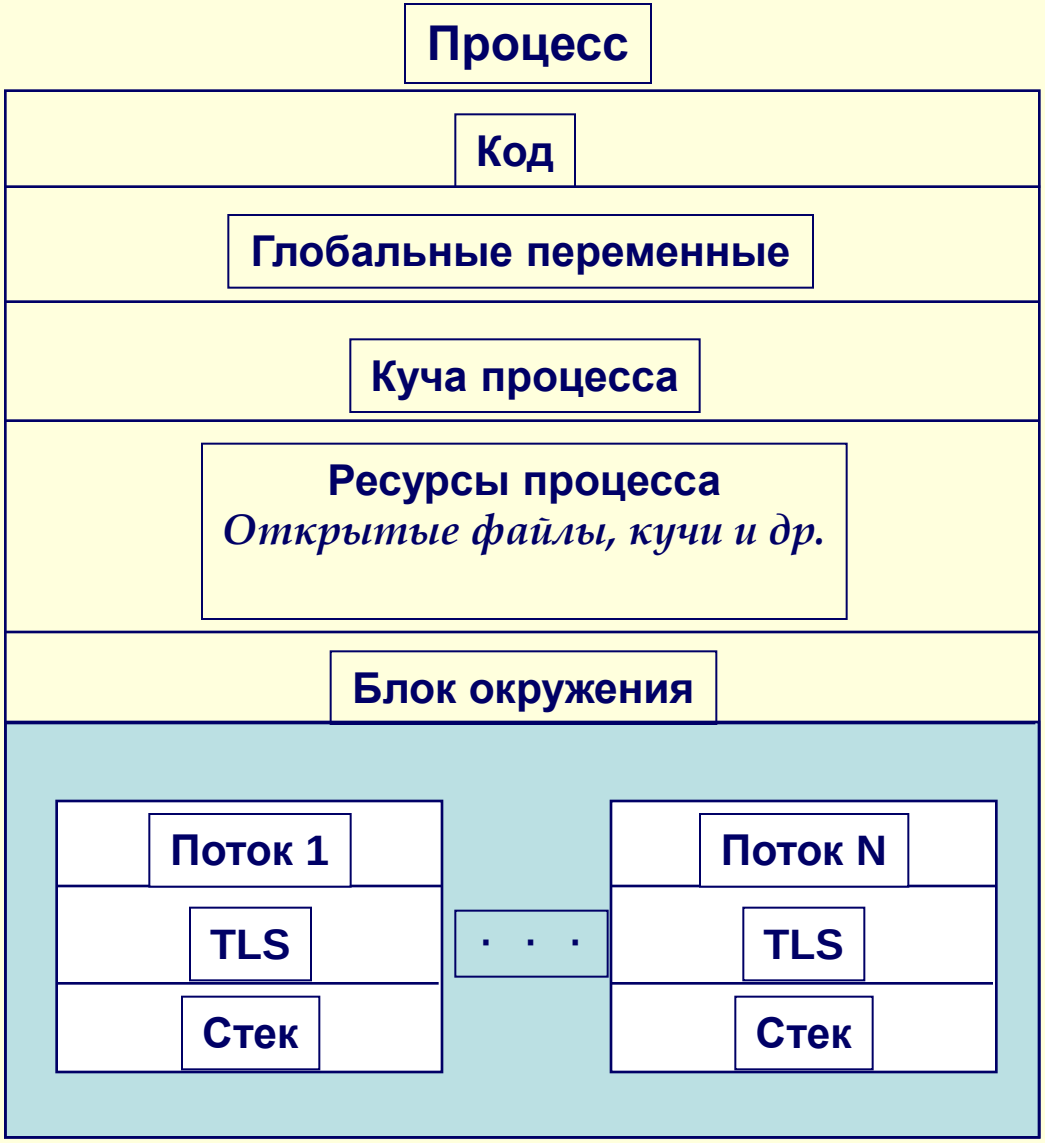
Организация процессов в операционных системах

