Sistemas Operacionais

Processos

Processos

Processos

- Conjunto de recursos alocados para a execução de uma tarefa
- Unidade de contexto (Contêiner de recursos)

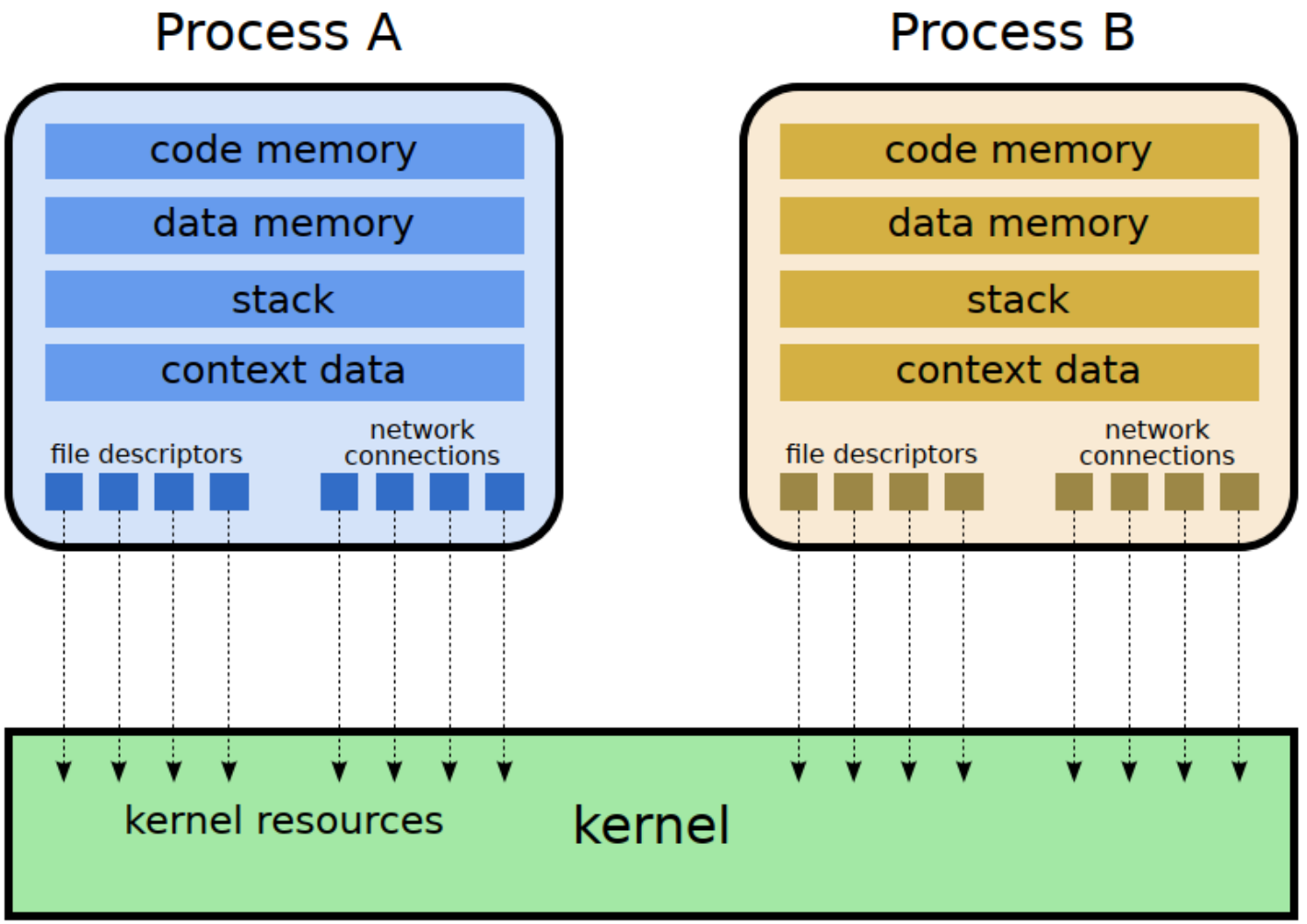
Processos

- Conjunto de recursos alocados para a execução de uma tarefa
- Unidade de contexto (Contêiner de recursos)

Recursos:

- Áreas de memória usada pela tarefa
- Dados
- Pilha
- Arquivos abertos
- Conexões de rede

