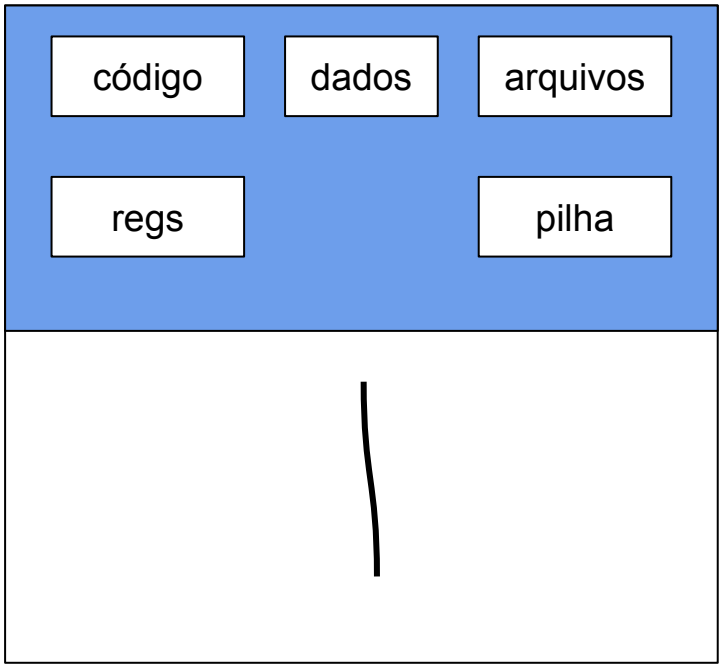
Sistemas Operacionais

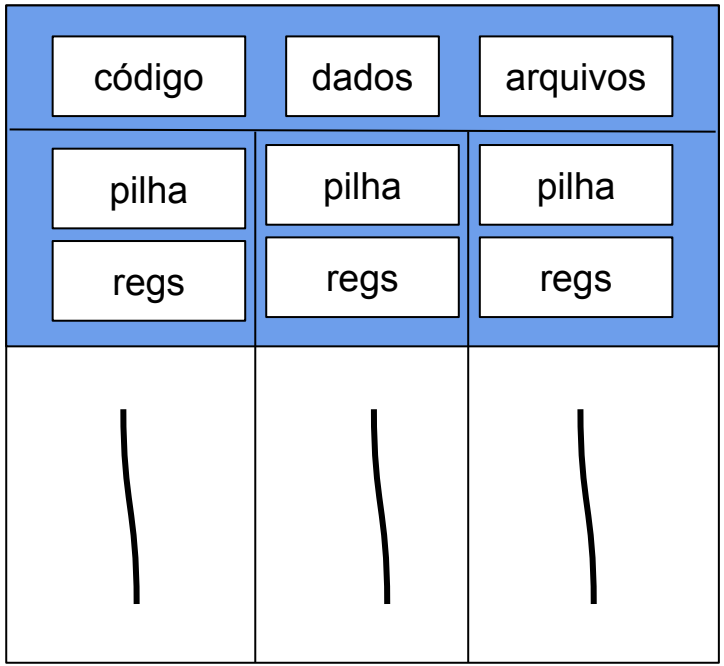
Threads

Processos Leves

Threads



Unidade básica de utilização de CPU.



