

Objetivos

A programação concorrente para o
processamento de alto desempenho

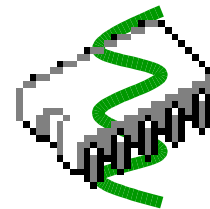
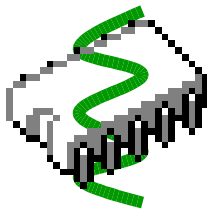
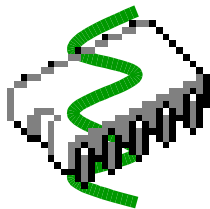
Exploração do poder computacional de
aglomerados

Ferramentas padrões para o processamento
paralelo e distribuído

Objetivos

Explorar a **concorrência intra-nó**

Memória



Objetivos

Explorar a **concorrência intra-nó**

Questões de **sincronização**

Estudo de caso: **threads POSIX**

Sumário

A programação concorrente

Tarefas e Sincronização

Memória compartilhada

Threads POSIX

Memória distribuída

Message Passing Interface - MPI

Uso conjunto

Problemas associados

Java

Memória compartilhada

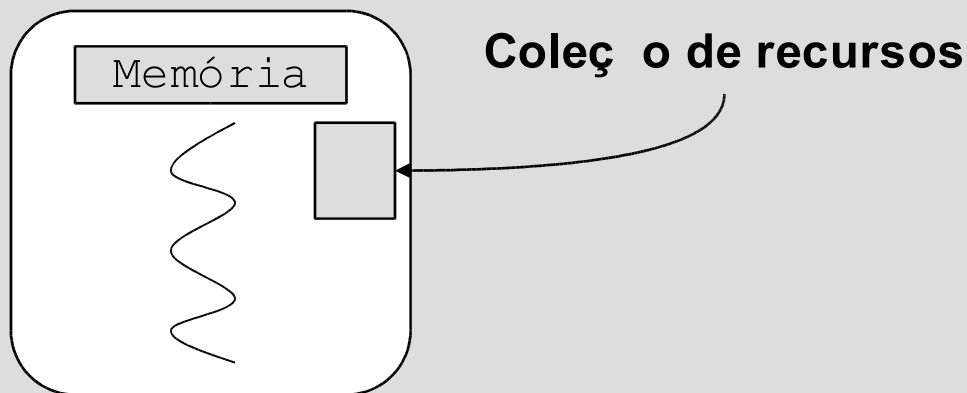
Threads POSIX

- Conceito de thread
- Motivação de uso
- Ciclo de vida de uma thread
- Pthreads básico
- Manipulação de threads
- Compartilhamento de memória
- Thread vs. tarefa
- Modelos de threads