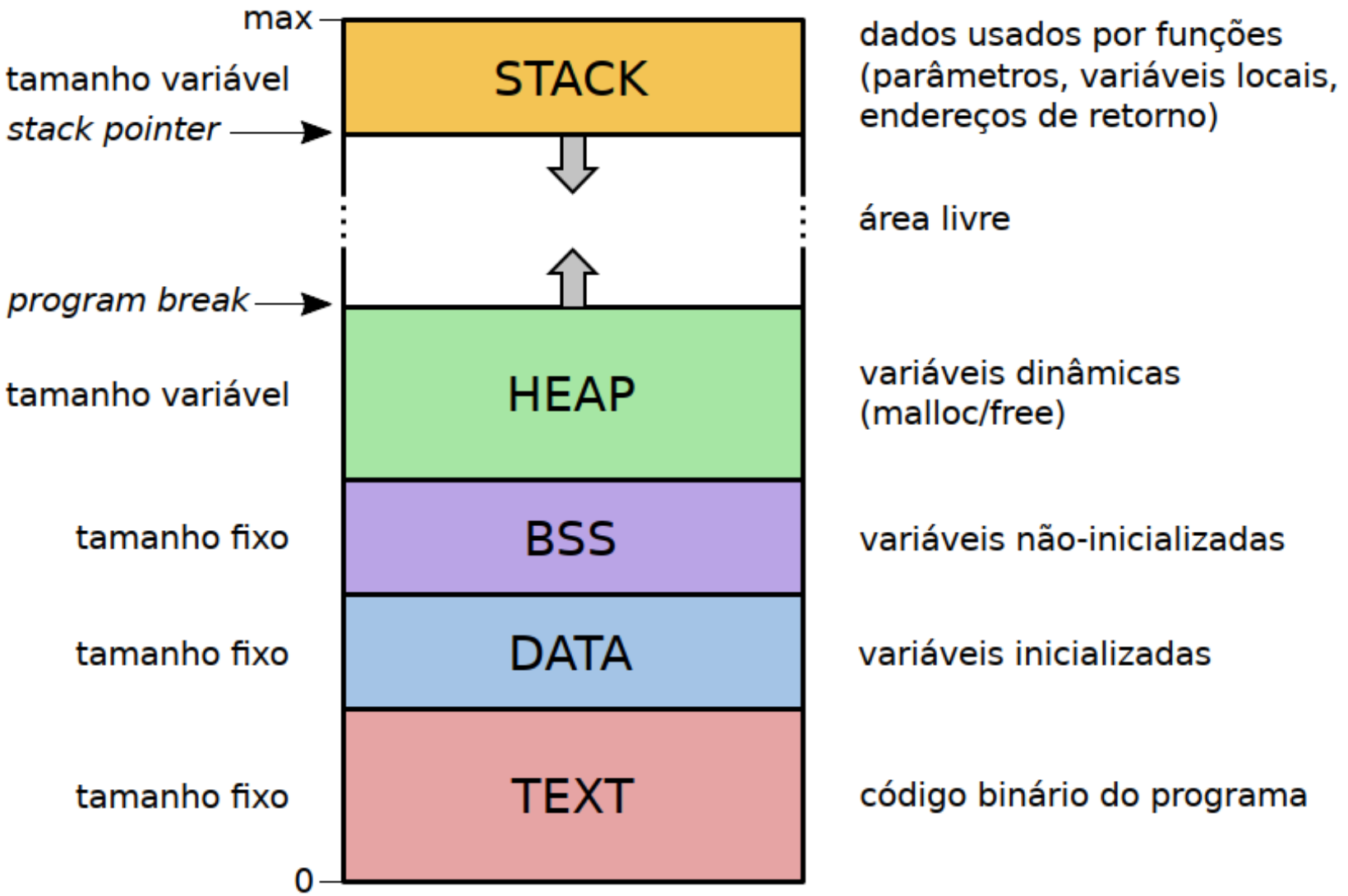
Processos


Process A
code memory
data memory
stack
context data
file descriptors
network connections
Process B
code memory
data memory
stack
context data
file descriptors
network connections
kernel resources
kernel

Modelo de memória dos processos

Cada processo é visto pelo sistema operacional como uma área de memória exclusiva que só ele e o núcleo do sistema podem acessar