Processo multithreads

Identificar tarefas

Balancear a carga

Desafios na programação

Separação dos dados

Dependência de dados

Testes e depuração

Tipos de Paralelismo

Paralelismo de Tarefas

Paralelismo de Dados

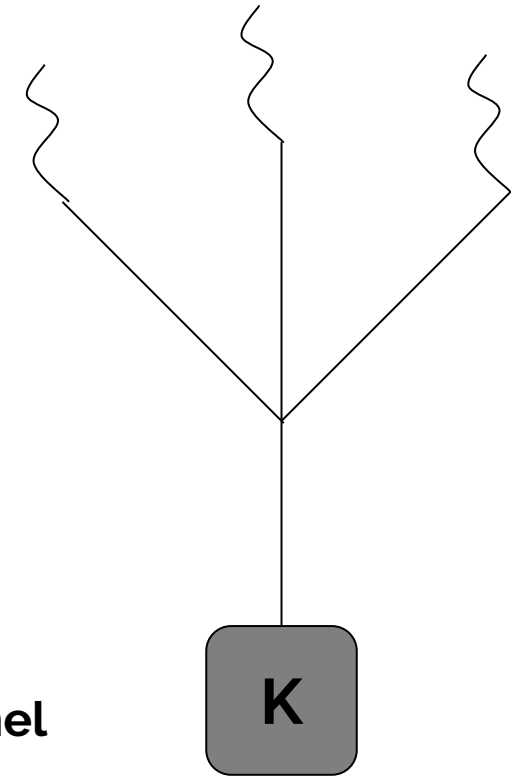
Nível de Usuário

Suportes para threads

Nível de Kernel

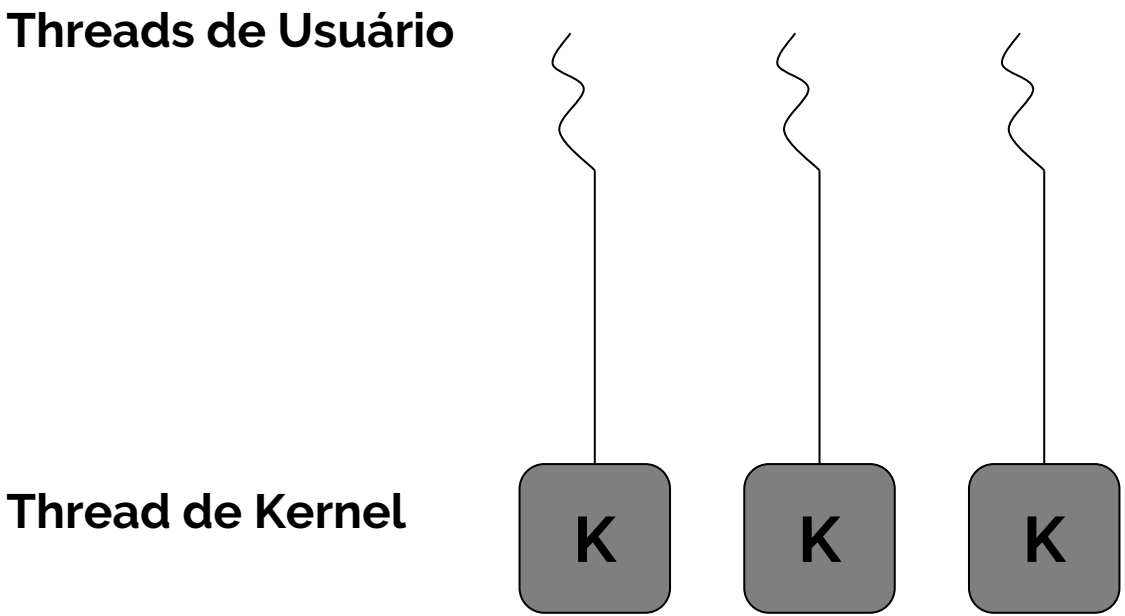
Modelo Many-to-One

Threads de Usuário



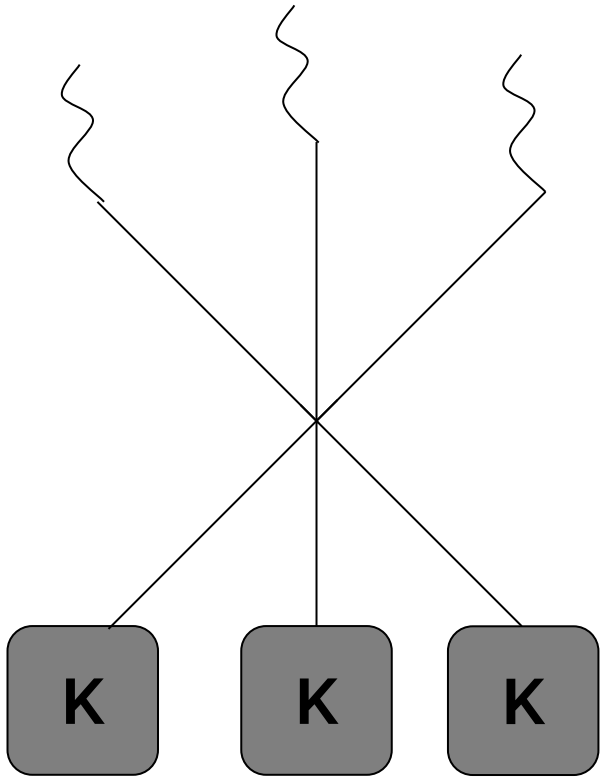
Thread de Kernel

Modelo
One-to-One



Modelo
Many-to-Many

Threads de Usuário



Threads de Kernel

Bibliotecas threads

Pthreads, Java

Threads Windows

PThreads

POSIX threads

PThreads refere-se ao padrão POSIX (IEEE 1003.1c) que define um padrão para criação e sincronização de threads.