Memória compartilhada

Conceito de thread

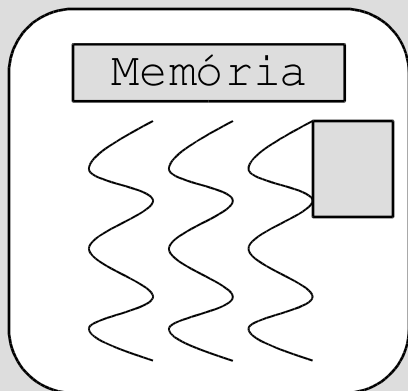
Fluxo de execução interno a um processo



Memória compartilhada

Conceito de thread

Fluxo de execução interno a um processo



Multithread:
múltiplos fluxo de
execução

**Compartilhamento de memória
e demais recursos do processo**

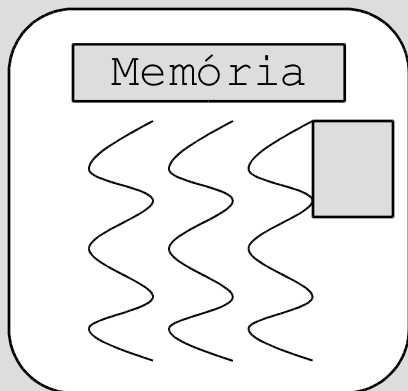
Memória compartilhada

Conceito de thread

Processo leve

Compartilhamento de
memória e recursos

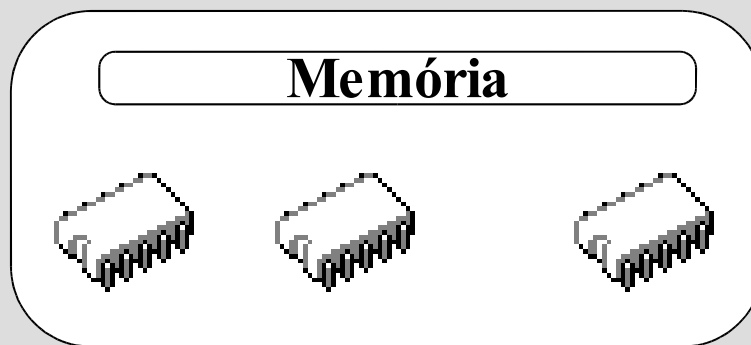
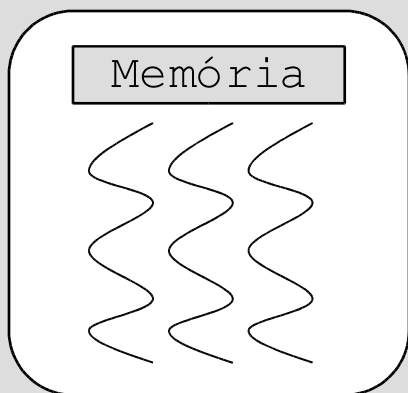
**Escalonamento menos
oneroso**



Memória compartilhada

Motivação de uso de thread

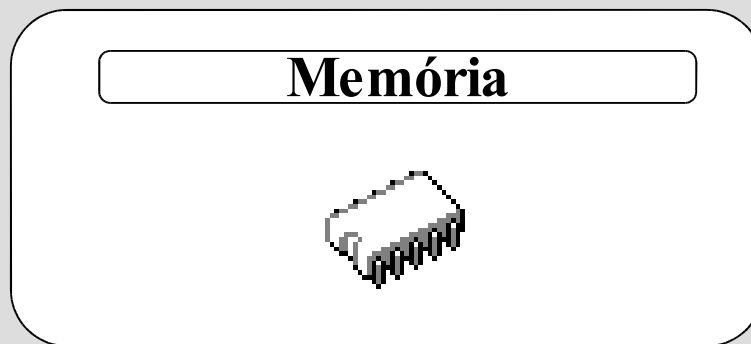
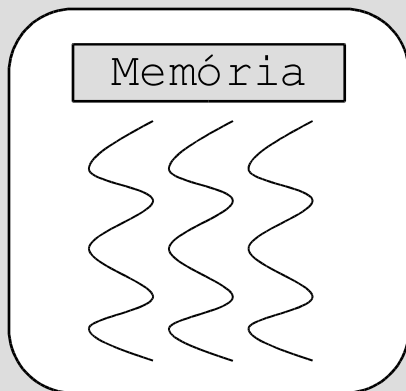
Reflete a arquitetura de um nó SMP



