GERENCIAMENTO DE PROCESSOS

NO WINDOWS 7

CONTEÚDO

1. Processos
2. Threads
3. Sincronização

1

PROCESSOS

GERENCIADOR DE PROCESSOS

- Fornece serviços para a criação, exclusão e uso de processos, threads e jobs.

- Questões sobre relacionamentos ou hierarquias dos processos, são tratadas pelo subsistema ambiental específico que possui o processo.

- Processos são criados com a chamada do método CreateProcess()

GERENCIAMENTO DE PROESSOS

- O Windows utiliza 4 classes de prioridade:
  - IDLE_PRIORITY_CLASS (nível 4)
  - NORMAL_PRIORITY_CLASS (nível 8)
  - HIGH_PRIORITY_CLASS (nível 13)
  - REALTIME_PRIORITY_CLASS (nível 24)

- Para processos NORMAL_PRIORITY_CLASS, o windows diferencia entre processos foreground e processos background.

- Processos foreground recebem um quantum maior em todas as suas threads.

2

THREADS

THREADS WINDOWS

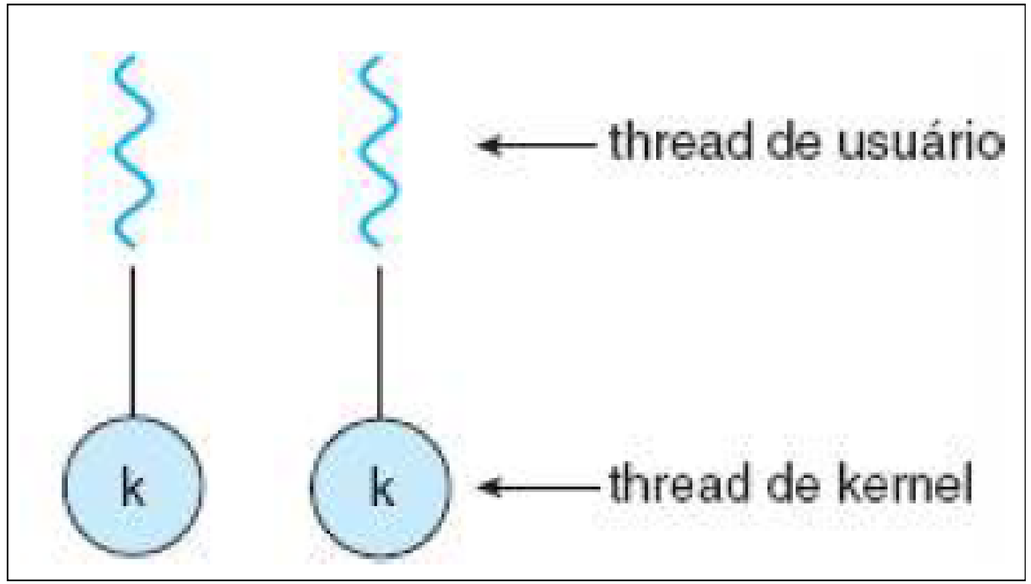
Thread: executa um pedaço do código do processo no contexto do processo, usando os recursos do processo

Uma aplicação do Windows é executada como um processo separado, e cada processo pode conter uma ou mais threads

