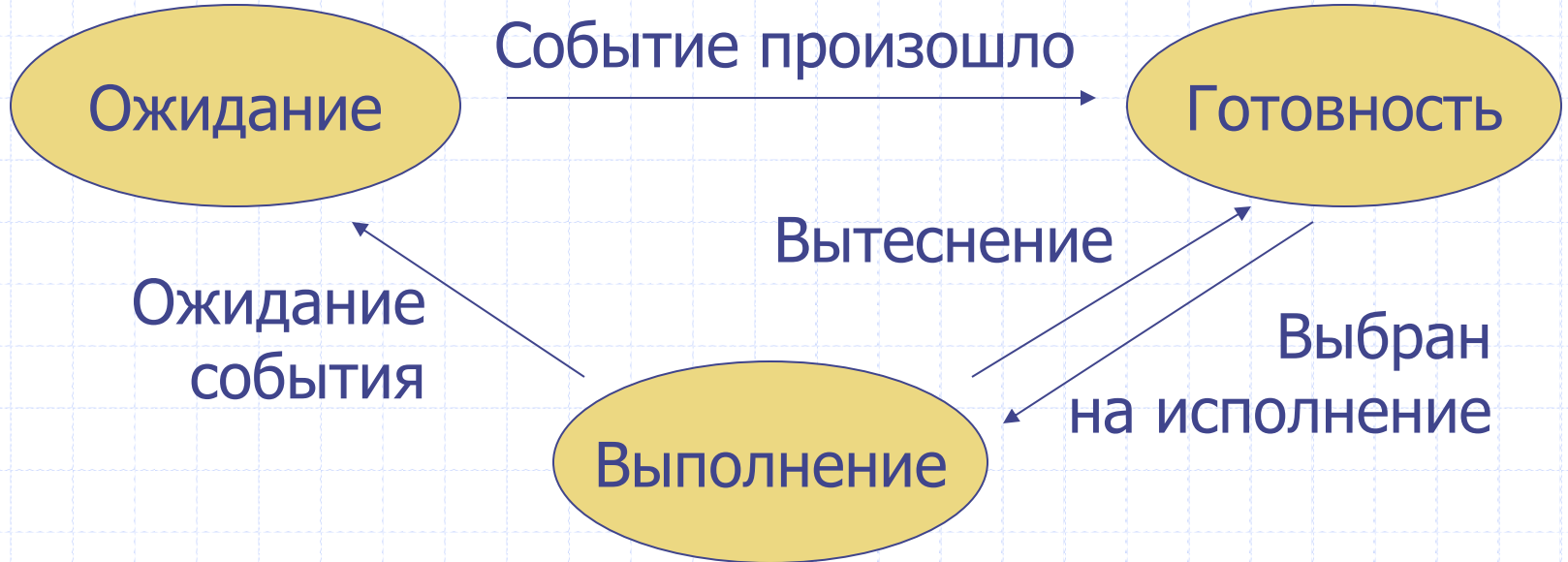
◆ Ожидание

Поток заблокирован по своим внутренним причинам. Его выполнение возможно после наступления какого-либо события.

