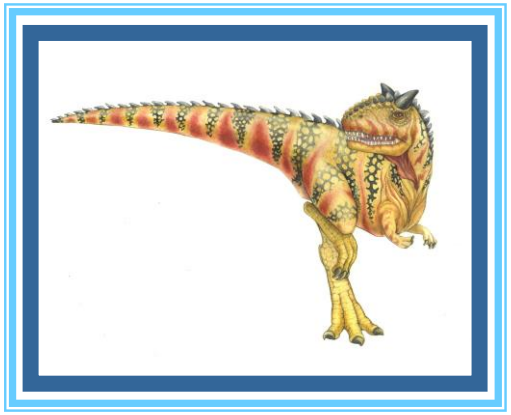
Capítulo 4: Threads



Capítulo 4: Threads

- Visão Geral
- Programação Multicore
- Modelos de Geração de Multithreads
- Bibliotecas de Threads
- Threading Implícito
- Threading Issues
- Exemplos de Sistemas Operacionais

Universidade de Brasília

Faculdade UnB Gama

Objetivos

- Introduzir a noção de thread —uma unidade básica de utilização da CPU que forma a base dos sistemas de computação multithreaded
- Discutir as APIs das bibliotecas de threads Pthreads, Windows e Java.
- Explorar várias estratégias que fornecem a criação de threads implícita
- Examinar questões relacionadas com a programação com múltiplos threads
- Abordar o suporte do sistema operacional aos threads no Windows e no Linux

Universidade de Brasília

Faculdade UnB Gama

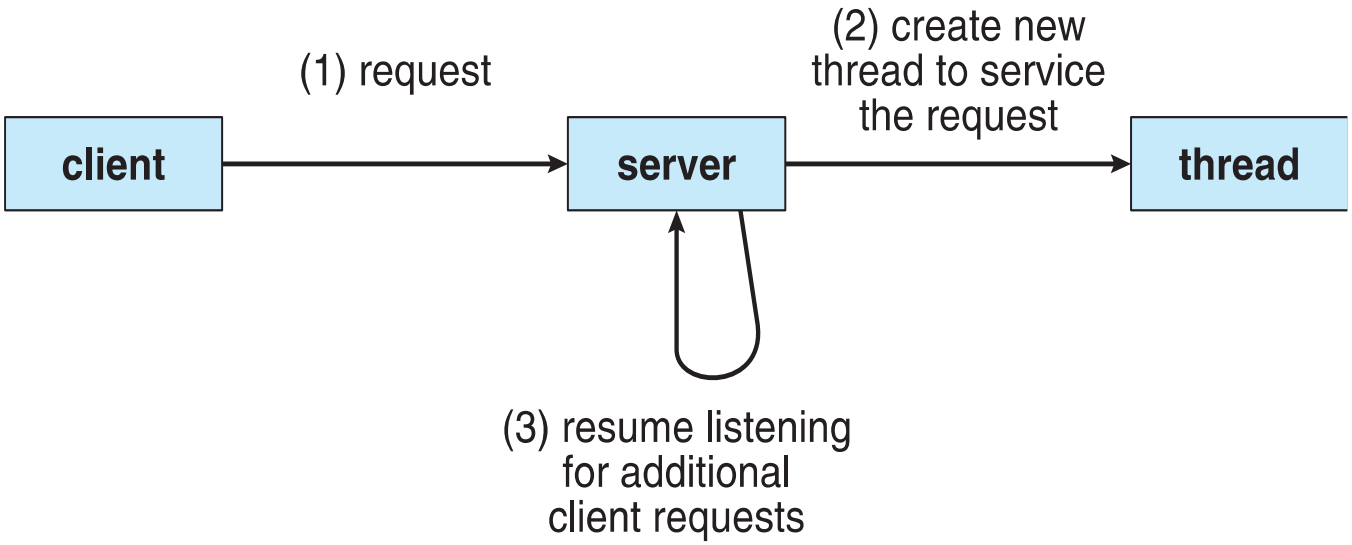
Motivação

- A maioria das aplicações de software executadas em computadores modernos é multithreads
- Threads são executados dentro de aplicações
- Múltiplas tarefas com uma aplicação podem ser implementadas por threads diferentes
  - Atualizar tela
  - Recuperar dados
  - Verificação ortográfica
  - Responder a uma solicitação de rede
- A criação de processos é demorada e usa muitos recursos enquanto a criação de threads é leve e ágil
- Pode simplificar códigos e aumentar a eficieência
- Kernels são geralmente multithreaded