É uma especificação para comportamento e não uma implementação.

Biblioteca pthread.h

```
struct pthread_t
pthread_attr_init();
pthread_create();
pthread_join();
pthread_kill();
pthread_exit();
```

POSIX threads

```
void *thr_func(void *param)
{
   ...
   return NULL;
}


int main(int argc, char *argv[]) {
   pthread_t tid;
   pthread_attr_t attr;

   ...

   pthread_attr_init(&attr);
   pthread_create(&tid,&attr,thr_func,NULL);
   pthread_join(tid,NULL);

   ...

   return 0;
}
```

POSIX threads - ex01.c

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

int sum;
void *runner(void *param);
int main(int argc, char *argv[]) {
   pthread_t tid;
   pthread_attr_t attr;
   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;
   }

   pthread_attr_init(&attr);
   pthread_create(&tid,&attr,runner,argv[1]);
   pthread_join(tid,NULL);
   printf("sum = %d\n",sum);
   return 0;
}
```

```c
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;
   for (i = 1; i <= upper; i++)
      sum += i;
   return NULL;
}
```

```
Para compilar:
      gcc -o ex01 ex01.c -lpthread

Para executar:
      ./ex01
```

Threads Windows

Threads Windows

As threads no windows são muito semelhantes às pthreads em muitos aspectos.

Threads Windows

```
HANDLE CreateThread();
DWORD WaitForSingleObject();
DWORD WaitForMultipleObjects();
BOOL CloseHandle();
DWORD SuspendThread();
DWORD ResumeThread();
VOID ExitThread();
BOOL TerminateThread();
```

Threads Windows

```
#include <windows.h>

DWORD WINAPI thr_func(LPVOID Param) {
   ...
   return 0;
}

int main(int argc, char *argv[]) {
   DWORD ThreadId;
   HANDLE ThreadHandle;
   ...
   ThreadHandle = CreateThread(
      NULL,      /* default security attributes */
      0,         /* default stack size */
      Thr_func,  /* thread function */
      NULL,      /* parameter to thread function */
      0,         /* default creation flags */
      &ThreadId);

   if (ThreadHandle != NULL) {
      WaitForSingleObject(ThreadHandle,INFINITE);
      CloseHandle(ThreadHandle);
      printf("sum = %d\n",Sum);
   }
}
```

Threads Windows

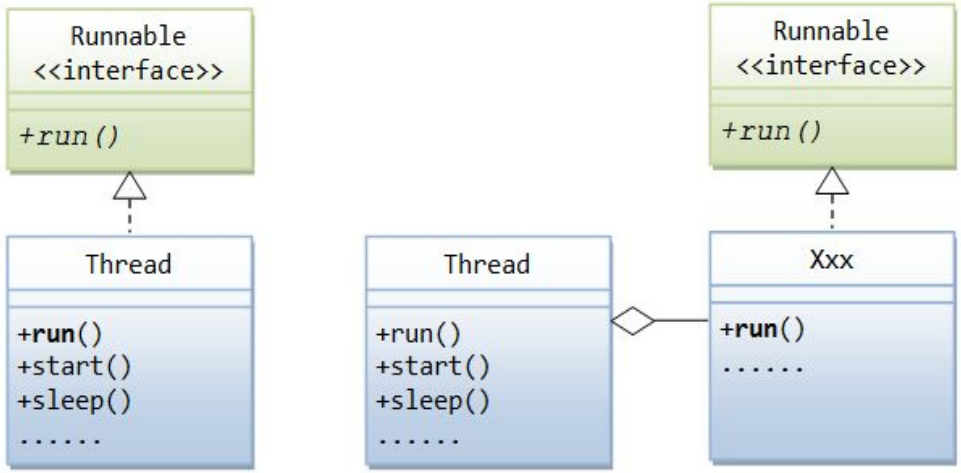
```
#include <windows.h>
#include <stdio.h>

