


UNIVALI
Universidade do Vale do Itajaí
Escola Politécnica
Engenharia de Computação

Threads e Concorrência

1




UNIVALI

Agenda

- Visão geral
- Programação Multicore
- Modelos Multithreading
- Bibliotecas de threads
- Threading - automatização
- Problemas de encadeamento
- Exemplos de sistemas operacionais

2




UNIVALI

Motivação

- A maioria dos aplicativos modernos são multithreaded
- Threads executadas dentro do aplicativo
- Várias tarefas podem ser implementadas por threads separadas
 - Atualizar exibição
 - Buscar dados
 - Verificação ortográfica
 - Responder a uma solicitação de rede
- A criação de processos é pesada, enquanto a criação de threads é leve
- Pode simplificar o código, aumentar a eficiência
- Os kernels são geralmente multithreaded

3



UNIVALI

Processos Single e Multithreaded

code

data

files

registers

PC

stack

thread

single-threaded process

code

data

files

registers

registers

registers

stack

stack

stack

PC

PC

PC

thread

multithreaded process

4

Benefícios

- **Responsividade** – pode permitir a execução contínua se parte do processo estiver bloqueada, especialmente importante para interfaces de usuário
- **Compartilhamento de recursos** – threads compartilham recursos de processo, mais fácil do que a memória compartilhada ou a passagem de mensagens
- **Economia** – mais barato do que a criação de processos, a alternância de threads é mais baixa do que a alternância de contexto
- **Escalabilidade** – o processo pode tirar proveito de arquiteturas multicore



5

Programação Multicore

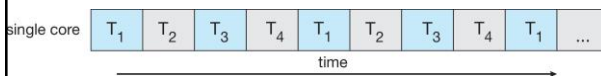
- Sistemas **Multicore** ou **Multiprocessador** colocando pressão sobre os programadores com desafios que incluem:
 - Dividindo atividades, balanceamento de carga, divisão de dado, dependência de dados e testar e depurar
- **Paralelismo** implica que um sistema pode executar mais de uma tarefa simultaneamente
 - Dados e instrução são diferentes
- **Concorrência** oferece suporte a mais de uma tarefa que está ainda executando
 - Processador de um único núcleo sofre concorrência por meio do escalanador



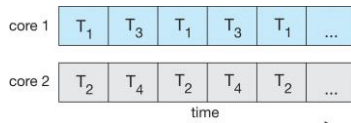
6

Concorrência vs. paralelismo

Execução simultânea em sistema single-core:



Paralelismo em um sistema multi-core:

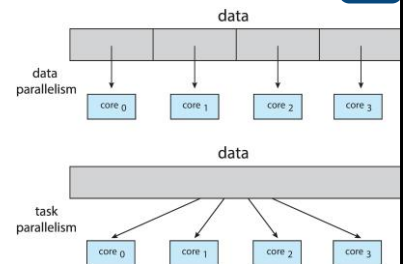


7

Programação Multicore

Tipos de paralelismo

- **Paralelismo de dados:** distribui subconjuntos dos mesmos dados em vários núcleos, mesma operação em cada um
- **Paralelismo de tarefas:** distribuindo threads entre núcleos, cada thread executando uma operação exclusiva



8

Lei de Amdahl

- Identifica ganhos de desempenho com a adição de núcleos adicionais a uma aplicação que possui componentes seriais e paralelos

- S é a porção serial
- N é número de núcleos de processamento

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

- Ou seja, se a aplicação é 75% paralela / 25% serial, passar de 1 para 2 núcleos resulta em speedup de 1,6 vezes
- À medida que N se aproxima do infinito, a aceleração se aproxima de $1 / S$

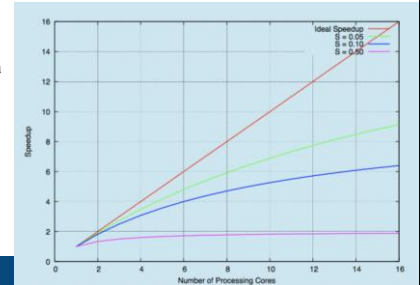


9

Lei de Amdahl

A parte serial de um aplicativo tem um efeito desproporcional sobre o desempenho obtido pela adição de núcleos

Mas a lei leva em conta os sistemas multicore contemporâneos?



10

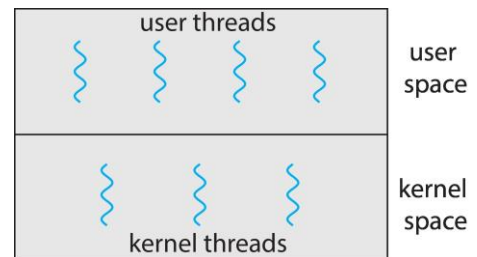
Threads de usuário e threads de kernel

- Threads de usuário** - gerenciamento feito pela biblioteca de threads em nível de usuário
- Três bibliotecas de threads principais:
 - POSIX **Pthreads** – iremos entender essa
 - Windows threads – pesquisar no material ou internet
 - Java threads – pesquisar no material ou internet (se ficar lento, já sabe)
- Threads do kernel** - suportado pelo Kernel
- Exemplos – praticamente todos os sistemas operacionais de uso geral utilizam:
 - Windows, Linux, Mac OS X, iOS e Android