Universidade de Brasília

Faculdade UnB Gama

Arquitetura de servidor com múltiplos threads



Benefícios

- **Capacidade de resposta –** pode permitir que um programa continue a ser executado, mesmo que parte dele esteja bloqueada, o que é particularmente útil no projeto de interfaces de usuário
- **Compartilhamento de recursos –** threads compartillham os recursos dos processos e isso é mais fácil do que memória compartilhada e transmissão de mensagens
- **Economia –** mais barato do que criação de processos, mudanças de thread tem overhead menor do que a mudança de contexto
- **Escalabilidade –** o processo pode aproveitar da arquitetura de multiprocessadora

Programação Multicore

- **Sistemas Multicore** or **multiprocessados** colocam pressão sobre os programadores, os desafios incluem:
  - **Identificação de tarefas**
  - **Equilíbrio**
  - **Divisão de dados**
  - **Dependência de dados**
  - **Teste e depuração**
- ***Paralelismo* -** um sistema é paralelo quando pode executar mais de uma tarefa simultaneamente

- ***Concorrência* -** dá suporte a mais de uma tarefa, permitindo que todas elas progridam
  - Único processador / núcleo, o scheduler fornece concorrência

Universidade de Brasília

Faculdade UnB Gama

Programação Multicore (Cont.)

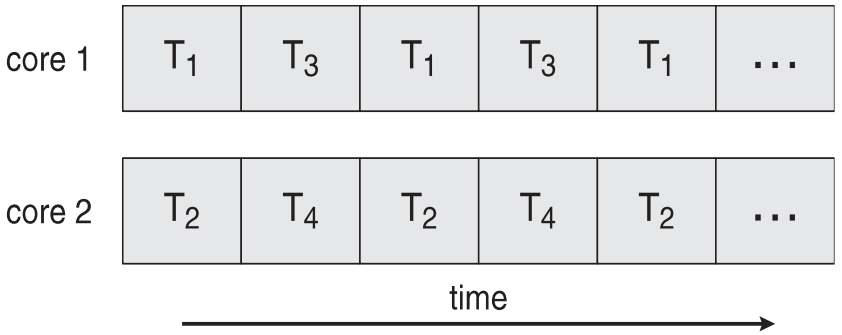
- Tipos de paralelismo
  - **Paralelismo de dados** – distribuição dos subconjuntos dos mesmos dados, por múltiplos núcleos de computação, e execução da mesma operação em cada núcleo
  - **Paralelismo de tarefas** –distribuição não de dados, mas de tarefas (threads) em vários núcleos de computação separados. Cada thread executa uma única operação

- A medida que o número de threads cresce, cresce também o suporte de arquitetrua para o threading
  - CPUs têm núcleos assim como ***threads de hardware***
  - Considere Oracle SPARC T4 com 8 núcleos, e 8 threads de hardware por núcleo