Modelo de memória dos processos



PCB – *Process Control Block*

O núcleo do sistema operacional mantém descritores de processos, denominados PCBs (*Process Control Blocks*), para armazenar as informações referentes aos processos ativos

Criação de Processos

- Durante a vida do sistema, processos são criados e destruídos
- Essas operações são disponibilizadas às aplicações pelas ***chamadas de sistema***
- Cada sistema operacional tem suas próprias chamadas para a criação e remoção de processos

Criação de Processos

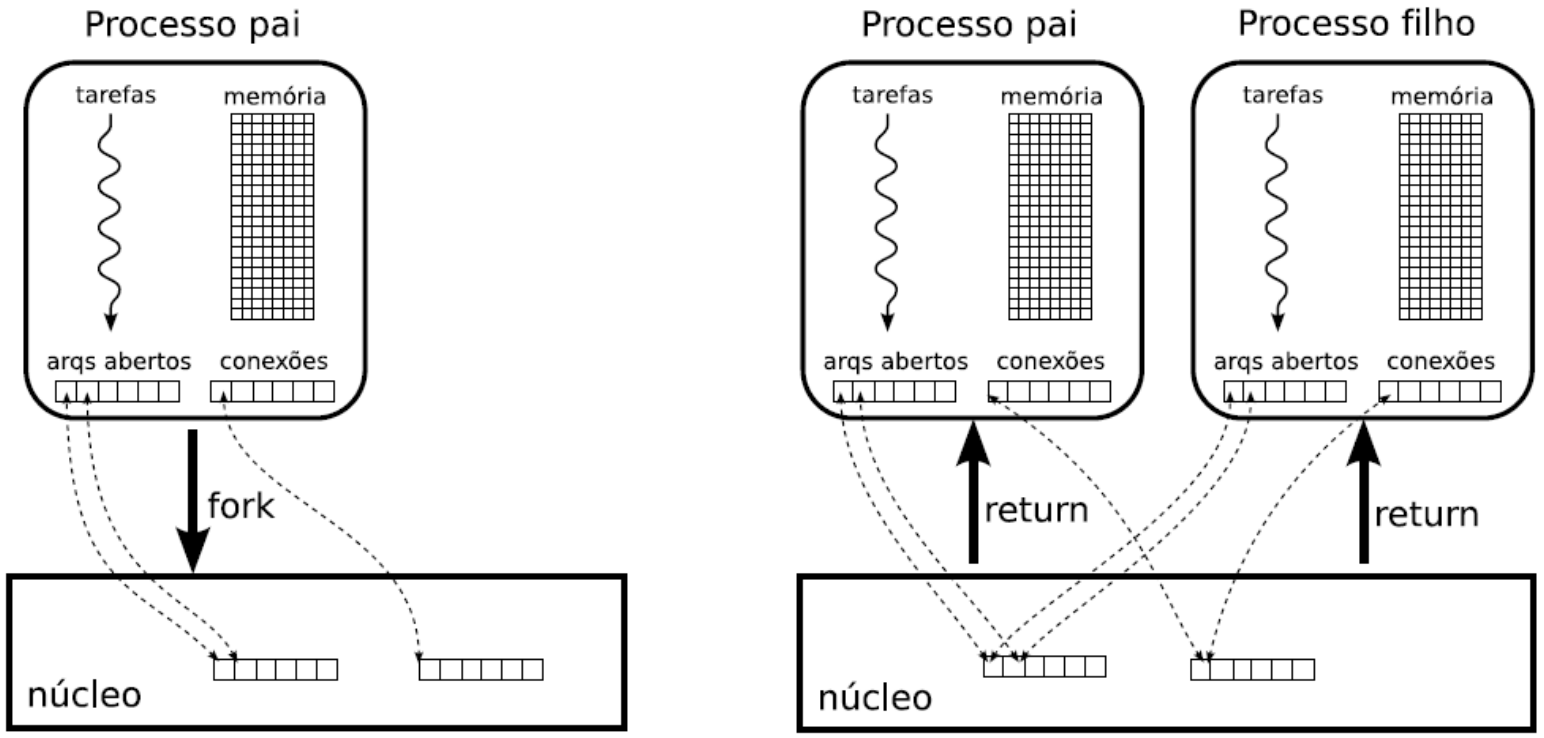
Ex.: FreeRTOS*

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask
);
```

* FreeRTOS denomina de Task

Criação de Processos

Ex.: Chamada de sistema *fork* do Unix



Criação de Processos

Ex.: Chamada de sistema *fork* do Unix

- Caso o processo filho deseje abandonar o fluxo de execução herdado do processo pai e executar outro código, poderá fazê-lo através da chamada de sistema execve

- Essa chamada substitui o código do processo que a invoca pelo código executável contido em um arquivo informado como parâmetro

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>

int main (int argc, char *argv[], char *envp[])
{
   int pid ;            /* identificador de processo */

   pid = fork () ;      /* replicação do processo */

   if ( pid < 0 )       /* fork não funcionou */
   {
      perror ("Erro: ") ;
      exit (-1) ;       /* encerra o processo */