Memória compartilhada

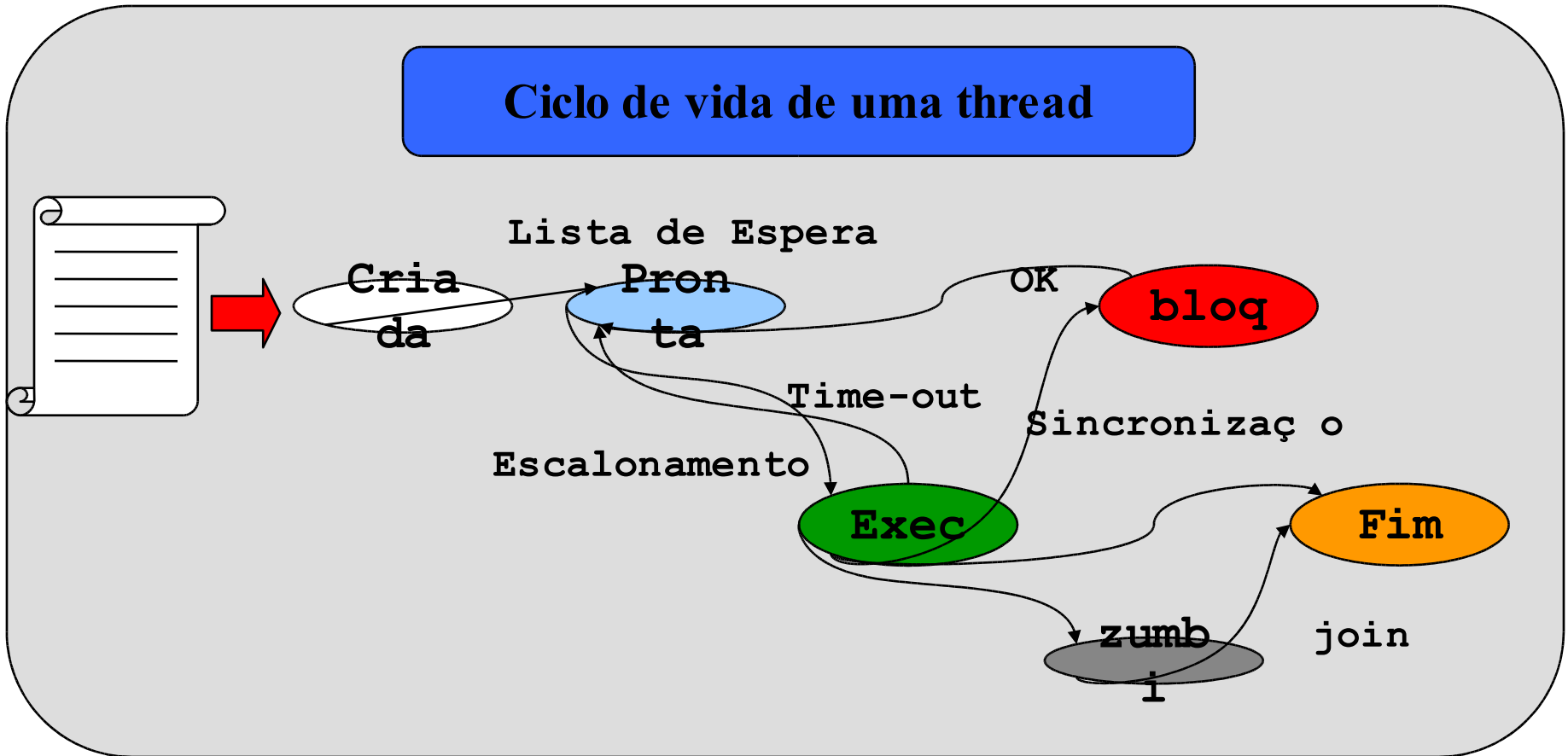
Motivação de uso de thread

Aumenta a disponibilidade do processador para o programa em um monoprocessador



Memória compartilhada

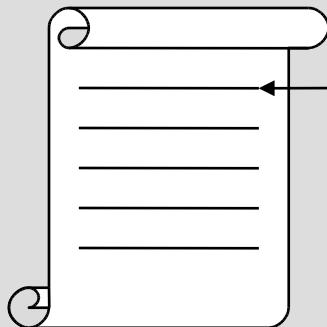
Ciclo de vida de uma thread



Memória compartilhada

Pthreads básico

Utilizando a biblioteca pthread no Linux



prog.c

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

Define:

```
tipos: pthread      t
```

```
serviços: pthread
```

Memória compartilhada

Pthreads básico

Utilizando a biblioteca pthread no Linux

Compilando:

```
gcc -I/usr/include -c prog c
```

prog c

```
01100110  
11101100  
00100100  
11110011  
11110001
```

prog o

Memória compartilhada

Pthreads básico

Utilizando a biblioteca pthread no Linux

```
01100110
11101100
00100100
11110011
11110001
```

prog.o

Link-edição:

```
gcc -L/usr/lib -lpthread prog.o
```

```
11001100
00110011
11001100
01100110
11101100
00100100
11110011
11110001
```

prog

Memória compartilhada

Manipulação de threads

Criação:

```
pthread_create ( pthread_t *thid,  
                pthread_attr_t *atrib,  
                void * (*func) (void *),  
                void *args );
```

Memória compartilhada

Manipulação de threads

Criação:

```
pthread_create ( pthread_t *thid,  
                pthread_attr_t *atrib,  
                void * (*func) (void *),  
                void *args );
```

func

função que contém o serviço a ser executado
pela thread

Memória compartilhada

Manipulação de threads

Criação:

```
pthread_create ( pthread_t *thid,  
                pthread_attr_t *atrib,  
                void * (*func) (void *),  
                void *args );
```

func

```
void *servico ( void *in ) { }
```

Memória compartilhada

Manipulação de threads

Criação:

```
pthread_create ( pthread_t *thid,  
                pthread_attr_t *atrib,  
                void * (*func) (void *),  
                void *args );
```

args

dados passados à thread criada

Memória compartilhada

Manipulação de threads

Criação:

```
pthread_create ( pthread_t *thid,  
                pthread_attr_t *atrib,  
                void * (*func) (void *),  
                void *args );
```

thid

retorna um identificador da thread criada

Memória compartilhada

Manipulação de threads

Criação:

```
pthread_create ( pthread_t *thid,  
                pthread_attr_t *atrib,  
                void * (*func) (void *),  
                void *args );
```

atrib

atributos de execução da thread

Memória compartilhada

Manipulação de threads

Criação: Exemplo

```
main() {  
    pthread_t tid;  
    char *str = "Oi mundo";  
    pthread_create( &tid, NULL,  
                  OiMundo, str );  
}
```

```
void *OiMundo(void *in){  
    char *str = (char *)in;  
    printf( "%s\n", str);  
}
```