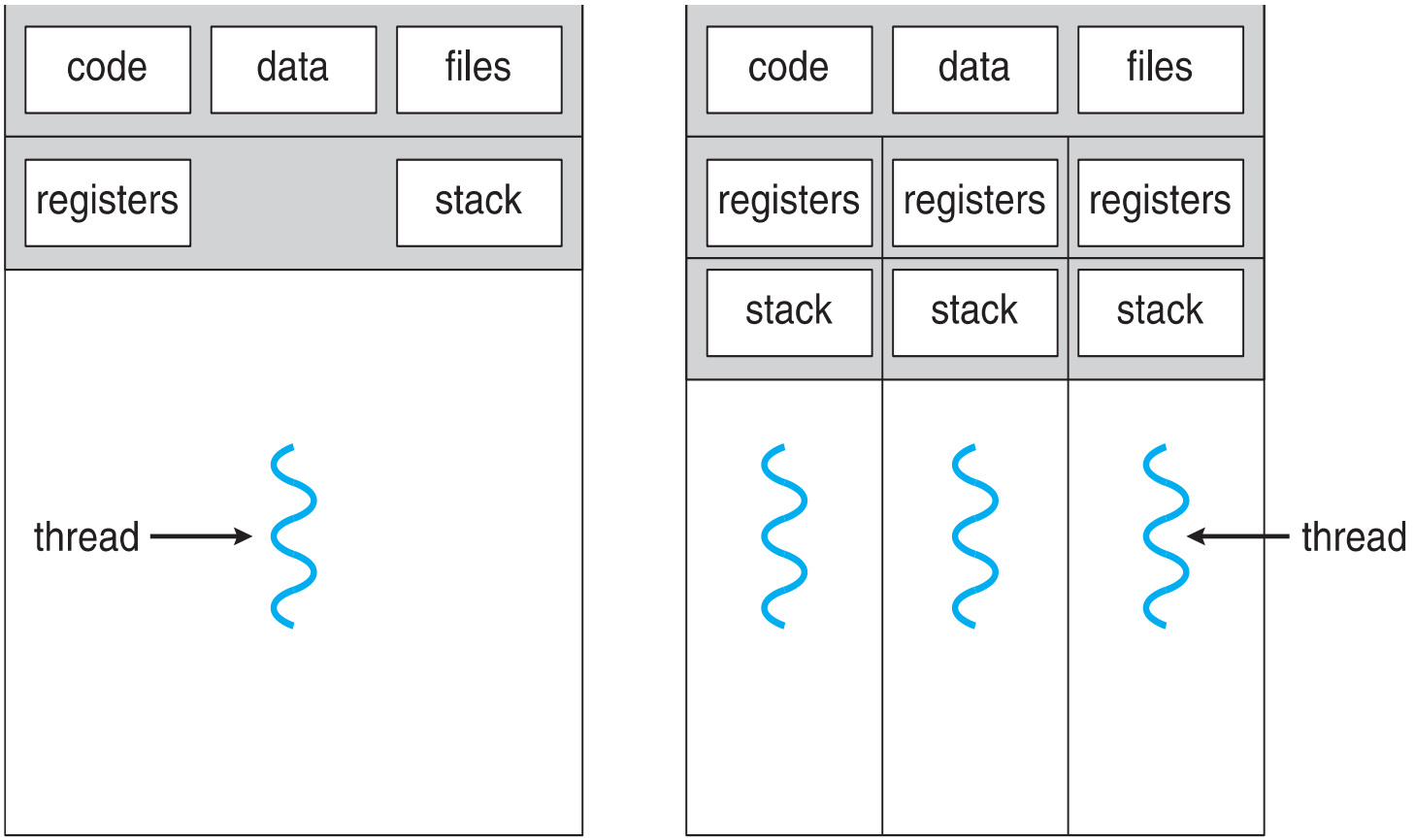
Concorrência vs. Paralelismo

- **Execução concorrente em um sistema com um único núcleo :**

- **Execução paralela em um sistema multicore:**



Processos com um único thread e com múltiplos threads



Universidade de Brasília

Faculdade UnB Gama

Lei de Amdahl

- Identifica ganhos de desempenho potenciais com a inclusão de núcleos de computação adicionais em uma aplicação que tenha componentes tanto seriais quanto paralelos
- *S* é a parte serial
- *N* núcleos de processamento

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- Caso tenhamos uma aplicação que seja 75% paralela e 25% serial, a mudança de 1 para 2 núcleos resulta em uma aceleração de 1,6 vez
- À medida que N se aproxima do infinito, a aceleração converge para 1/S
  A parte serial de uma aplicação pode ter um efeito desproporcional sobre o ganho de desempenho com a inclusão de núcleos de computação adicionais
- Mas essa lei leva em consideração sistemas multicore contemporâneos ?

Universidade de Brasília

Faculdade UnB Gama

Threads de Usuário e Threads de Kernel

- **Threads de Usuário** – o suporte aos threads é feito no nível do usuário
- Três bibllliotecas primárias de thread :
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Threads de Kernel** – Suporte feito no nível do kernel
- Exemplos – virtualmente todos os sistemas operacionais de propósito geral, incluindo:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Universidade de Brasília