Múltiplas threads podem ser executadas em paralelo em multiprocessadoress

Overhead de criação de threads de kernel pode sobrecarregar desempenho

Windows usa o mapeamento um-para-um



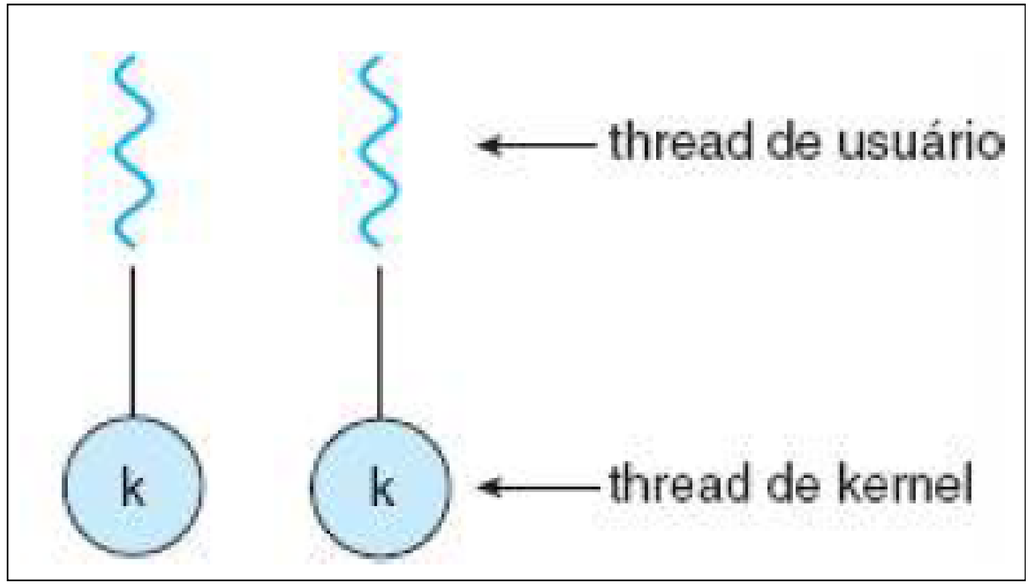
PRINCIPAIS ESTRUTURAS

- ETHREAD — bloco de thread executivo
  - Inclui um ponteiro para o processo ao qual o thread pertence e um endereço
  - Ponteiro para KTHREAD

- KTHREAD — bloco de thread do kernel
  - Inclui informações de scheduling e sincronização
  - Pilha e ponteiro para TEB

- TEB — bloco de ambiente do thread
  - Estrutura de dados do espaço do usuário

Windows usa o mapeamento um-para-um



THREADS WINDOWS

- Uso da API Windows para criação de threads com a função `CreateThread ( )`

- Conjunto de atributos passados para a função

- Inclusão de informações de segurança, o tamanho da pilha e uma flag
  - Permite criar a thread em um estado suspenso
  - Quando suspensa, a thread não é executado até que a função ResumeThread( ) seja chamada.

CRIAÇÃO DE THREADS

- Sintaxe

```
C++

HANDLE CreateThread(
  [in, optional]  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  [in]            SIZE_T                  dwStackSize,
  [in]            LPTHREAD_START_ROUTINE  lpStartAddress,
  [in, optional]  __drv_aliasesMem LPVOID lpParameter,
  [in]            DWORD                   dwCreationFlags,
  [out, optional] LPDWORD                 lpThreadId
);
```

Se o identificador retornado pode ser herdado por processos filhos

Tamanho da pilha em bytes

Ponteiro para a função que a thread executará

Parâmetro que será passado para a função

Controle do comportamento da criação da thread

Ponteiro para variável que recebe o id da thread

ESPERANDO A THREAD TERMINAR

- No Windows, quando um processo termina, todas as suas threads também
- É necessário que o programa principal espere pelo término das threads que criou.

```C++
DWORD WaitForSingleObject(
  [in] HANDLE hHandle,
  [in] DWORD  dwMilliseconds
);
```

Identificador para o objeto a ser esperado

Intervalo em milissegundos

OUTRAS FUNÇÕES PARA GERENCIAMENTO DE THREADS NO WINDOWS

`ExitThread()`

Encerra a execução da thread atual e retorna um código de saída

`GetExitCodeThread()`

Coleta informações sobre o resultado de uma thread que terminou.

`GetCurrentThread()`

Retorna um identificador para a thread atual

`OpenThread()`

Obtém um identificador de uma thread existente (sem ser a atual)

`SuspendThread()`

Suspende a execução de uma thread.