Memória compartilhada

Manipulação de threads

Sincronizando com o término:

```
pthread_join ( pthread_t *thid,  
              void **res );
```

Aguarda o término da thread **thid**
e coloca o retorno em **res**

Memória compartilhada

Manipulação de threads

Sincronizando

Exemplo:

```
main() {  
    pthread_t tid;  
    char *str = "Oi mundo";  
    pthread_create( &tid, NULL,  
                  OiMundo, str );  
    //trecho concorrente  
    pthread_join( tid, NULL );  
}
```

Memória compartilhada

Manipulação de threads

Atributos de threads

```
pthread_t create ( pthread_t *thid,  
                  pthread_attr_t *atrib,  
                  void * (*func) (void *),  
                  void *args );
```

Memória compartilhada

Manipulação de threads

Atributos de threads

`pthread_attr_t`

É uma estrutura de dados cujos campos descrevem características de execução das threads.

Manipulada através de funções

Principais campos:

independente ou não
escalonamento
escopo
(memória, ...)

Memória compartilhada

Manipulação de threads

Atributos de threads

`pthread_attr_t`

```
int pthread_attr_t init( pthread_attr_t *attr );
```

Permite inicializar todos atributos com os valores default.

Memória compartilhada

Manipulação de threads

Atributos de threads

`pthread_attr_t`

```
int pthread_attr_t pthread_attr_t::setdetachedstate (
    pthread_attr_t *attr, int st );
```

Inicializa o campo `detachedstate` do descritor **attr** com o valor informado em **st**.

Valores:

PTHREAD_JOINABLE (default)
PTHREAD_DETACHED

Memória compartilhada

Manipulação de threads

Atributos de threads

`pthread_attr_t`

```
int pthread_attr_t pthread_attr_setschedpolicy (
    pthread_attr_t *attr, int pol );
```

Inicializa o campo política de escalonamento de **attr**
com
valor informado em **pol**.

Valores:

SCHED_FIFO, SCHED_RR
SCHED_OTHER

Memória compartilhada

Manipulação de threads

Atributos de threads

`pthread_attr_t`

```
int pthread_attr_t pthread_attr_setscope (  
    pthread_attr_t *attr, int esc );
```

Inicializa o campo escopo de escalonamento de **attr**
com
valor informado em **esc**.

Valores:

PTHREAD_SCOPE_SYSTEM
PTHREAD_SCOPE_PROCESS

Memória compartilhada

Manipulação de threads

Atributos de threads

`pthread_attr_t`

O escopo

Thread sistema:

São manipuladas, no que diz respeito ao escalonamento da CPU, como processos.

Cada thread sistema tem um quanta de tempo no processador

Memória compartilhada

Manipulação de threads

Atributos de threads

`pthread_attr_t`

O escopo

Thread usuário:

São escalonadas, para execução, no interior do processos.

O processo, ao receber um quantum, escolhe uma thread para executar dentre as threads usuário,

Memória compartilhada

Manipulação de threads

Atributos de threads

O escopo

Questão:

Um programa em execução em um monoprocessador é composto de 2 threads. Em um intervalo de tempo de 100 segundos, com quanta de 1 segundo, quantas vezes o processo recebeu a CPU? (Existem outros 9 processos executando)

Se thread **usuário**:
tempo total / número de processos
 $100/10 = 10$ segundos

Memória compartilhada

Manipulação de threads

Atributos de threads

O escopo

Questão:

Um programa em execução em um monoprocessador é composto de 2 threads. Em um intervalo de tempo de 100 segundos, com quanta de 1 segundo, quantas vezes o processo recebeu a CPU? (Existem outros 9 processos executando)

Se thread **sistema**:
tempo total / número de threads
 $100/11 = 9,1$ segundos

Memória compartilhada

Manipulação de threads

Atributos de threads

O escopo

Questão:

Um programa em execução em um monoprocessador é composto de 2 threads. Em um intervalo de tempo de 100 segundos, com quanta de 1 segundo, quantas vezes o processo recebeu a CPU? (Existem outros 9 processos executando)

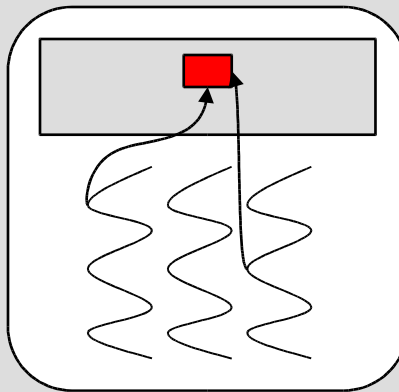
Se thread **sistema**:
logo, o dito processo executou
 $2 * 9,1 = 18,2$ segundos

Memória compartilhada

Compartilhamento de memória

Sessão crítica:

Competição pelos dados



Controla a semântica evitando que threads acessem um dado ao mesmo tempo

Uma sessão crítica é o trecho de código de uma thread que acessa um dado global.