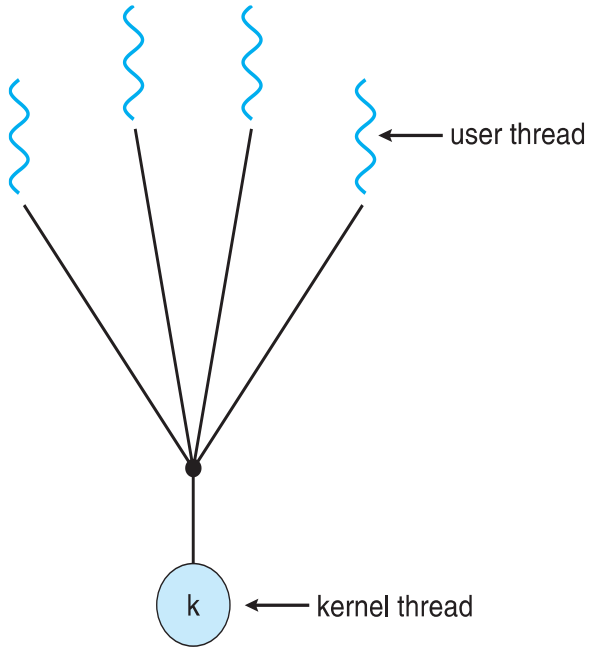
Faculdade UnB Gama

Modelos de Geração de Multithreads

- Muitos-Para-Um
- Um-Para-Um
- Muitos-Para-Muitos

Muitos-Para-Um

- Mapeia muitos threads de nível de usuário para um thread de kernel
- Se um thread fizer uma chamada de sistema bloqueadora, o processo inteiro será bloqueado
- muitos threads ficam incapazes de executar em paralelo em sistemas multicore porque apenas um thread por vez pode acessar o kernel
- Poucos sistemas atuais usam esse modelo:
- Exemplos:
  - **Solaris Green Threads**
  - **GNU Portable Threads**

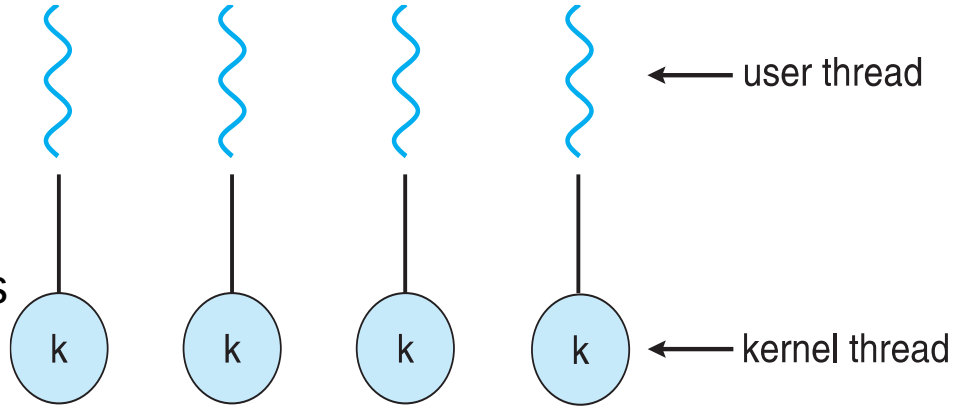


Universidade de Brasília

Faculdade UnB Gama

Um-Para-Um

- Mapeia cada thread de usuário para um thread de kernel
- Ao criar um thread de usuário cria um thread dekernel
- Mais concorrência do que muitos-para-um
- Número de threads por processo às vezes é restringido por causa de overhead
- Exemplos
  - Windows
  - Linux
  - Solaris 9 e mais recentes

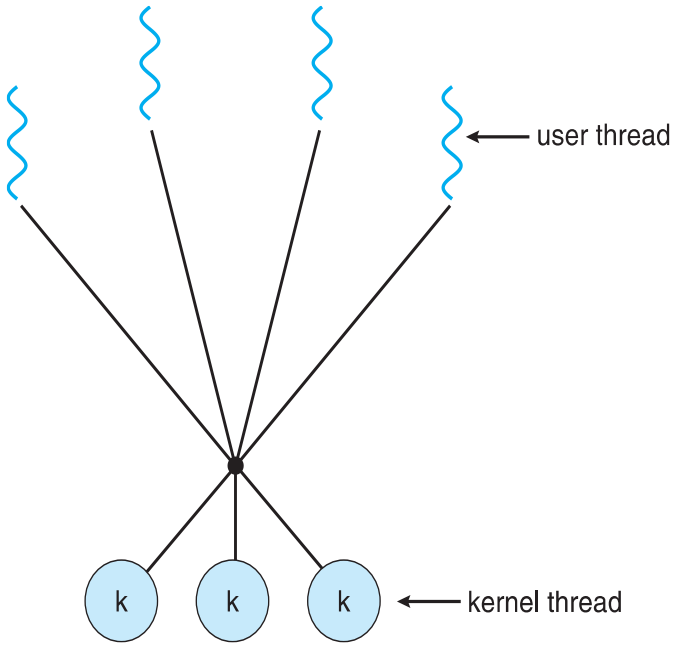


Universidade de Brasília

Faculdade UnB Gama

Modelo Muitos-Para-Muitos

- Permite que muitos threads de usuários sejam mapeados para muitos threads kernel
- Permite que o sistema operacional crie um número suficiente de threads de kernel
- Solaris anterior a versão 9
- Windows com o pacote *ThreadFiber*