Для слушателей переподготовки по специальности

"Программное обеспечение информационных систем "

Лк1, Лк2

Структура процесса



Создание процесса (1 из 8)

```
BOOL CreateProcess (
    LPCTSTR lpImageName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpsaProcess,
    LPSECURITY_ATTRIBUTES lpsaThread,
    BOOL bInheritHandles,
    DWORD dwCreate,
    LPVOID lpvEnvironment,
    LPCTSTR lpCurDir,
    LPSTARTUPINFO lpsiStartInfo,
    LPPROCESS_INFORMATION lppiProcInfo
    )
```

- Возвращает **TRUE** если процесс и поток успешно созданы.

Создание процесса(2 из 8)

- Параметры:
- **`lpImageName`** — специфицирует исполнямый файл
- **`lpCommandLine`** — указатель на аргументы
- **`lpsaProcess`** — указатель на структуру атрибутов безопасности объекта “процесс”
- **`lpsaThread`** — указатель на структуру атрибутов безопасности объекта “поток”
  - (**`NULL`** предполагает умолчательный уровень безопасности)

Создание процесса(3 из 8)

- **`bInheritHandles`** — флаг, указывающий, что новый процесс может наследовать дескрипторы
  - Индивидуально каждый дескриптор может быть помечен как ненаследуемый
  - Обычное использование – перенаправление стандартного ввода/вывода

Создание процесса (4 из 8),

- **dwCreate** — комбинация флагов - опций создания процесса :
  - **CREATE_SUSPENDED** — Первичный поток создается в приостановленном состоянии. Продолжение работы после вызова **ResumeThread**
  - **DETACHED_PROCESS** — создается процесс, “отвязанный” от консоли родительского процесса (без консоли)
  - **CREATE_NEW_CONSOLE** — новый процесс создается с новой консолью (этот и предыдущий - два взаимоисключающих флага.)
  - **CREATE_NEW_PROCESS_GROUP** — создаваемый процесс начинает новую группу процессов
  - **и др.**

Замечание Некоторые флаги являются взаимо-исключающими

Создание процесса(5 из 8)

- `lpvEnvironment` — указатель на блок окружения процесса. Если `NULL`, то наследуется родительское окружение
- `lpCurDir` — указатель на строку, определяющую текущую директорию процесса. Если `NULL`, то наследуется родительская директория
- `lpsiStartInfo` — указатель на набор флагов, определяющих некоторые стартовые параметры процесса
- `lppiProcInfo` — структура, куда записываются дескрипторы и идентификаторы созданных объектов “процесс” и “поток”

Создание процесса (6 из 8)

структура **PROCESS_INFORMATION**

```
typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
    } PROCESS_INFORMATION;
```

Создание процесса (7 of 8)

структура **STARTUPINFO**

```
typedef struct tagSTARTUPINFO {
    DWORD  cb;          // размер структуры
    LPTSTR lpReserved;// установить в NULL
    LPTSTR lpDesktop; // имя рабочего стола
    LPTSTR lpTitle;   // и заголовок для консоли
    DWORD  dwX;         // положение (x,y)
    DWORD  dwY;
    DWORD  dwXSize;   // и размер окна(cx,cy)
    DWORD  dwYSize;
    DWORD  dwXCountChars;    // размер консоль-
    DWORD  dwYCountChars;    // ного окна (симв)
    DWORD  dwFillAttribute; //цвет фона конс.
    DWORD  dwFlags;   // маска доступности полей
    WORD   wShowWindow;
    WORD   cbReserved2;// установить в 0
    LPBYTE lpReserved2;// установить в NULL
    HANDLE hStdInput;
    HANDLE hStdOutput;
    HANDLE hStdError;
    } STARTUPINFO, *LPSTARTUPINFO;
```

Создание процесса(8 из 8)