   }
   else if ( pid > 0 )  /* sou o processo pai */
   {
      wait (0) ;        /* vou esperar meu filho concluir */
   }
   else                 /* sou o processo filho*/
   {
      /* carrega outro código binário para executar */
      execve ("/bin/date", argv, envp) ;
      perror ("Erro: ") ; /* execve não funcionou */
   }
   printf ("Tchau !\n") ;
   exit(0) ;            /* encerra o processo */
}
```

Árvore de Processos

```
1  init-+-aacraid
2       |-ahc_dv_0
3       |-atd
4       |-avaliacao_horac
5       |-bdflush
6       |-crond
7       |-gpm
8       |-kdm-+-X
9       |     `-kdm---kdm_greet
10      |-keventd
11      |-khubd
12      |-2*[kjournald]
13      |-klogd
14      |-ksoftirqd_CPU0
15      |-ksoftirqd_CPU1
16      |-kswapd
17      |-kupdated
18      |-lockd
19      |-login---bash
20      |-lpd---lpd---lpd
21      |-5*[mingetty]
22      |-8*[nfsd]
23      |-nmbd
24      |-nrpe
25      |-oafd
26      |-portmap
27      |-rhnsd
```

comando *pstree*

Threads

Threads

- Primeiros SO's → UMA tarefa por processo
- Para aplicações mais complexas, isto é um incoveniente. Por ex.:
  - Um editor de textos geralmente executa tarefas simultâneas sobre a mesma massa de dados (texto sob edição)
    - Edição
    - Formatação
    - Paginação
    - Verificação ortográfica
- Necessidade de executar mais de uma tarefa no mesmo contexto (processo)

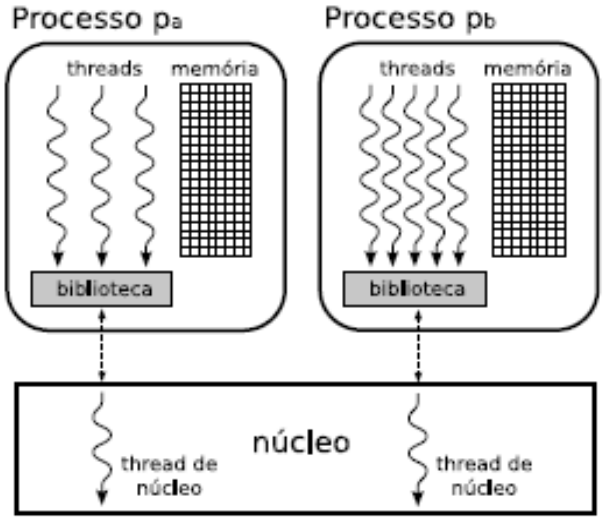
Threads

Definição:

Cada fluxo de execução do sistema

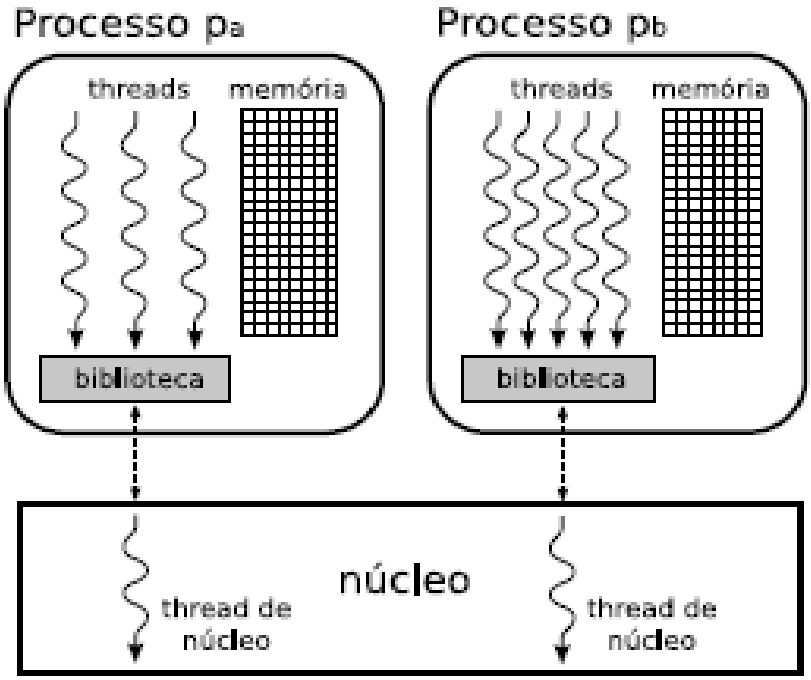
Thread de usuário: Executada dentro de um processo
Corresponde a uma tarefa a ser executada

Thread de núcleo: Fluxos de execução que são reconhecidos e executados pelo núcleo



Modelos de geração Multi-threads

N:1

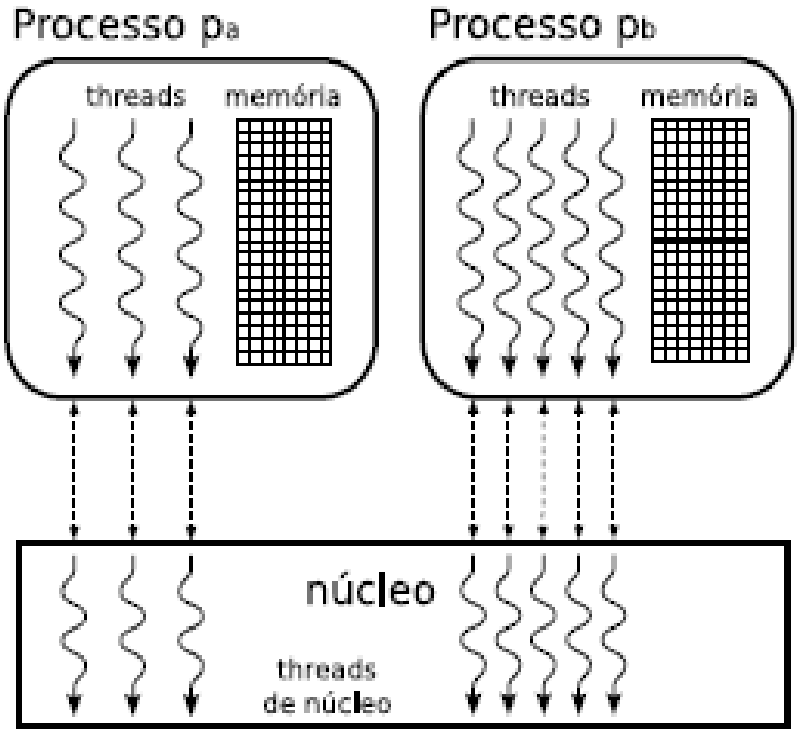


- Complexidade baixa (SO)
- Divisão injusta da CPU
- Problemas com E/S
- Threads do mesmo processo não executam em paraleleo