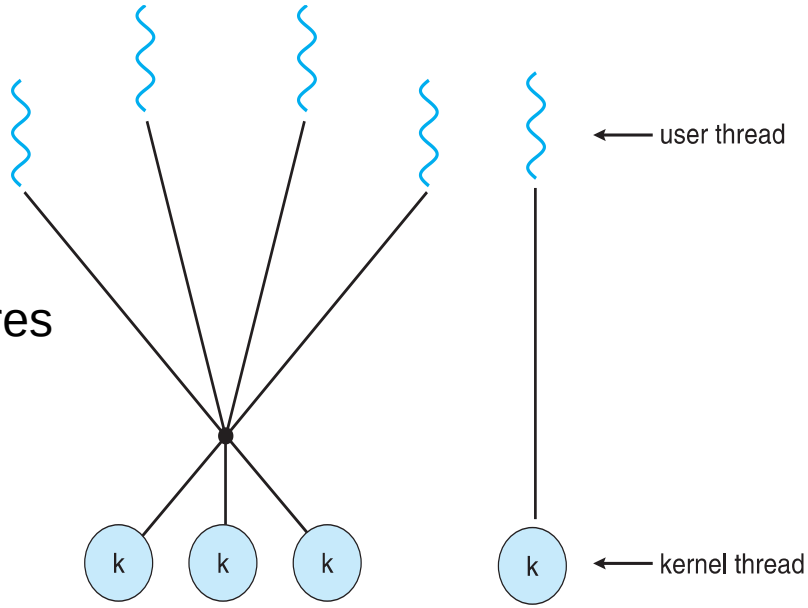


Universidade de Brasília

Faculdade UnB Gama

Modelo de Dois Níveis

- Similar a M:M, mas permite que um thread de nível de usuário seja **limitado** a um thread de kernel
- Exemplos
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 e versões anteriores



Bibliotecas de Threads

- **Biblioteca de Threads** fornece ao programador uma API para criação e gerenciamento de threads

- Há duas maneiras principais de implementar uma
  - Biblioteca inteira no espaço do usuário
  - Biblioteca no nível do kernel com suporte direto do sistema operacional

Pthreads

- Pode ser fornecida tanto para o nível do usuário quanto o nível do kernel
- Uma API de padrão POSIX (IEEE 1003.1c) para a criação e sinchronização de threads
- ***Especificação***, não ***implementação***
- API especifica o o comportamento da biblioteca de threads, implementação depende do desenvolvimento da biblioteca
- Comum em sistemas operacionais UNIX (Solaris, Linux, Mac OS X)

Exemplo de Pthreads

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }
```

Universidade de Brasília

Faculdade UnB Gama

Exemplo de Pthreads (Cont.)

```
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

Código Pthread para o agrupamento de dez threads.

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Universidade de Brasília

Faculdade UnB Gama

Programa em C com múltiplos threads usando a API Windows.

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;

   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }
```

Programa em C com múltiplos threads usando a API Windows.

```
/* create the thread */
ThreadHandle = CreateThread(
   NULL, /* default security attributes */
   0, /* default stack size */
   Summation, /* thread function */
   &Param, /* parameter to thread function */
   0, /* default creation flags */
   &ThreadId); /* returns the thread identifier */

if (ThreadHandle != NULL) {
    /* now wait for the thread to finish */
   WaitForSingleObject(ThreadHandle,INFINITE);

   /* close the thread handle */
   CloseHandle(ThreadHandle);

   printf("sum = %d\n",Sum);
}
}
```

Threads Java

- Os threads do Java são gerenciados pela JVM
- São tipicamente implementados usando o modelo de threads fornecido pelo sistema operacional.
- Threads de Java podem ser criados por:

```
public interface Runnable
{
    public abstract void run();
}
```

  - Criação de uma nova classe derivada da classe Thread e a sobreposição de seu método run ( )
  - Implementação de interface Runnable

Universidade de Brasília

Faculdade UnB Gama

Programa Multithreads em Java

```
class Sum
{
   private int sum;

   public int getSum() {
     return sum;
   }

   public void setSum(int sum) {
     this.sum = sum;
   }
}

class Summation implements Runnable
{
   private int upper;
   private Sum sumValue;

   public Summation(int upper, Sum sumValue) {
     this.upper = upper;
     this.sumValue = sumValue;
   }

   public void run() {
     int sum = 0;
     for (int i = 0; i <= upper; i++)
        sum += i;
     sumValue.setSum(sum);
   }
}
```

Programa Multithreads em Java (Cont.)