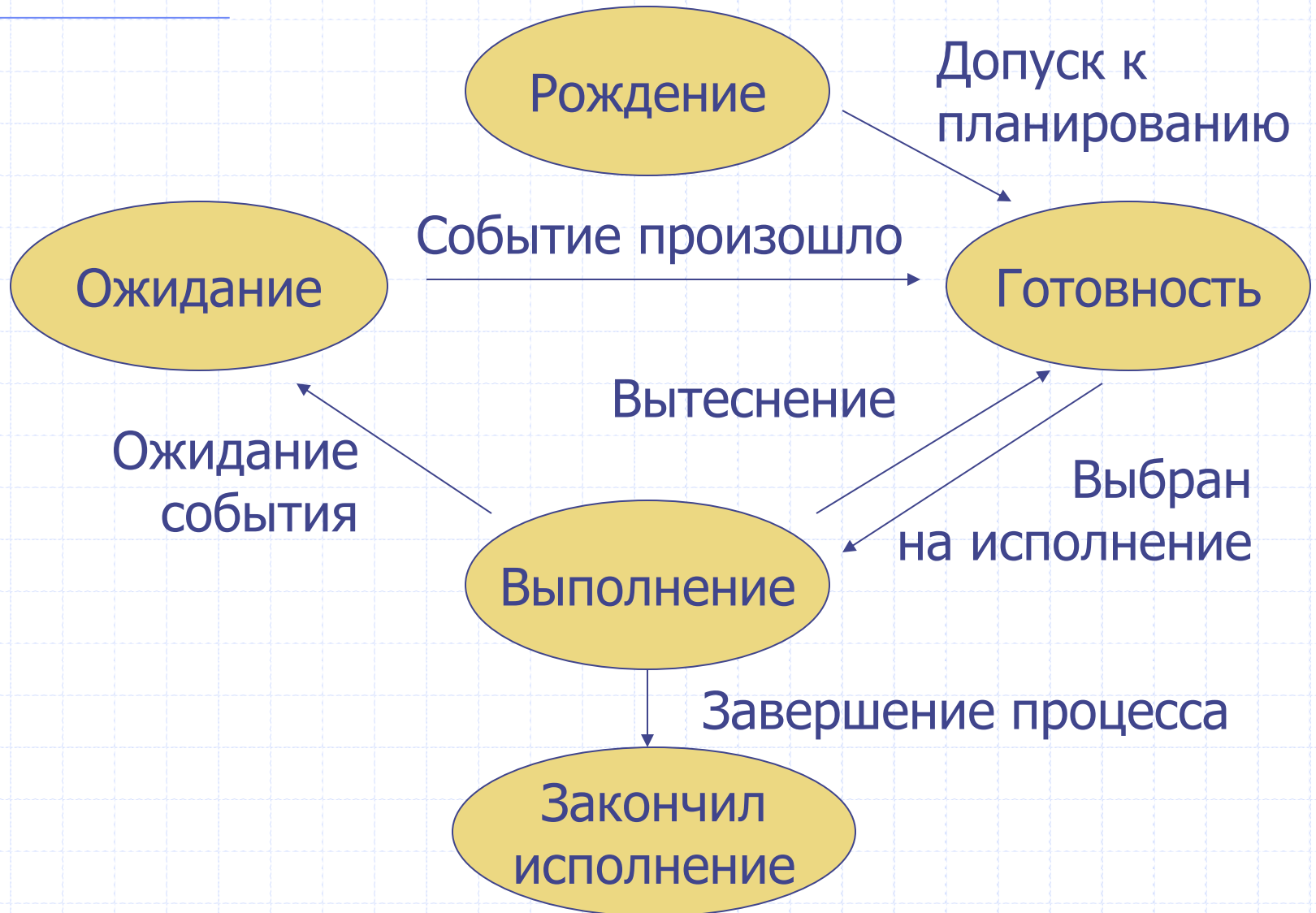
◆ ГОТОВНОСТЬ

Поток обладает всеми необходимыми ресурсами но в данный момент не выполняется (ожидает освобождения процессора).

