Curso de C/C++ Avançado

Aula 10 - Threads

Allan Lima – http://allanlima.wordpress.com

- *Você pode:*
  - copiar, distribuir, exibir e executar a obra
  - criar obras derivadas
  - fazer uso comercial da obra
- *Sob as seguintes condições:*
  - **Atribuição**. Você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante.
  - **Compartilhamento pela mesma Licença**. Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.
  - Para cada novo uso ou distribuição, você deve deixar claro para outros os termos da licença desta obra.
  - Qualquer uma destas condições podem ser renunciadas, desde que Você obtenha permissão do autor.
- ***Veja aqui a licença completa***

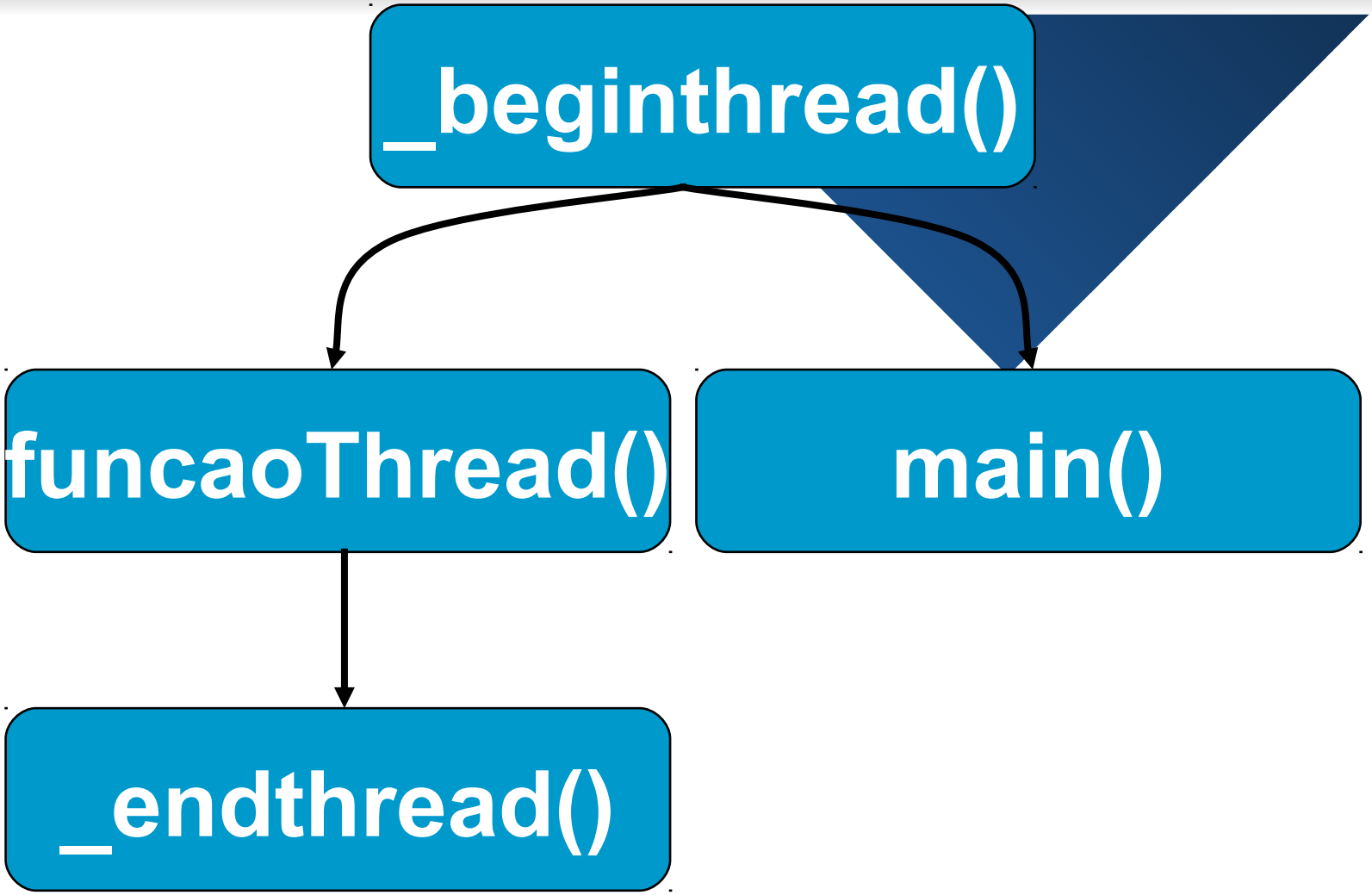
Como criar um thread?