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

```
int x = 313;
```

```
// thread A
```

```
a = x;
```

```
a = a + 1;
```

```
x = a;
```

```
// thread B
```

```
b = x;
```

```
b = b - 1;
```

```
x = b;
```

A	a = x	313
A	a = a + 1	313
B	b = x	313
A	x = a	314
B	b = b - 1	314
B	x = b	312

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

```
int x = 313;
```

```
// thread A
```

```
a = x;
```

```
a = a + 1;
```

```
x = a;
```

```
// thread B
```

```
a = x;
```

```
a = a - 1;
```

```
x = a;
```

Proteger o acesso
às sessões
críticas

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Mutex

Proteger o acesso
às sessões
críticas

O uso de um mutex garante que apenas uma das threads execute código na sessão crítica.

Princípio de obter um passe

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Mutex

Uso de mutex

```
pthread_mutex_t
```

tipo de dado mutex

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Mutex

Uso de mutex

```
pthread_mutex_t      tipo de dado mutex
int pthread_mutex_init(
    pthread_mutex_t *m
    pthread_mutexattr_t *attrib );
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Mutex

Uso de mutex

Requisita direito de entrar:

```
int pthread_mutex_lock ( pthread_mutex_t *m );
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Mutex

Uso de mutex

Requisita direito de entrar:

```
int pthread_mutex_lock ( pthread_mutex_t *m );
```

Informa que saiu:

```
int pthread_mutex_unlock ( pthread_mutex_t *m );
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Mutex

```
int x = 313;
```

```
// thread A
```

```
a = x;
```

```
a = a + 1;
```

```
x = a;
```

```
// thread B
```

```
b = x;
```

```
b = b - 1;
```

```
x = b;
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Mutex

```
int x = 313;  
pthread_mutex_t m;  
pthread_mutex_init(&m, NULL);
```

```
// thread A  
pthread_mutex_lock(&m);  
a = x;  
a = a + 1;  
x = a;  
pthread_mutex_unlock(&m);
```

```
// thread B  
pthread_mutex_lock(&m);  
b = x;  
b = b - 1;  
x = b;  
pthread_mutex_unlock(&m);
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

Permite um controle de sincronização
que leva em conta o estado da
execução

A thread é bloqueada
enquanto aguarda que uma
condição seja satisfeita

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

Permite um controle de sincronização
que leva em conta o estado da
execução

Uso típico:

compartilhamento de buffer

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

`pthread_cond_t`

define o tipo

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

```
pthread_cond_t      define o tipo
int pthread_cond_init(
    pthread_cond_t *c
    pthread_condattr_t *attrib );
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

Aguarda condição:

```
int pthread_cond_wait( pthread_cond_t *c );
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

Aguarda condição:

```
int pthread_cond_wait( pthread_cond_t *c );
```

Informa que a condição foi atendida:

```
int pthread_cond_signal( pthread_cond_t *c );
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

Aguarda condição:

```
int pthread_cond_wait( pthread_cond_t *c );
```

Informa que a condição foi atendida:

```
int pthread_cond_signal( pthread_cond_t *c );
```

```
int pthread_cond_broadcast( pthread_cond_t *c );
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Variável de Condição

```
for(;;) { // Produtor
    it =      ;
    pthread mutex lock(&m);
    WrBuffer(it);
    nb itens++;
    pthread cond signal(&c);
    pthread mutex unlock(&m);
}
```

```
for(;;) { // Consumidor
    pthread mutex lock(&m);
    while( nb itens <= 0 )
        pthread cond wait(&m, &c);
    it = InBuffer();
    nb itens--;
    pthread mutex unlock(&m);
}
```

Memória compartilhada

Sessão
crítica

Compartilhamento de memória

Semáforos

Não tem !!!

Memória compartilhada

Thread vs tarefa

Uma thread pode executar diversas tarefas

Cada sincronização faz com que a thread
bloqueie aguardando um dado

Questão de modelagem do programa

Memória compartilhada

Modelos de threads

1 : 1

Thread sistema

SMP

N : 1

Thread usuário

Mais leves

M : N

Misto

LWP em Solaris
Compromiss

o