11

Threads do usuário e do kernel



12

Bibliotecas de threads

- **Biblioteca de threads** fornece ao programador uma API para criar e gerenciar threads
- Duas formas principais de implementar
 - Biblioteca inteiramente no espaço do usuário
 - Biblioteca no nível do kernel suportada pelo SO



13

Pthreads

- Pode ser fornecido como nível de usuário ou nível de kernel
- Uma API padrão POSIX (IEEE 1003.1c) para criação e sincronização de threads
- Especificação, não implementação
- API especifica o comportamento da biblioteca de threads, a implementação depende do desenvolvimento da biblioteca
- Comum em sistemas operacionais UNIX (Linux e Mac OS X)
 - Aceita no Windows via IDE - CodeBlocks, CLion, Qt



14

Exemplo de Pthreads

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

#include <stdlib.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 */

    /* set the default attributes of the thread */
    pthread_attr_t 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 execute 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);
}
```



15

Código Pthreads para unir 10 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);
```



16

Threading implícita

- Com o número de threads aumenta, a manutenção do programa é mais difícil com threads explícitas
- Criação e gerenciamento de threads feito por compiladores e bibliotecas de tempo de execução em vez de programadores
- Cinco métodos explorados
 - Thread Pools (falaremos)
 - Fork-Join (pesquisar)
 - OpenMP (falaremos)
 - Grand Central Dispatch (pesquisar)
 - Intel Threading Building Blocks (pesquisar)

17

Pools de threads

- Criar vários threads em um pool onde eles aguardam o trabalho
- Vantagens:
 - Normalmente, um pouco mais rápido para atender a uma solicitação com uma thread existente do que criar uma nova thread
 - Permite que o número de threads no(s) aplicativo(s) seja vinculado ao tamanho do pool
 - Separar a tarefa a ser executada da mecânica de criação da tarefa permite diferentes estratégias para executar a tarefa
 - ou seja, as tarefas podem ser agendadas para serem executadas periodicamente
- A API do Windows oferece suporte a pools de threads:

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

18

Java Thread Pools

- Três métodos para criar pools de threads na classe Executors:

```
static ExecutorService newSingleThreadExecutor()  
static ExecutorService newFixedThreadPool(int size)  
static ExecutorService newCachedThreadPool()  
  
import java.util.concurrent.*;  
  
public class ThreadPoolExample  
{  
    public static void main(String[] args) {  
        int numTasks = Integer.parseInt(args[0].trim());  
  
        /* Create the thread pool */  
        ExecutorService pool = Executors.newCachedThreadPool();  
  
        /* Run each task using a thread in the pool */  
        for (int i = 0; i < numTasks; i++)  
            pool.execute(new Task());  
  
        /* Shut down the pool once all threads have completed */  
        pool.shutdown();  
    }  
}
```

19

OpenMP

- Conjunto de diretivas do compilador e uma API para C, C++, FORTRAN
 - Fornece suporte para programação paralela em ambientes de memória compartilhada
 - Identifica regiões paralelas – blocos de código que podem ser executados em 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 */

 #pragma omp parallel
 Crie tantos threads quanto houver núcleos
 return 0;
}
```

20

## OpenMP

- Execute o loop for em paralelo

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

21

## Problemas com Threads

- Semântica das chamadas do sistema fork() e exec()
- Manipulação de sinais
  - Síncrono e assíncrono
- Cancelamento de thread de destino
- Armazenamento local de thread
- Ativações do Escalonador

22

## Cancelamento de Thread

- Encerrando um thread antes que ele tenha terminado
- O thread a ser cancelado é o thread de destino
- Duas abordagens gerais:
  - O cancelamento assíncrono encerra o thread de destino imediatamente
  - O cancelamento adiado permite que o thread de destino verifique periodicamente se 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);
/* wait for the thread to terminate */
pthread_join(tid, NULL);
```

23

## Armazenamento local de thread

- **Thread-local storage (TLS)** permite que cada thread tenha sua própria cópia dos dados
- Útil quando você não tem controle sobre o processo de criação de threads (ou seja, ao usar um pool de threads)
- Diferente das variáveis locais
  - Variáveis locais visíveis apenas durante a invocação de uma única função
  - TLS visível em invocações de função
- Semelhante a dados estáticos
  - TLS é exclusivo para cada thread

24

## Ativações do Escalonador



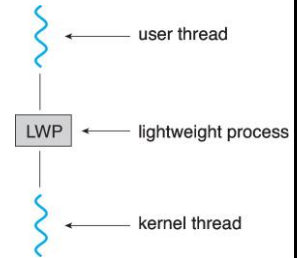
- Os modelos M:M e de dois níveis exigem comunicação para manter o número apropriado de threads do kernel alocados para o aplicativo
- Normalmente, use uma estrutura de dados intermediária entre threads do usuário e do kernel – **lightweight process (LWP)**
  - Parece ser um processador virtual no qual o processo pode agendar a execução do thread do usuário
  - Cada LWP anexado ao thread do kernel
  - Quantos LWPs criar?

25

## Ativações do Escalonador



- As ativações do escalonador fornecem upcalls - um mecanismo de comunicação do kernel para o manipulador de upcall na biblioteca de threads
- Essa comunicação permite que um aplicativo mantenha os número correto de threads do kernel



26



Universidade do Vale do Itajaí  
Escola Politécnica  
Engenharia de Computação

## Threads e Concorrência

27