DWORD Sum; /* data is shared by the thread(s) */

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 is required\n");
      return -1;
   }
```

```
   Param = atoi(argv[1]);

   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }

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

   if (ThreadHandle != NULL) {
      WaitForSingleObject(ThreadHandle,INFINITE);
      CloseHandle(ThreadHandle);
      printf("sum = %d\n",Sum);
   }

}
```

Threads no Java

Há duas maneiras de criar threads no Java

1. Classe herdada de Thread
2. Interface Runnable



Threads no Java
Interface Runnable

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

Threads Windows

```
public class HelloPrinter implements Runnable {
   String nome;
   public HelloPrinter(String nome){
      this.nome = nome;
   }

   @Override
   public void run() {
      for (int i = 0; i < 10; i++)
         System.out.println(nome + " diz: olá" );
      }
}
```

```
public class Principal {
   public static void main(String[] args) {
      Thread []v = new Thread[10];

      for (int i = 0; i < 10; i++){
         HelloPrinter h= new HelloPrinter("Thread " + i);
         v[i] = new Thread(h);
      }

      for (int i = 0; i < 10; i++)
         v[i].start();

      System.out.println("Terminou a main!");
   }
}
```

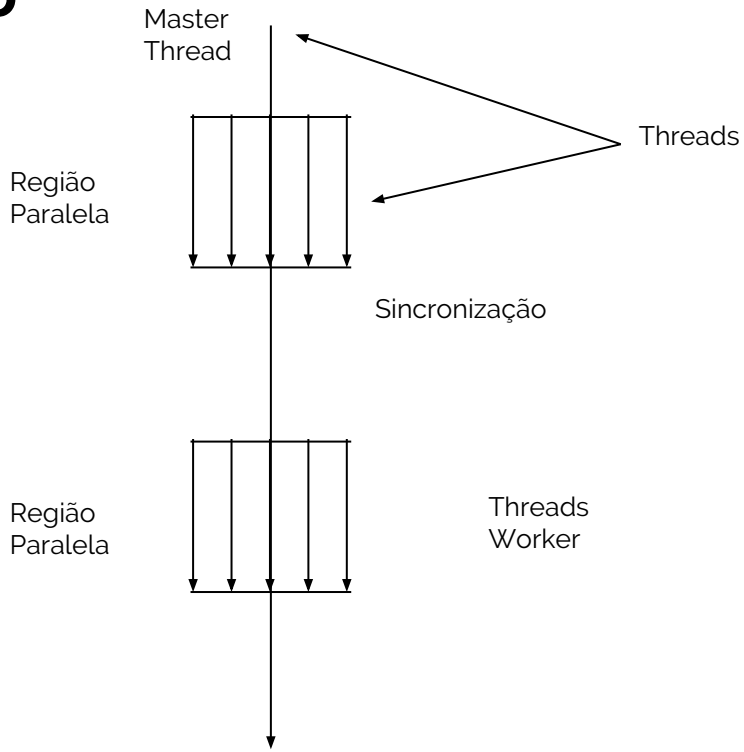
OpenMP

OpenMP

OpenMP é uma interface de programação (API - Application Programming Interface) que fornecem suporte de programação paralela em ambientes de memória compartilhada.

Modelo de Execução

Modelo Fork-Join



Região Paralela

Uma região paralela é um bloco de código que será executado por múltiplas threads.

```
código sequencial

#pragma omp parallel {

} /* fim da região paralela */

código sequencial
```

OpenMP
ex02.c

```
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
      #pragma omp parallel
      {
            printf("I am a parallel region.");
      }

      return 0;
}
```

```
Para compilar:
      gcc -fopenmp ex02.c -o ex02

Para executar:
      ./ex02
```

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

int main(int argc, char *argv[])
{
   int i, id;

   #pragma omp parallel for
      for (i = 0; i < 8; i++)
      {
        id = omp_get_thread_num();
        printf("%d - %d  \n", id, i);
      }
   return 0;
}
```

```
> gcc ex03.c -o ex03 -fopenmp
> ./ex03
0 - 0
0 - 1
3 - 6
3 - 7
2 - 4
2 - 5
1 - 2
1 - 3
```

For paralelo