- *Existem várias maneiras de se criar um thread:*
  - Usando a API do Windows
  - Usando a MFC
  - Usando uma alguma biblioteca implementada por terceiros
- *Qual a melhor ???*
  - Depende do seu problema

Usando a API do Windows

- *A API do windows possui diversas funções para a criação e destrução de threads:*
  - _beginthread
  - _beginthreadex
  - _endthread
  - _endthreadex
- *Todas estas funções estão definidas no header process.h*

Usando a API do Windows



_beginthread

- *uintptr_t _beginthread(*

  *void(__cdecl *start_address) (void *), unsigned stack_size,*

  *void *arglist*

);

- *Cria um thread*

_beginthread

- *start_address é a função que será executada pelo thread*
- *stack_size é o tamanho da pilha que será usada pelo thread*
- *arglist é um array com os parâmetros que serão passados para start_address*

_beginthreadex

```
uintptr_t _beginthreadex(
        void *security,
        unsigned stack_size,
        unsigned (
                __stdcall *start_address) (void
  *),           void *arglist,
        unsigned initflag,
        unsigned *thrdaddr
);
```

_beginthreadex

- *security*
- *start_address é a função que será executada pelo thread*
- *stack_size é o tamanho da pilha que será usada pelo thread*
- *arglist é um array com os parâmetros que serão passados para start_address*
- *initflag é o estado inicial do novo thread*
- *thrdaddr é variável de 32 bits que irá receber o descritor do thread*

Destruindo um thread

- *void _endthread(void);*
  - Encerra um thread criado pela função _beginthread
- *void _endthreadex(unsigned retval);*
  - Encerra um thread criado pela função _beginthreadex

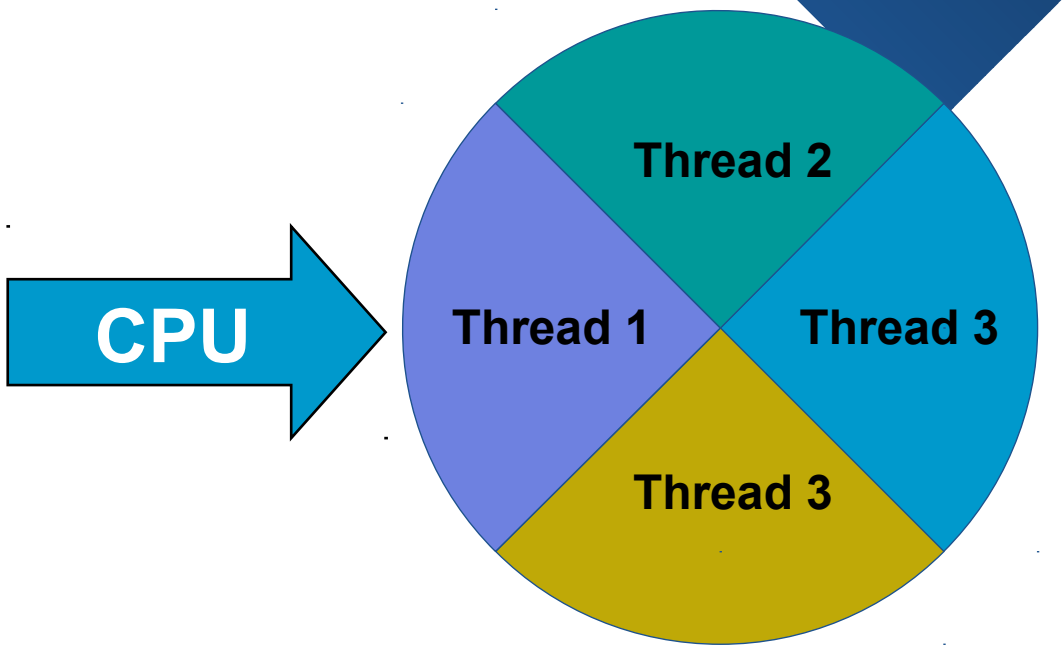
- *Estas duas funções são chamadas automaticamente quando função do thread é encerrada*

