Процессы и потоки

LW_OS

Процесс
Создание процесса
Завершение процесса
Состояние процесса
Подходы

Предыдущая лекция

Следующая лекция: Синхронизация

Процессор переключается между программами, предоставляя каждой свое время. В каждый конкретный момент времени занят только одной программой. Но при этом создается иллюзия параллельного выполнения всех программ.

Процесс

Все многозадачные ОС используют модель процессов. Согласно этой модели выполняемые программы, организуются в виде процессов.

☝ **Процесс это абстрактное понятие, описывающее работу программы.**

☝ Процесс представляет собой экземпляр исполняемого приложения, который обладает независимым виртуальным адресным пространством, в котором могут размещаться данные, недоступные другим процессам

Для реализации используется ОС специальная таблица : ТАБЛИЦА процессов.

Каждый элемент такой таблицы содержит следующую информацию:

1. PID – идентификатор процесса.

2. ID пользователя от имени которого запущен процесс.
3. Значение сегментных регистров – сегмент данных, сегмент кода, сегмент стека – CS, DS и т.д.
4. Значение регистров данных процессора – AX аккумулятор, CX – регистр счетчика, DX – регистр данных.
5. Значение счетчика команд
6. Значение базового указателя
7. Состояние процесса
8. Приоритет процесса
9. Использованное процессорное время
10. Доп. Информация:

Восстановление с точки прерывания. По той информации, которая была сохранена в таблице (хранится в ядре).

Создание процесса

При загрузке ОС создается несколько системных процессов. Кроме того, уже созданные процессы могут создавать новые процессы. Для этого созданный процесс выполняет специальный системный вызов, который заставляет ОС создать новый процесс. В различных ОС для создания новых процессов используются различные подходы.

В ОС linux, solari, IOS используется подход предложенный в стандарте POSIX (переносимый интерфейс ОС UNIX).

Согласно этому стандарту с помощью функции fork, которая создает копию вызывающего процесса. В дальнейшем созданный процесс может продолжить выполнять ту же программу или же вызывать функцию exec для загрузки и выполнения новой программы.

☝️ Главный поток начинается с точки входа (WinMain-CRTStartup, wWinMainCRTStartup, mainCRTStartup или wmainCRTStartup), в которой выполняется инициализация различных библиотек C/C++, загрузка необходимых DLL и создание глобальных переменные. Когда все это будет сделано, вызывается главная функция приложения (WinMain, wWinMain, main или wmain).

Завершение процесса

Завершается в одно из:

1. Завершения работы и выполнения спец. вызов, который завершает процесс: ExitProcess, exit
2. Произошла неустранимая ошибка в результате которой программа более не может исполнять свою работу.

В MW преимущество отдается другой функции **CreateProcess**, которая создает вызывает новую программу. При это создается новое адресное пространство.

1. Другой процесс выполнил системный вызов, в результате которого был принудительно завершен указанный процесс. Радикально: В POSIX – kill. В Win TerminateProcess. При завершении процесса удаляется созданное для него адресное пространство, однако ОС не сразу удаляет из таблицы запись о совершенном процессе Благодаря этого какое-то время все еще можно узнать информацию о операции: сколько процессорного времени.

(!) Если программа завершается совсем, но не возвращаются дескрипторы.

Состояние процесса

Готовность - процесс приостановлен, но может быть запущен как только освободиться процессор.

Ожидание - процесс приостановлен, пока не произойдет вызов.

Завершение - процесс завершен, но еще не удален из таблицы процессов

Переходы из исполнения в готовность осуществляются компонентом, называемым планировщиком, если тот решил, что пора приостановить процесс и предоставить процессор другому процессу

- Переход из готовности в исполнение, если решил снова вернуть процессор процессу.
- Переход из исполнения в ожидания происходит когда продолжение процесса невозможно пока не произойдет определенное события.
- Переход из ожидания в готовность, когда происходит событие
- Переход из исполнения в завершение - при достижении точки завершения, или при возникновении неустранимой ошибки, или принудительно другим процессом.

> *Example: ожидание ввода с клавиатуры, ожидание пока прочитается файл*

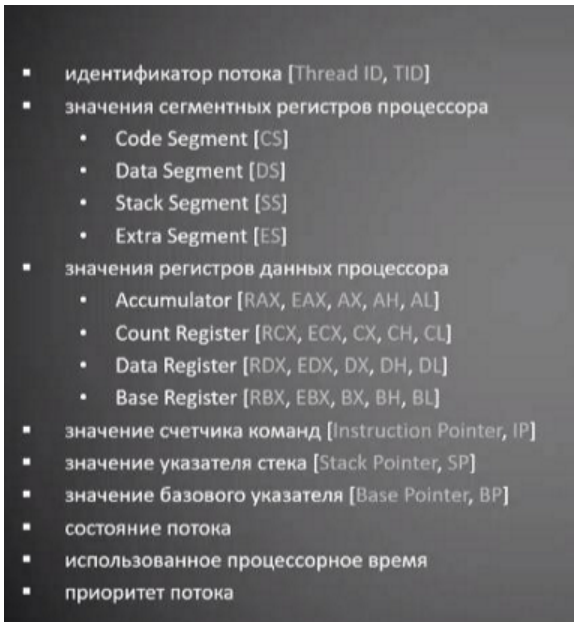
При создании процесса требуются значительные ресурсы адресного пространства.

Вместе с процессами используются потоки.

Поток исполняя свой код, расходует меньше ресурсов чем процесс, и этим позволяет сэкономить на ресурсах.

Поток исполняется внутри процесса, и при этом используется общее адресное пространство.

Сведения о потоках хранятся в специальной таблице. Таблица дескрипторов создается при инициализации процесса, и изначально она пуста. Но стоит одному из потоков процесса вызвать какую-нибудь функцию, создающую объект ядра, как тут же в таблицу дескрипторов данного процесса будет добавлена соответствующая запись



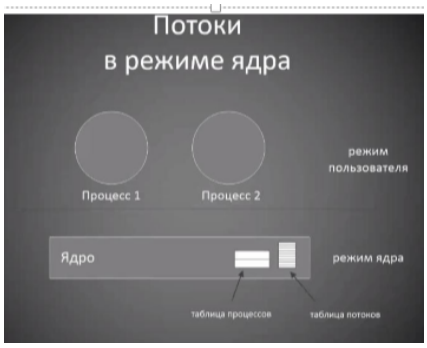
Содержимое таблицы потоков

Подходы

1. В пользовательском режиме

   Высокая эффективность

   - управление потоком осуществляется внутри процесса
   - ядро ничего не знает о потоках и управляет только процессами и знает только о них



2. Режим ядра

   Простая реализация. Для общих задач.

   - поток (?располагается) в ядре, управление там же

☝️ Каждый процесс может управлять своим потоком

Операционная система:

- организует очередь
- каждый поток обрабатывает только 1 запрос из поступающих 1 млн
- те, что не попали под обработку - ждут.

ПОТОКИ.pdf