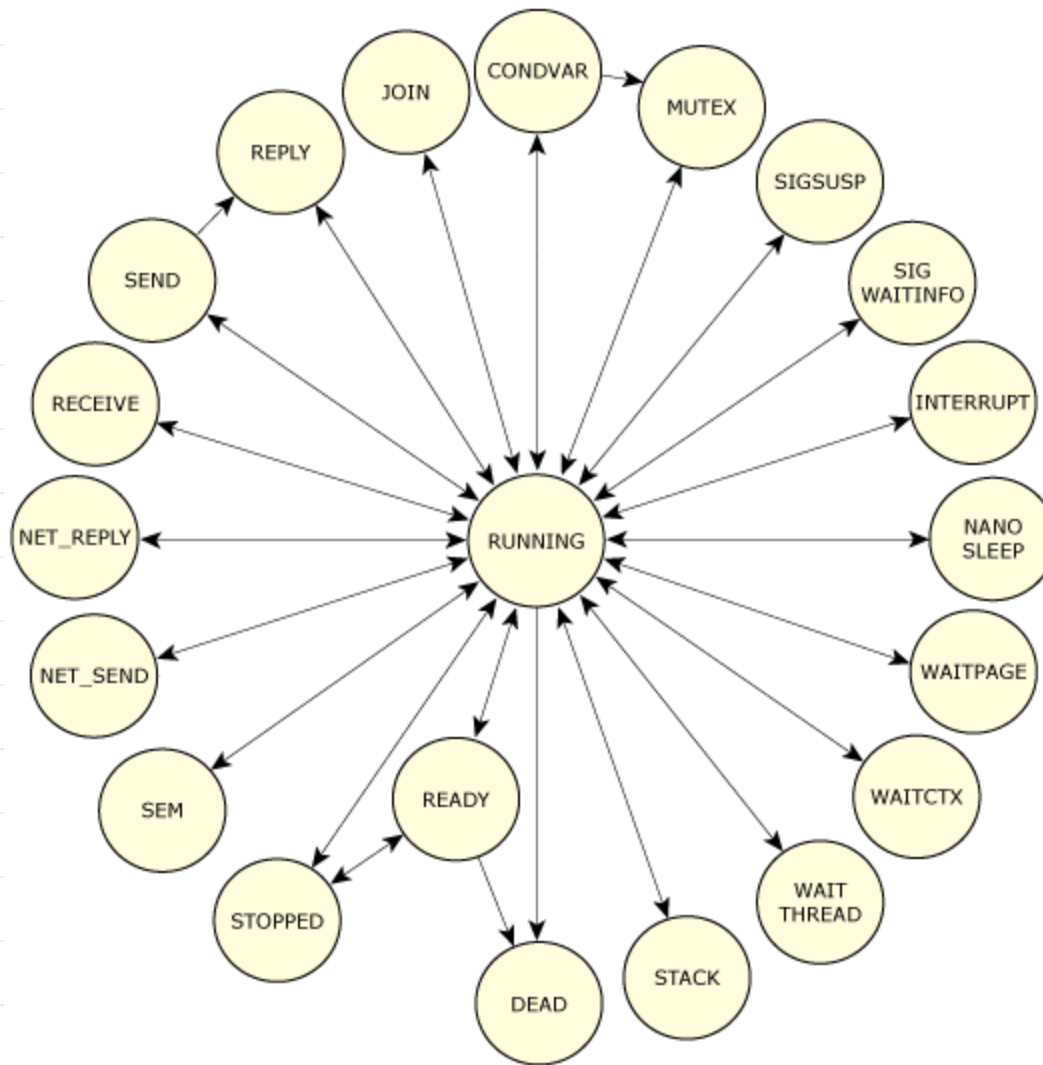
Переходы между состояниями



Переходы между состояниями

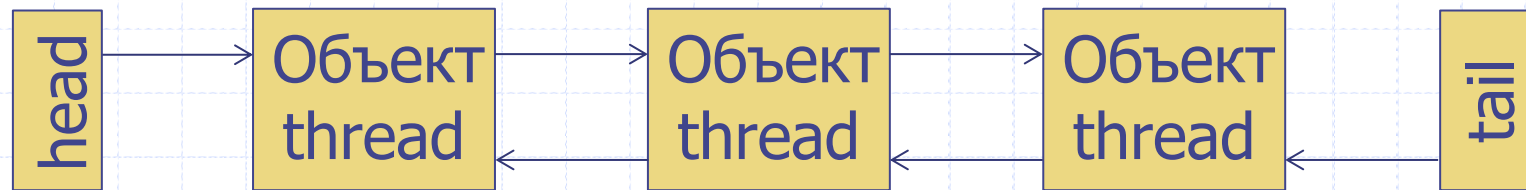


Состояния потока в QNX



Очереди потоков

- ◆ Очередь потоков в состоянии ГОТОВНОСТИ
 - По одной очереди на процессор
- ◆ Очереди потоков в состоянии ожидания
- ◆ Очередь завершённых потоков



Запуск потока на выполнение

- ◆ Выбирается один из потоков в состоянии ГОТОВНОСТЬ
- ◆ Обеспечивается наличие в оперативной памяти необходимой для выполнения информации
- ◆ Контекст потока восстанавливается из РСВ
- ◆ Состояние потока изменяется на выполнение
- ◆ Управление передаётся коду, на который указывает IP потока.