Exemplos

- *exemploThreadSimples.cpp*
- *exemploThreadSimplesEx.cpp*

Problema

- *A sincronização dos threads é sempre um problema*



Exemplo

- *exemploProblemaSincronizacao.cpp*

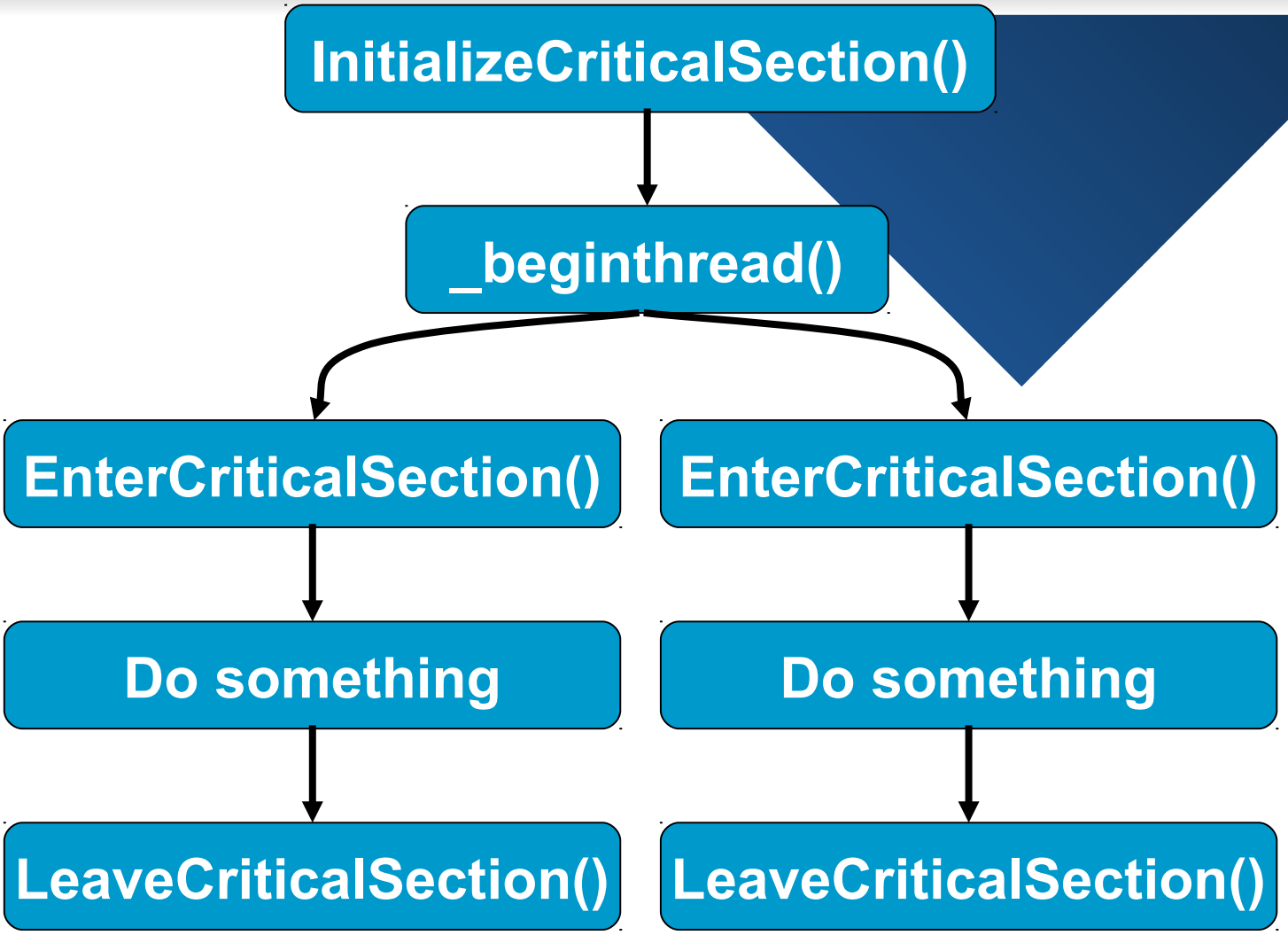
Solução

- *Existem diversas técnicas para sincronizar os threads:*
  - Critical Section Objects
  - Mutex Objects
  - Eventos
  - ...
- *Mas qual a melhor?*
  - Depende do seu problema

Critical Section Objects

- *Permitem a sincronização de forma simples, prática e rápida*
- *Quando uma função entra em uma sessão crítica ela não perde a prioridade na CPU*
- *Restrições:*
  - Só podem ser usadas por threads em um mesmo processo
  - Só podem atender a um thread por vez

Critical Section Objects



Critical Section Objects

- *void InitializeCriticalSection(LPCRITICAL_SECTION lpCS);*
  - Inicializa um sessão crítica
- *void EnterCriticalSection(LPCRITICAL_SECTION lp);*
  - Entra em uma sessão crítica
- *void LeaveCriticalSection(LPCRITICAL_SECTION lp);*
  - Sai em uma sessão crítica