```
public class Driver
{
   public static void main(String[] args) {
     if (args.length > 0) {
       if (Integer.parseInt(args[0]) < 0)
         System.err.println(args[0] + " must be >= 0.");
       else {
         Sum sumObject = new Sum();
         int upper = Integer.parseInt(args[0]);
         Thread thrd = new Thread(new Summation(upper, sumObject));
         thrd.start();
         try {
            thrd.join();
            System.out.println
                    ("The sum of "+upper+" is "+sumObject.getSum());
       } catch (InterruptedException ie) { }
       }
     }
     else
       System.err.println("Usage: Summation <integer value>"); }
}
```

Threading Implícito

- As aplicações com centenas ou milhares de threads estão crescendo em popularidade e por isso a correção dos programas é mais difícil com threads explícitos
- A criação e gerenciamento de threads é feito por compiladores done by compilers bibliotecas de tempo de execução em vez de desenvolvedores de aplicações.
- Three methods explored
  - Pools de Threads
  - OpenMP
  - Grand Central Dispatch
- Outors métodos incluem Microsoft Threading Building Blocks (TBB), pacote **`java.util.concurrent`**

Universidade de Brasília

Faculdade UnB Gama

Thread Pools

- Cria múltiplos threads em um pool onde eles ficam esperando para entrar em ação
- Vantagens:
  - Geralmente é mais rápido atender a solicitação com um thread existente do que criar um novo thread
  - Permite que o número de threads nas aplicaçõers seja vinculado ao tamanho do pool
  - A separação da tarefa a ser executada da mecânica de criação da tarefa permite-nos usar diferentes estratégias para execução da tarefa.
    - Exemplo: As tarefas podem ser agendadas para serem executadas periodicamente
- A API do Windows suportapools de thread :

```
DWORD WINAPI PoolFunction(AVOID Param) {
    /*
    * this function runs as a separate thread.
    */
}
```

Universidade de Brasília Faculdade UnB Gama 

OpenMP

- É um conjunto de diretivas de compilador assim como uma API para programas escritos em C, C++, FORTRAN
- Dá suporte à programação paralela em ambientes de memória compartilhada
- Identifica **regiões paralelas** como bloco de código que podem ser executados er paralelo

`#pragma omp parallel`

Cria tantos threads quanto o número de núcleos

```
#pragma omp parallel for
   for(i=0;i<N;i++) {
     c[i] = a[i] + b[i];
}
```

Executa o loop for m paralelo

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   /* sequential code */

   #pragma omp parallel
   {
      printf("I am a parallel region.");
   }

   /* sequential code */

   return 0;
}
```

Grand Central Dispatch

- Tecnologia para sistemas operacionais Mac OS X e iOS
- Extenções para as linguagens C, C++, uma API, e uma biblioteca de tempo de execução
- Permite identificação de seções de código a serem executadas em paralelo
- Gerencia a maioria dos detalhes da criação de threads
- Bloco está em "^{ }" - `^{ printf("I am a block"); }`
- Os blocos são colocados em uma fila de despacho
  - São designados a um thread disponível no quando são removidos de uma fila

Grand Central Dispatch

- Dois tipos de filas de despacho:
  - serial – blocos são removidos em ordem FIFO, cada processo tem sua fila serial, chamada de **fila principal**
    - Os desenvolvedores podem criar filas seriais adicionais dentro do programa
  - concorrente – blocos também são removidos em ordem FIFO, mas vários blocos podem ser removidos ao mesmo tempo
    - Três filas de despacho concorrentes com abrangência em todo o sistema com as seguintes prioridades: baixa, default e alta.

```
dispatch_queue_t queue = dispatch_get_global_queue
  (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{ printf("I am a block."); });
```

Questões Relacionadas com a Criação de Threads

- A semântica das chamadas de sistema **fork ( )** e **exec ( )**
- Manipulação de Sinais
  - Síncrona ou assíncrona
- Cancelamento de Threads do thread alvo
  - Assíncrona ou adiado
- Armazenamento Local do Thread
- Ativações do Scheduler

Universidade de Brasília

Faculdade UnB Gama

Semantics of fork() and exec()