Ex.: biblioteca *GNU Portable Threads*

Modelos de geração Multi-threads

1:1

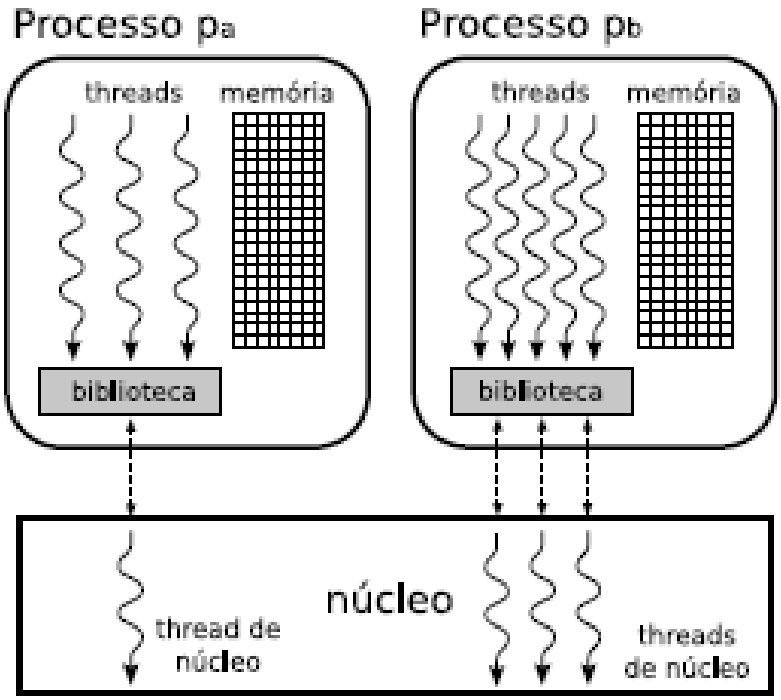


- Adequado para a maioria das situações usuais
- Baixa Escalabilidade:
  - Alta carga do SO (N>>1)

Ex.: Windows NT, UNIX

Modelos de geração Multi-threads

N:M, M≤N



- Complexidade moderada

Ex.: Solaris, projeto KSE (Kernel-Scheduled Entities) do FreeBSD

Modelos de geração Multi-threads

Modelo	**N:1**	**1:1**	**N:M**
Resumo	Todos os N *threads* do processo são mapeados em um único *thread* de núcleo	Cada *thread* do processo tem um *thread* correspondente no núcleo	Os N *threads* do processo são mapeados em um conjunto de M *threads* de núcleo
Local da implementação	bibliotecas no nível usuário	dentro do núcleo	em ambos
Complexidade	baixa	média	alta
Custo de gerência para o núcleo	nulo	médio	alto
Escalabilidade	alta	baixa	alta
Suporte a vários processadores	não	sim	sim
Velocidade das trocas de contexto entre *threads*	rápida	lenta	rápida entre *threads* no mesmo processo, lenta entre *threads* de processos distintos
Divisão de recursos entre tarefas	injusta	justa	variável, pois o mapeamento *thread*→ processador é dinâmico
Exemplos	GNU Portable Threads	Windows XP, Linux	Solaris, FreeBSD KSE

IEEE POSIX 1003.1c (*Pthreads*)

Se cada sistema operacional definir sua própria interface para a criação de *threads* $\Rightarrow$ problemas de portabilidade das aplicações

Pthreads $\Rightarrow$ Define uma interface padronizada para a criação e manipulação de threads na linguagem C

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

#define NUM_THREADS 5

/* cada thread vai executar esta função */
void *PrintHello (void *threadid)
{
   printf ("%d: Hello World!\n", (int) threadid);
   pthread_exit (NULL);
}

/* thread "main" (vai criar as demais threads) */
int main (int argc, char *argv[])
{
   pthread_t thread[NUM_THREADS];
   int status, i;

   /* cria as demais threads */
   for(i = 0; i < NUM_THREADS; i++)
   {
      printf ("Creating thread %d\n", i);
      status = pthread_create (&thread[i], NULL, PrintHello, (void *) i);

      if (status) /* ocorreu um erro */
      {
         perror ("pthread_create");
         exit (-1);
      }
   }

   /* encerra a thread "main" */
   pthread_exit (NULL);
}
```

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

#define NUM_THREADS 5

/* cada thread vai executar esta função */
void *PrintHello (void *threadid)
{
   printf ("%d: Hello World!\n", (int) threadid);
   pthread_exit (NULL);
}

/* thread "main" (vai criar as demais threads) */
int main (int argc, char *argv[])
{
   pthread_t thread[NUM_THREADS];
   int status, i;

   /* cria as demais threads */
   for(i = 0; i < NUM_THREADS; i++)
   {
      printf ("Creating thread %d\n", i);
      status = pthread_create (&thread[i], NULL, PrintHello, (void *) i);

      if (status) /* ocorreu um erro */
      {
         perror ("pthread_create");
         exit (-1);
      }
   }

   /* encerra a thread "main" */
   pthread_exit (NULL);
}
```