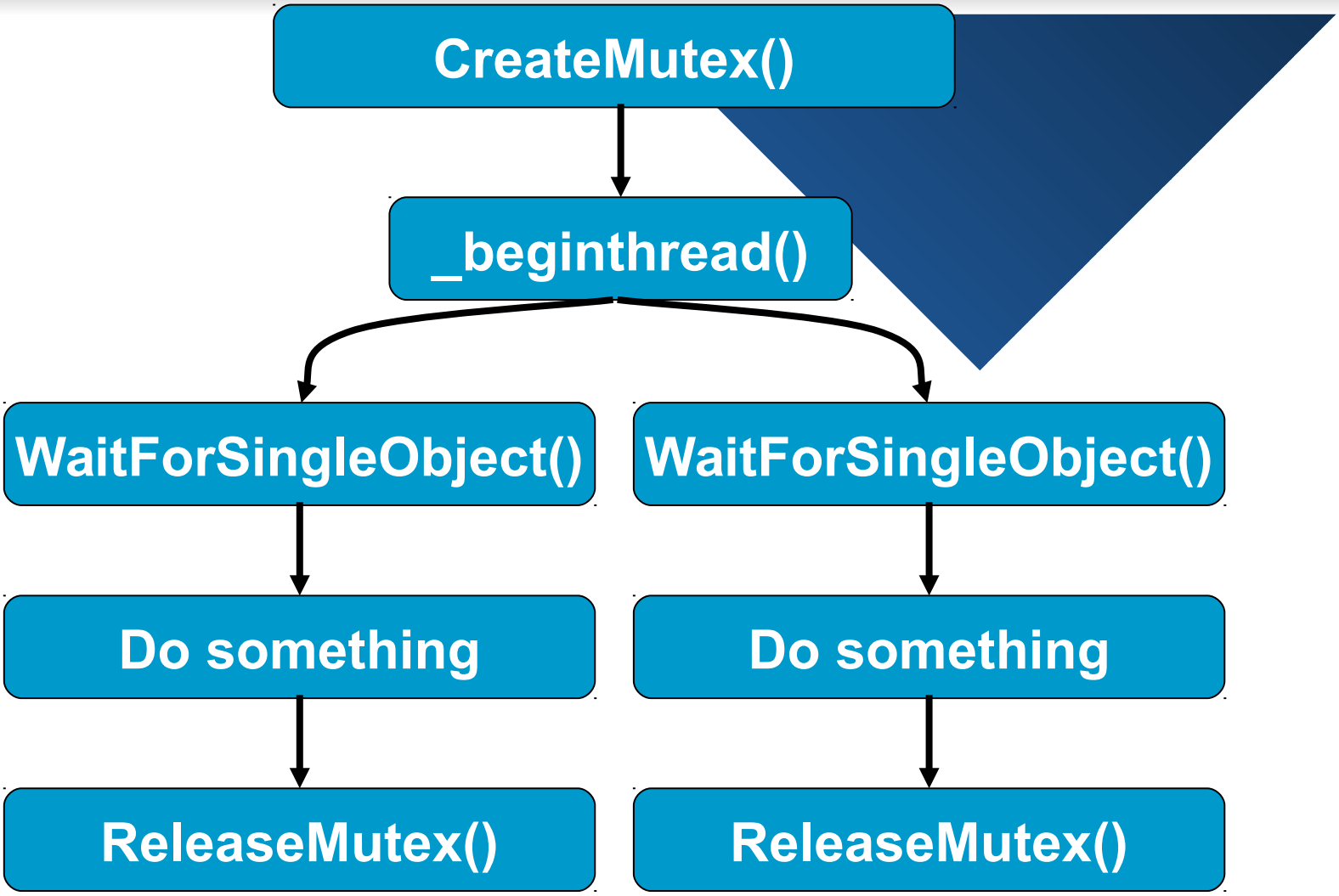
Exemplo

- *exemploThreadSincronizacaoSessao.cpp*

Mutex Objects

- *Permite a sincronização de threads em processos distintos*
- *É uma técnica mais lenta do que Critical Section Objects*
- *Múltiplos processos compartilham apenas um Mutex Object*
- *Com ele também podemos determinar quanto tempo iremos "aguardar"*

Mutex Objects



Mutex Objects

- *HANDLE CreateMutex(*

  LPSECURITY_ATTRIBUTES lpMutAtt, BOOL bInitialOwner,

  LPCTSTR lpName

);

  - Cria um Mutex Object

Mutex Objects

- *lpMutAtt determina se o id será herdado por processos filhos*
- *bInitialOwner indica se o mutex object será criado ou se ele já existe*
- *lpName é nome que o Mutex Object terá*
- *Retorno:*
  - A função retorna o id do objeto criado
  - Se o nome já existir a função GetLastError retorna ERROR_ALREADY_EXISTS
  - Em caso de falha a constante NULL é retornada

Mutex Objects

- *DWORD WaitForSingleObject(*
  HANDLE hHandle,
  DWORD dwMilliseconds
  );

- *Espera até que o objeto seja liberado ou o tempo tenha se esgotado:*
  - *hHandle* é o identificador objeto
  - *dwMilliseconds* é o tempo de espera

Mutex Objects

- *BOOL ReleaseMutex(*
  HANDLE hMutex
  );