`ResumeThread()`

Retoma a execução de uma thread que foi suspensa

`CreateRemoteThread()`

Cria uma nova thread em um processo remoto

PRIORIDADE DE THREADS

- O despachante usa um esquema de prioridades de 32 níveis para determinar a ordem de execução das threads

- Divisão em duas classes:
  - Variável: contém threads que têm prioridades de 1 a 15
  - Tempo real: contém threads com prioridades variando de 16 a 31

CLASSE VARIÁVEL

- A prioridade da thread pode ser alterada dinamicamente
- Usada para threads que não precisam de tempo de resposta garantido

CLASSE EM TEMPO REAL

- A prioridade do thread não pode ser alterada dinamicamente
- Usada para threads que precisam de tempo de resposta garantido (ex.: drivers)

PRIORIDADES DE THREADS

- Uma thread começa com uma prioridade inicial determinada por sua classe.

- A prioridade pode ser alterada pela função SetThreadPriority ( ).
  - THREAD_PRIORITY_LOWEST: base – 2
  - THREAD_PRIORITY_BELOW_NORMAL: base – 1
  - THREAD_PRIORITY_NORMAL: base + 0
  - THREAD_PRIORITY_ABOVE_NORMAL: base + 1
  - THREAD_PRIORITY_HIGHEST: base + 2

- O kernel ajusta dinamicamente a prioridade de um thread de classe variável, dependendo de o thread ser limitado por I/O ou limitado por CPU.

ESCALONAMENTO DE THREADS

- O despachante do kernel é responsável por scheduling de threads e a mudança de contexto.

- Há seis estados possíveis para as threads: pronto, disponível, em execução, em espera, em transição e encerrado.

- É utilizado um esquema de prioridades de 32 níveis. Existe uma fila para cada nível de prioridade.

ESCALONAMENTO DE THREADS

- O despachante percorre as filas em ordem de prioridade até achar uma thread no estado de pronto.

- A thread pronta de mais alta prioridade é passada para o estado disponível, e será a próxima a ser executada.

- A thread continua em execução até que:
  - Terminar de executar.
  - Sofrer preenpção por uma nova thread de prioridade mais alta.
  - Seu tempo de alocação (quantum) finalizar.
  - Ou entrar em espera, como em um evento de I/O.

3

SINCRONIZAÇÃO DE PROCESSOS

SOBRE A SINCRONIZAÇÃO

Para sincronizar o acesso a um recurso, usamos:

- Objetos de sincronização
- Funções de Espera

O estado de um objeto de sincronização pode ser sinalizado ou não atribuído

As funções de espera permitem que uma thread bloqueie sua própria execução até que um objeto não atribuído especificado seja definido como o estado sinalizado

OBJETOS DE SINCRONIZAÇÃO

- Evento
- Mutex
- Sinal
- Temporizador de espera

OBJETOS DE SINCRONIZAÇÃO

- Evento
- Mutex
- Sinal
- Temporizador de espera

EVENTO

Notifica um ou mais threads de espera em que um evento ocorreu

- Evento de redefinição manual

  Um objeto de evento cujo estado permanece sinalizado até que seja redefinido explicitamente para não atribuído pela função ResetEvent .

- Evento de redefinição automática

  Um objeto de evento cujo estado permanece sinalizado até que um único thread de espera seja liberado, momento em que o sistema define automaticamente o estado como não atribuído.

OBJETOS DE SINCRONIZAÇÃO

- Evento
- Mutex
- Sinal
- Temporizador de espera

MUTEX

Pode ser propriedade de apenas um thread por vez, permitindo que os threads coordenem o acesso mutuamente exclusivo a um recurso compartilhado

Um objeto mutex é um objeto de sincronização cujo estado é definido como sinalizado quando não pertence a nenhum thread e não é atribuído quando pertence

Apenas um thread de cada vez pode possuir um objeto mutex

OBJETOS DE SINCRONIZAÇÃO

- Evento
- Mutex
- Sinal
- Temporizador de espera

SINAL

Mantém uma contagem entre zero e determinado valor máximo, limitando o número de threads que estão acessando simultaneamente um recurso compartilhado

A contagem é decrementada sempre que um thread conclui uma espera pelo objeto de semáforo e incrementada sempre que um thread libera o semáforo.