- // флаги для поля `dwFlags`
- STARTF_FORCEONFEEDBACK
- STARTF_FORCEOFFFEEDBACK
- STARTF_RUNFULLSCREEN
- STARTF_USECOUNTCHARS
- STARTF_USEFILLATTRIBUTE
- STARTF_USEPOSITION
- STARTF_USESHOWWINDOW
- STARTF_USESIZE
- STARTF_USESTDHANDLES

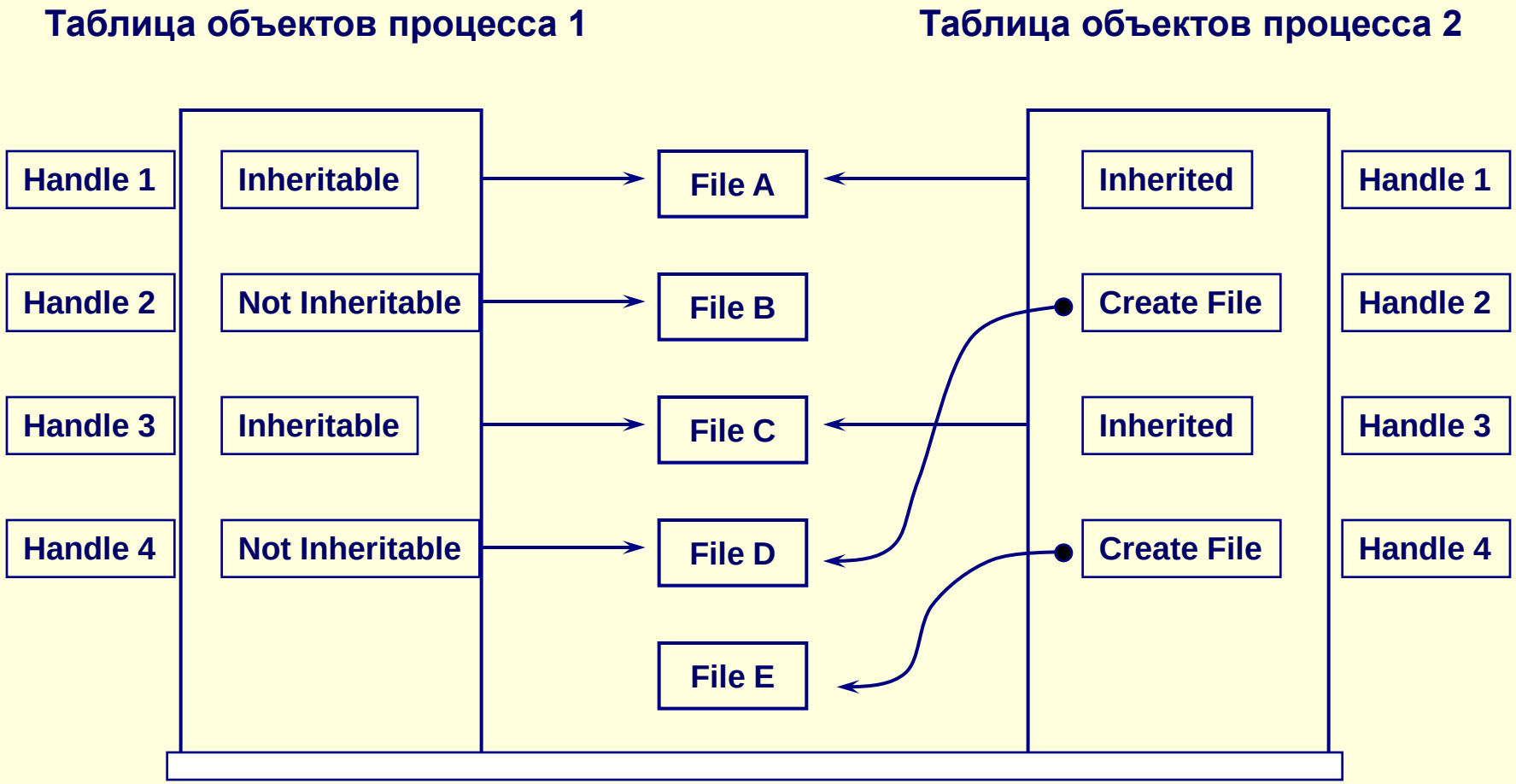
Наследуемые дескрипторы (1 из 2)

```
typedef struct SECURITY_ATTRIBUTES {
    DWORD nLength;
    LPVOID lpSecurityDescriptor;
    BOOL bInheritHandle;
} SECURITY_ATTRIBUTES;
```