- *Libera o objeto*
  - *hMutex* é o objeto a ser liberado

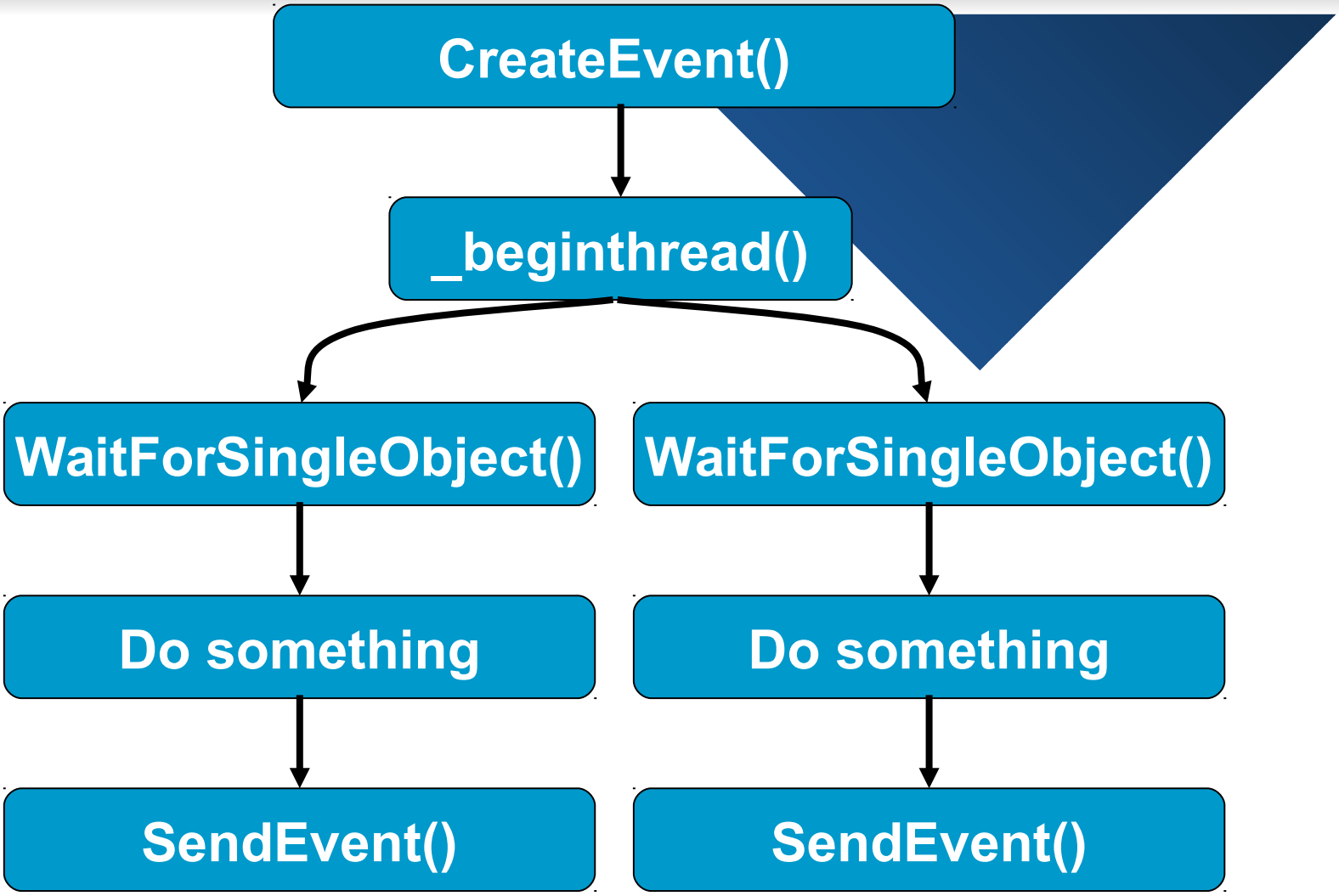
Exemplo

- *exemploThreadSincronizacaoMutex.cpp*

Eventos

- *Podemos utilizar eventos para sincronizar nossos threads*
- *Idéia básica:*
  - Fazer um thread esperar até que outro thread lance um evento
- *Para aguardar um evento também utilizamos a função WaitForSingleObject*

Eventos


CreateEvent()
_beginthread()
WaitForSingleObject()
WaitForSingleObject()
Do something
Do something
SendEvent()
SendEvent()