- `fork()` duplicará todos os threads, ou o novo processo terá um único thread?
- Alguns sistemas UNIX tem duas versões de fork
- `exec()` geralmente funciona como de costume – substitui o processo sendo executado incluindo todos os threads

Universidade de Brasília

Faculdade UnB Gama

Manipulação de Sinais

- **Sinais** são usados em sistemas UNIX para notificar um processo de que um evento específico aconteceu.
- Um **manipulador de sinal** é usado para processar sinais
  1. Sinal é gerado por um evento específico
  2. Sinal é liberado para um processo
  3. Sinal é manipulado por um ou dois manipuladores de sinal:
     1. default
     2. definido pelo usuário
- Todo sinal tem um **manipulador de sinal** default que o kernel executa ao manipular o sinal that kernel runs when handling signal
  - **Manipulador de sinais definido pelo usuário** pode se sobrer ao default
  - Para os programas com um único thread, os sinais são sempre liberados para um processo

Universidade de Brasília

Faculdade UnB Gama

Manipulação de Sinais (Cont.)

- Para onde um sinal deve ser liberado nos programas multithreaded?
  - Liberar o sinal para o thread ao qual ele é aplicável
  - Liberar o sinal para cada thread do processo
  - Liberar o sinal para certos threads do processo
  - Designar um thread específico para receber todos os sinais do processo

Cancelamento de Threads

- Encerramento de um thread antes que ele seja concluído
- O thread que está para ser cancelado é o **thread-alvo**
- Duas abordagens gerais:
  - **Cancelamento assíncrono** encerra o thread-alvo imediatamente
  - **Cancelamento adiado** permite que o thread-alvo verifique periodicamente se ele deve ser cancelado
- Código Pthread para criar e cancelar um thread:

```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

. . .

/* cancel the thread */
pthread_cancel(tid);
```

Cancelamento de Threads (Cont.)

- A invocação de cancelamento de thread indica apenas uma solicitação, mas o cancelamento real depende do estado do thread

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- Se o thread tiver o cancelamento desabilitado, o cancelmento permanece pendente até que o thread o habilite
- O modo default é o adiado
  - Cancelamento somente occorre quando o thread alcança um **ponto de cancelamento**
    - I.e. `pthread_testcancel()`
    - Depois **o manipulador de limpeza** é invocado.
- Nos sistemas Linux, o cancelamento de thread é manipulado por sinais.

Universidade de Brasília

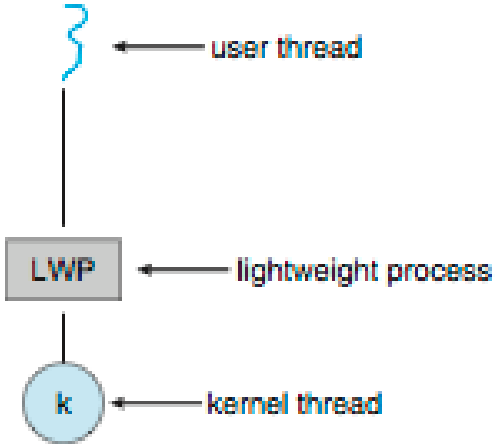
Faculdade UnB Gama

Armazenamento Local do Thread

- **Armazenamento local de thread** (**TLS**) permite que cada thread tenha a sua propria cópia de dados
- É útil quando não se tem controle sobre o processo de criação de thread (exemplo: quando se usa um pool de threads)
- É diferente das variáveis locais
  - As variáveis locais são visíveis apenas durante uma única invocação de função
  - TLS são visíveis ao longo das invocações de funções
- Similar ao dados `static`
  - TLS são únicos para cada thread

Ativações do Scheduler

- Tanto M:M e modelos de dois níveis requerem comunicação para manter o número adequado de threads de kernel alocados para a aplicação
- Normalmente usam uma estrutura de dados intermediária entre entre os threads do usuário e do kernel– **processo peso leve** (**LWP**)
  - Aparece como um processador virtual em que a aplicação pode incluir no schedule um thread de usuário para execução
  - Cada LWP é anexado a um thread do kernel
  - Quantos LWPs devem ser criados?
- Ativação do scheduler fornece **upcalls** – um mecanismo de comunicação do kernel com o **manipulador de upcall** na biblioteca de threads
- Essa comunicação permite que uma aplicação mantenha o número correto de threads de kernel