DUPLICATING HANDLES (1 из 3)

- **BOOL DuplicateHandle (HANDLE hSourceProcess,**
- **HANDLE hSource, HANDLE hTargetProcess,**
- **LPHANDLE lphTarget, DWORD dwAccess,**
- **BOOL fInherit, DWORD dwOptions)**

Совместное использование объектов



Идентификация процесса(1 of 4)

- `HANDLE GetCurrentProcess (VOID)`
  - Возврат : “pseudo handle” , который не наследуется
  - Используется когда процессу нужен собственный дескриптор
- `DWORD GetCurrentProcessId (VOID)`
  - Возврат: ID текущего процесса

Идентификация процесса(2 of 4)

- HANDLE OpenProcess (DWORD dwAccess,
- BOOL fInherit,
- DWORD IDProcess)

- Возврат : handle процесса или NULL

Идентификация процесса (3 of 4)

- Параметры
- `dwAccess`
  - См. след. слайд
- `fInherit`
  - Указывает является ли новый дескриптор наследуемым
- `IDProcess`
  - Идентификатор процесса, дескриптор которого запрашивается

Идентификация процесса(4 of 4)

- dwAccess — Определяет допустимые для дескриптора (handle) операции. Возможные значения:
  - SYNCHRONIZE
    - Возможно ожидание другими процессами завершения данного процесса
  - PROCESS_ALL_ACCESS
    - Все флаги режимов доступа установлены
  - PROCESS_TERMINATE
    - Возможно завершение процесса через вызов TerminateProcess
  - PROCESS_QUERY_INFORMATION
    - Дескриптор может быть использован GetExitCodeProcess и GetPriorityClass для получения информации о процессе

Ожидание завершения и синхронизация

Выход из процесса

- VOID ExitProcess (UINT nExitCode)

- BOOL GetExitCodeProcess (HANDLE hProcess,
- LPDWORD lpdwExitCode)

- Для десриптора процесса hProcess должен быть установлен флаг режима доступа PROCESS_QUERY_INFORMATION
- lpdwExitCode указатель на DWORD, куда будет возвращено значение кода завершения
  - Значение STILL_ACTIVE будет возвращено, если процесс еще не завершился

Завершение процесса

- `BOOL TerminateProcess (HANDLE hProcess,`
- `    UINT uExitCode)`

  - **Для десриптора процесса** hProcess **должен быть установлен флаг режима доступа** `PROCESS_TERMINATE`
  - **Фунция завершения определяет код завершения(** `exit code`)
  - **Перед завершением процесса необходимо позаботиться об освобождении им занятых ресурсов, которые могут использоваться другими процессами**
  - **Для процесса, завершаемого через** TerminateProcess **не выполняется структурная обработка исключений (** SEH)

Ожидание завершения процесса (1)

- **Функции синхронизации с другими объектами.**
  - **Специальные функции обеспечивают разные варианты ожидания на одиночных и множественных объектах**
  - Wait for: // **Ожидание**
    - A single object //одного объекта
    - The first of several specified objects // первого из множества
    - All objects in a specified group //все объекты множества
  - **Возможно указание допустимого времени ожидания** (timeout)
  - **Следующие функции могут быть использованы для обеспечения синхронизации на различных объектах (в том числе и на объектах типа “Процес”):**