Eventos

- *O Windows possui dois tipos básicos de eventos:*
- *Manual-reset event:*
  - É um evento cujo estado permanece até que a função ResetEvent seja chamada
- *Auto-reset event:*
  - É um evento cujo estado permanece até que a função SetEvent seja chamada

Eventos

- *HANDLE CreateEvent(*
  LPSECURITY_ATTRIBUTES lpEvAtt,
  BOOL bManualReset,
  BOOL bInitialState,
  LPCTSTR lpName
);

  – Cria ou abre um evento

Eventos

- *lpEvAtt é determina se o id do evento será herdado por processos filhos*
- *bManualReset indica se o evento será resetado manualmente*
- *bInitialState indica o estado inicial do evento*
- *lpName é o nome do evento*
- *Retorno:*
  - A função retorna o id do objeto criado
  - Se o nome já existir a função GetLastError retorna ERROR_ALREADY_EXISTS
  - Em caso de falha a constante NULL é retornada

Eventos

- *BOOL SetEvent(*
  HANDLE hEvent

);

- *Dispara um evento*
  - O estado do evento é modificado
- *hEvent é o identificador do evento*
- *Retorna não zero em caso de sucesso e zero em caso de falha*

Resumo

Técnica	Performance	Múltiplos Processos	Timer	Plataformas
Critical Section	Rápida	Não	Não	9x/NT/CE
Mutex	Lenta	Sim	Sim	9x/NT/CE
Event	Lenta	Sim	Sim	9x/NT/CE

Threads e a MFC

- *A MFC possui um conjunto de classe e funções para a criação e sincronização de threads*
- *As classe encapsulam todos as funcionalidades da API do windows*
- *Principais classes:*
  - CWinThread
  - CCriticalSection
  - CMutex
  - CEvent

Criando um Thread na MFC

- *Existem várias maneiras de se cria um thread na MFC*
- *A principal é utilizando a função AfxBeginThread*
- *Esta função está definida do arquivos cabeçalho afxwin.h*

Criando um Thread na MFC

- *CWinThread* AfxBeginThread(*
  AFX_THREADPROC pfnThreadProc,
  LPVOID pParam,
  int nPriority =
  THREAD_PRIORITY_NORMAL,
  UINT nStackSize = 0,
  DWORD dwCreateFlags = 0,
  LPSECURITY_ATTRIBUTES
  lpSecurityAttrs = NULL
);

Criando um Thread na MFC

- *pfnThreadProc é a função que será executada pelo thread*
- *LPVOID pParam, são os parâmetros para a função*
- *nPriority é a prioridade do thread*
- *nStackSize é o tamanho da pilha*
- *dwCreateFlags indica se o thread será iniciado após a criação*
- *lpSecurityAttrs atributos de segurança*

Prioridades de um thread

- *REALTIME_PRIORITY_CLASS*
- *HIGH_PRIORITY_CLASS*
- *NORMAL_PRIORITY_CLASS*
- *IDLE_PRIORITY_CLASS*

Criando um Thread na MFC

- *A função AfxBeginThread retorna o objeto do tipo CWinThread*
- *Ela representa o thread criado*

Destruindo um Thread

- *void AFXAPI AfxEndThread(*
  UINT nExitCode,
  BOOL bDelete = TRUE
  );

- *nExitCode especifica o código de saída do thread*
- *bDelete apaga o objeto thread da memória*

Sincronizando threads

- *A MFC possui classes que encapsulam a API do windows:*
  - exemploThreadSincronizacaoSessaoMFC.cpp
  - exemploThreadSincronizacaoMutexMFC.cpp
  - exemploThreadSincronizacaoEventoMFC.cpp

Referências

- *MSDN*
  - www.msdn.microsoft.com
- *Introduction to Multi-threaded Code by By William T. Block*
  - http://www.codeproject.com/threads/sync.asp
- *Multithreaded Applications using MFC*
  - http://www.murrayc.com/learning/windows/multithreading.shtml
- *Digital Mars*
  - http://www.digitalmars.com/rtl/process.html