O objeto semáforo é útil para controlar um recurso compartilhado que pode dar suporte a um número limitado de usuários.

OBJETOS DE SINCRONIZAÇÃO

- Evento
- Mutex
- Sinal
- Temporizador de espera

TEMPORIZADOR DE ESPERA

Notifica um ou mais threads de espera de que chegou uma hora especificada

- temporizador de redefinição manual

  Um temporizador cujo estado permanece sinalizado até que SetWaitableTimer seja chamado para estabelecer um novo tempo de conclusão

OBJETOS DE SINCRONIZAÇÃO

- Evento
- Mutex
- Sinal
- Temporizador de espera

TEMPORIZADOR DE ESPERA

- temporizador de sincronização

  Um temporizador cujo estado permanece sinalizado até que um thread conclua uma operação de espera no objeto de temporizador

- temporizador periódico

  Um temporizador que é reativado sempre que o período especificado expira, até que o temporizador seja redefinido ou cancelado

OBJETOS DE TEMPORIZADOR ESPERÁVEIS

temporizador.cpp > ...

```cpp
#include <windows.h>
#include <stdio.h>

int main()
{
    HANDLE hTimer = NULL;
    LARGE_INTEGER liDueTime;

    liDueTime.QuadPart = -100000000LL;

    // Create an unnamed waitable timer.
    hTimer = CreateWaitableTimer(NULL, TRUE, NULL);
    if (NULL == hTimer)
    {
        printf("CreateWaitableTimer failed (%d)\n", GetLastError());
        return 1;
    }

    printf("Waiting for 10 seconds...\n");

    // Set a timer to wait for 10 seconds.
    if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0))
    {
        printf("SetWaitableTimer failed (%d)\n", GetLastError());
        return 2;
    }

    // Wait for the timer.

    if (WaitForSingleObject(hTimer, INFINITE) != WAIT_OBJECT_0)
        printf("WaitForSingleObject failed (%d)\n", GetLastError());
    else printf("Timer was signaled.\n");

    return 0;
}
```

OBJETOS DE TEMPORIZADOR ESPERÁVEIS

```
liDueTime.QuadPart = -100000000LL;
```

- Define o temporizador para esperar por 10 segundos (10.000.000 microssegundos)

```
hTimer = CreateWaitableTimer(NULL, TRUE, NULL);
```

Cria um temporizador

- NULL: Indica que o temporizador é sem nome
- TRUE: Indica que o temporizador é manual-reset, ou seja, ele permanece sinalizado até ser reiniciado manualmente.
- NULL: Parâmetro de segurança

OBJETOS DE TEMPORIZADOR ESPERÁVEIS

```
if (NULL == hTimer)
{
    printf("CreateWaitableTimer failed (%d)\n", GetLastError());
    return 1;
}
```

- Se a criação do temporizador falhar, uma mensagem de erro é impressa e o programa termina com código de retorno 1

OBJETOS DE TEMPORIZADOR ESPERÁVEIS

```
if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0))
```

- Configura o temporizador para aguardar 10 segundos

```
if (WaitForSingleObject(hTimer, INFINITE) != WAIT_OBJECT_0)
    printf("WaitForSingleObject failed (%d)\n", GetLastError());
else printf("Timer was signaled.\n");
```

- Aguarda até que o temporizador seja sinalizado, se falhar, exibe uma mensagem de erro, caso contrário, exibe uma mensagem indicando que o temporizador foi sinalizado.

MERCI!