Universidade de Brasília

Faculdade UnB Gama

Exemplos de Sistemas Operacionais

- Threads no Windows
- Threads no Linux

Universidade de Brasília

Faculdade UnB Gama

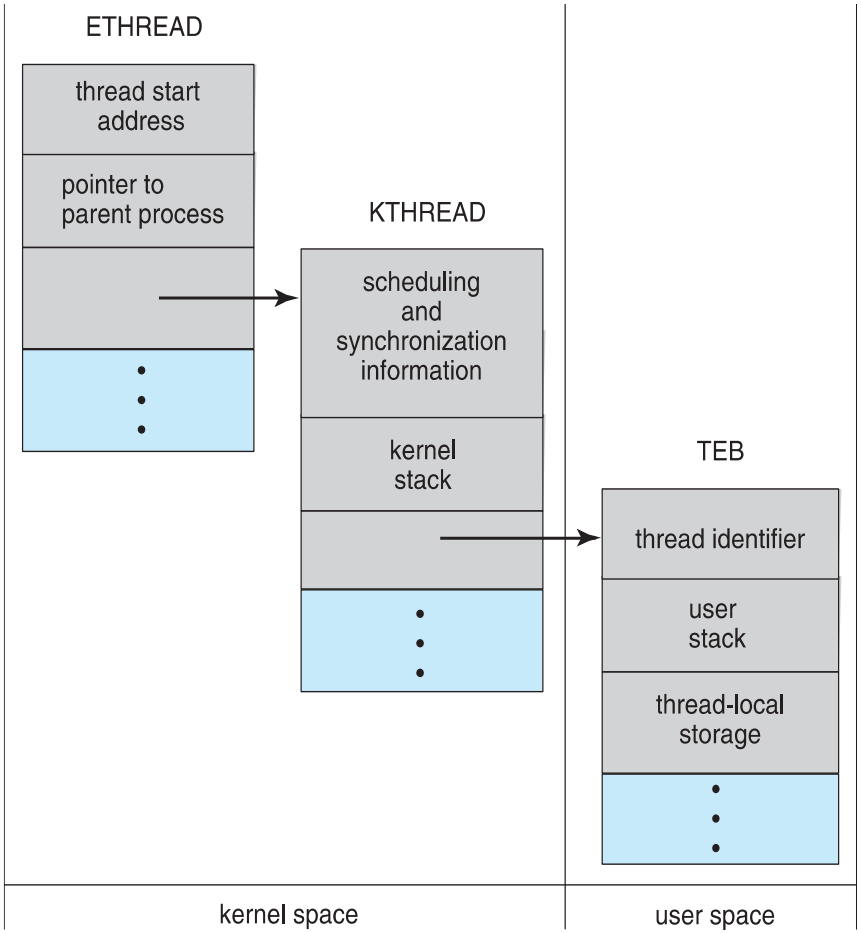
Threads no Windows

- Windows implementa a API Windows– principal API para Win 98, Win NT, Win 2000, Win XP, and Win 7
- Implementa o mapeamento um-para-um no nível de kernel
- Cada thread contém
  - Um ID de thread
  - Um conjunto de registradores representando o status do processador
  - Pilhas de usuário e de kernel separadas para quando os threads forem executados na modalidade de usuário e de de kernel
- Uma área de armazenamento privada usada por várias bibliotecas de tempo de execução e bibliotecas de links dinâmicos (DLLs)
- O conjunto de registradores, as pilhas e a área de armazenamento privada são conhecidos como **context** do thread

Threads no Windows (Cont.)

- As principais estruturas de dados de um thread incluem :
  - ETHREAD (bloco de thread executivo) – inclue um ponteiro para o processo ao qual o thread pertence e para o KTHREAD, no espaço kernel
  - KTHREAD (bloco de thread do kernel) – inclui informações de scheduling e sincronização do thread, a pilha do kernel na modalidade de kernel, e um ponteiro para o TEB, no espaço kernel
  - TEB (bloco de ambiente do thread) – contém o identificador do thread, uma pilha de modalidade de usuário, armazenamento local de thread, no espaço do usuário

Estruturas de dados de um thread do Windows



Threads no Linux

- Linux usa o termo ***tarefas*** em vez de ***threads***
- A criação de threads é feita por meio da chamada de Sistema **`clone()`**
- **`clone()`** permite que uma tarefa filha compartilhe o mesmo endereço de espaço da tarefa pai (processo)
  - Flags controlam o comportamento

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

- **`struct task_struct`** contém ponteiros para estruturas de dados de processos (compartilhadas ou únicas)

Fim do Capítulo 4