Приостановка (вытеснение) потока

- ◆ Происходит прерывание
- ◆ CPU сохраняет регистр IP и некоторые специальные регистры
- ◆ Запускается обработчик прерывания
- ◆ ОС сохраняет контекст потока
- ◆ Состояние потока изменяется на ГОТОВНОСТЬ

Блокирование потока

- ◆ Процесс обращается к ОС с помощью специального вызова
 - WaitFor...
 - Sleep()
 - Чтение/запись файлов, сокетов
 - ...
- ◆ ОС обрабатывает вызов
- ◆ Поток регистрируется в очереди ожидающих события
- ◆ Контекст потока сохраняется в РСВ
- ◆ Состояние потока изменяется на ожидание

Ожидание в многозадачных ОС

Никогда не делайте так

```
bool some_flag;  
...  
while(!some_flag);  
process();
```

Делайте так

```
HANDLE SyncObject;  
...  
WaitForSingleObject  
    (SyncObject, INFINITE);  
process();
```

Если никак не получается WaitFor...

```
bool some_flag;  
...  
while(!some_flag)  
    Sleep(300);  
process();
```

Надо быстрее и я знаю, что делаю

```
bool some_flag;  
HANDLE SyncObject;  
...  
for(int i=0; i<N; i++)  
    if(some_flag) break;  
if(!some_flag)  
    WaitForSingleObject  
        (SyncObject, INFINITE);  
process();
```

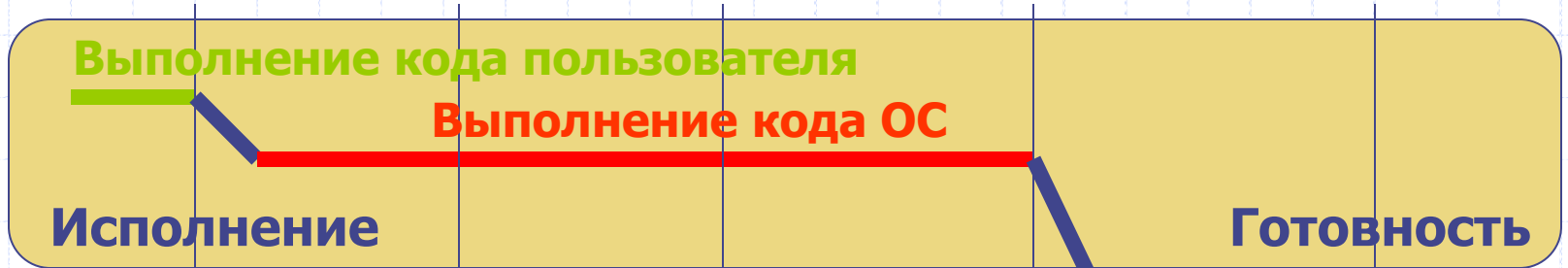
Готовый вариант - смотрите
InitializeCriticalSectionAndSpinCount()

Разблокирование потока

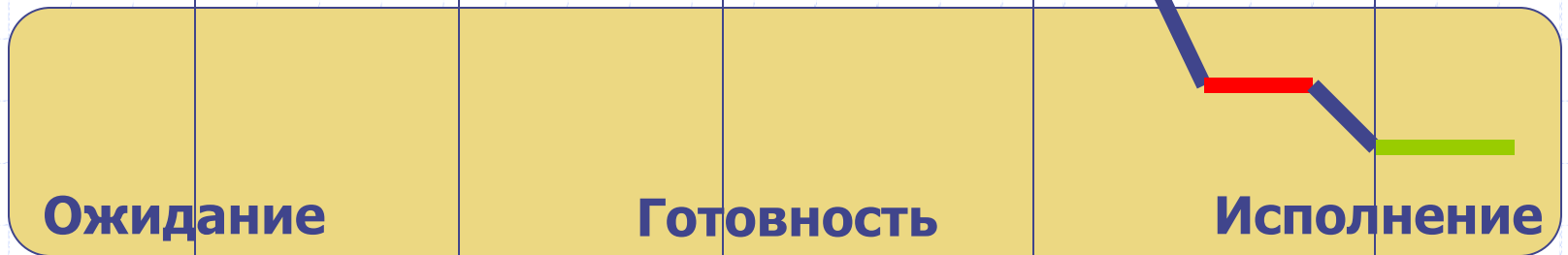
- ◆ Происходит событие
- ◆ ОС находит процесс в очереди ожидающих данного события
- ◆ Выполняются действия, связанные с наступлением события
- ◆ Состояние процесса изменяется на ГОТОВНОСТЬ