Ожидание завершения процесса(2)

Функции ожидания

- **DWORD WaitForSingleObject (**
  HANDLE **hObject,**
  DWORD **dwTimeOut)**

- **DWORD WaitForMultipleObjects (**
  DWORD **cObjects,**
  LPHANDLE **lphObjects,**
  BOOL **fWaitAll,**
  DWORD **dwTimeOut)**

Ожидание завершения процесса(3)

- **Параметры**:
  - **`hObject`** – дескриптор одиночного объекта или
  - **`lphObjects`** – указатель на массив из **`cObjects`** десрипторов
  - **`cObjects`** не должен превышать MAXIMUM_WAIT_OBJECTS
  - **`dwTimeOut`** (в миллисекундах). Если значение параметра **`0`** – функция проверяет состояние объекта и возвращает управление немедленно. Если **`INFINITE`** - ожидание без ограничения по временит)
- **Возвращаемое значение**:
  - причина завершения ожидания

  или
  - **`0xFFFFFFFF`** в случае ошибки (используйте **`GetLastError`** для получения подробной информации)

Ожидание завершения процесса(4), причины завершения ожидания

- **Значение TRUE для fWaitAll определяет ожидание всех объектов.**
- **Возможные возвращаемые значения:**
  - **WAIT_OBJECT_0 — объект освободился (для WaitForSingleObject или WaitForMultipleObjects со значением fWaitAll равным TRUE)**
  - **WAIT_OBJECT_0 + n, где 0 <= n < cObjects**

  (вычитаем WAIT_OBJECT_0 из возвращаемого значения для определения номера объекта в массиве в случае ожидания освобождения любого объекта из множества)
  - **WAIT_TIMEOUT — выход из ожидания по причине истечения допустимого времени ожидания**
  - **WAIT_ABANDONED — только для бъекта типа “мьютекс”**

Код завершения процесса

- **`GetExitCodeProcess`**
  - Определяет код завершения процесса

Окружение процесса(Environments) и безопасность(Security)

Блок окружения и строки окружения

- Блок окружения содержит множество строк вида:
- **Name = Value**

  - Каждая строка блока окружения завершается нулевым байтом(**NULL-terminated**)
  - Для передачи дочернему процессу окружения родительского процесса устанавливаем `lpvEnvironment` в значение `NULL`
  - Каждый процесс может опрашивать или изменять свои переменные окружения, удалять либо добавлять новые

Блок окружения и строки окружения(1 of 2)

- **DWORD GetEnvironmentVariable (LPCTSTR lpName,**
- **LPTSTR lpValue,**
- **DWORD cchValue)**

- **BOOL SetEnvironmentVariable (LPCTSTR lpName,**
- **LPCTSTR lpValue)**

Блок окружения и строки окружения(2 of 2)

- **`lpName`** — имя переменной
  - Переменна будет добавлена в блок если она отсутсвует и ее значение не **`NULL`**
  - Если значение переменной **`NULL`**, то переменная удаляется из блока окружения
  - Символ “**`=`**” не должен появляться в части value строки блока
- **`GetEnvironmentVariable`** — возвращает длинну строки (0 в случае ошибки)
  - Если **`lpValue`** указывает на недостаточное место в памяти(размер указан в **`cchValue`**), то возвращаемое значение равно требуемому размеру полной строки

Безопасность процесса

- Обычно, **`CreateProcess`** создает объект ядра процесс с правами доступа **`PROCESS_ALL_ACCESS`**
- Другие режимы доступа:
  - **`PROCESS_TERMINATE`**
  - **`CREATE_THREAD`**
  - **`CREATE_PROCESS`**
  - **`DUPLICATE_HANDLE`**
  - **`PROCESS_SET_INFORMATION`**
  - **`PROCESS_QUERY_INFORMATION`**