Переключение контекста

Поток 1



Поток 2



Работа hardware

Сохранение
контекста

Обработка
прерывания

Планирование

Восстановление
контекста

Прерывание

Завершение процесса

- ◆ Состояние процесса изменяется на «завершение работы»
- ◆ Освобождаются использованные процессом ресурсы
- ◆ PCB процесса может оставаться в системе
- ◆ Что происходит с дочерними процессами при смерти родительского?
 - Дети уничтожаются (VAX/VMS)
 - Дети усыновляются специальным системным процессом (Unix)
 - Дети становятся корнями новых деревьев (Windows)

Завершение потока в Windows

Причины

- ◆ функция потока возвращает управление
- ◆ поток вызывает функцию `ExitThread`:

```
VOID ExitThread(  
    DWORD dwExitCode    // код возврата  
);
```

- ◆ другой поток вызывает функцию `TerminateThread`:

```
BOOL TerminateThread(  
    HANDLE hThread,    // дескриптор потока  
    DWORD dwExitCode    // код возврата  
);
```

- ◆ завершается процесс

Завершение потока в Windows

Действия ОС

1. освобождается память, выделенная под стек потока;
2. отменяются начатые потоком операции ввода-вывода;
3. код возврата потока меняется со значения `STILL_ACTIVE (0x103)` на свое значение;
4. объект ядра «поток» переходит в свободное (signaled) состояние;
5. счетчик использования объекта «процесс» уменьшается на 1;
6. если это был последний поток в процессе, то процесс завершается.

Завершение процесса в Windows

Причины

- ◆ входная функция первичного потока (обычно это `main`, `wmain`, `WinMain` или `wWinMain`) возвращает управление;
- ◆ один из потоков процесса вызывает функцию `ExitProcess`;
- ◆ поток другого процесса вызывает функцию `TerminateProcess`;
- ◆ все потоки процесса умирают;
- ◆ пользователь выходит из системы.

Завершение процесса в Windows

Действия ОС

1. выполнение всех потоков в процессе прекращается;
2. все открытые процессом объекты USER и GDI уничтожаются, а объекты ядра закрываются;
3. Выделенная процессу память освобождается;
4. код возврата процесса меняется со значения `STILL_ACTIVE (0x103)` на свое значение;
5. объект ядра «процесс» переходит в свободное (signaled) состояние;
6. счетчик использования объекта «процесс» уменьшается на 1.

Пользовательские потоки

- ◆ Потоки ядра (kernel threads)
- ◆ Потоки пользователя (user threads)
 - Зелёные потоки (Green threads)
 - Волокна (fiber)

Потоки пользователя

Потоки могут быть реализованы с помощью пользовательских библиотек и работать без какого-либо вмешательства ОС.

- ◆ Небольшие накладные расходы на создание и переключение
- ◆ Пользователь может контролировать алгоритм планирования
- ◆ Как правило не могут использовать несколько ядер
- ◆ Один заблокированный поток может привести к блокировке остальных

Green threads

Зелёные потоки – потоки, управление которыми осуществляет виртуальная машина (Java 1.1, Ruby 1.9, Smalltalk,...)

Волокна (fiber)

Волокна – пользовательские потоки, работающие в режиме невытесняющей многозадачности.

- ◆ Самые низкие накладные расходы на создание и переключение.
- ◆ Отсутствие проблем с race condition.
- ◆ Как правило не могут использовать несколько ядер.
- ◆ Один заблокированный поток может привести к блокировке остальных.

Модели потоков

◆ 1:1 – потоки ядра

Все создаваемые потоки отображаются как объекты планирования ядра

◆ N:1 – потоки пользователя

Все потоки пользователя отображаются на один объект планирования ядра

◆ N:M – гибридные потоки

Обеспечивается выполнение потоков пользователя с использованием нескольких объектов планирования ядра