Paralelismo - *multitasking*

As aplicações disporem de ferramentas para tirar partido do paralelismo do sistema

1

Programação paralela

- Os processos permitem executar em paralelo (real ou em pseudoparalelismo) programas
- Quando executado num processo os programas não partilham o espaço de endereçamento, **não podem ter variáveis partilhadas***

  * Veremos mais tarde ser possível mas necessita de programação sistema
- Este isolamento entre processos, fundamental na operação normal, é indesejável em muitas situações de programação onde se pretende paralelismo sobre os mesmos dados

2

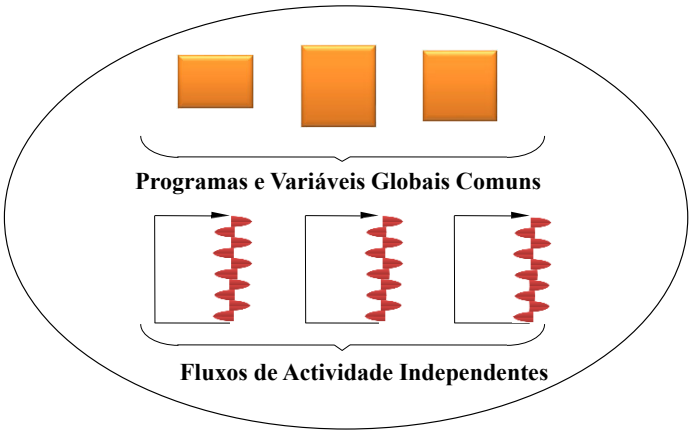
Exemplos de programação paralela

- Os exemplos óbvios são os algoritmos matemáticos intrinsecamente paralelos de calculo vectorial, operações algébricas, etc.
- Mas exemplos mais vulgares são a possibilidade de interactuar através de múltiplas janelas com o utilizador ou um servidor na Web que trata múltiplos pedidos de cliente em paralelo
- Um exemplo ilustrativo pode ser uma aplicação tipo *email* que quer manter uma actividade de interacção com o utilizador, mas em *background* trata eventos e executa tarefas de envio ou recepção de mensagens

3

O conceito de tarefa

- Fluxos independentes de execução, mas que partilham um contexto comum: as variáveis globais, o *heap* os ficheiros abertos



4

As potencialidades da programação paralela

- As tarefas começaram nas linguagens de programação, há muito tempo
- Estas pseudotarefas eram executadas em modo utilizador sem intervenção do núcleo, mas tinham diversas limitações
- Contudo, se a comutação for efectuada no núcleo o paralelismo pode ser semelhante ao obtido com a comutação dos processo

5

Definição de tarefa

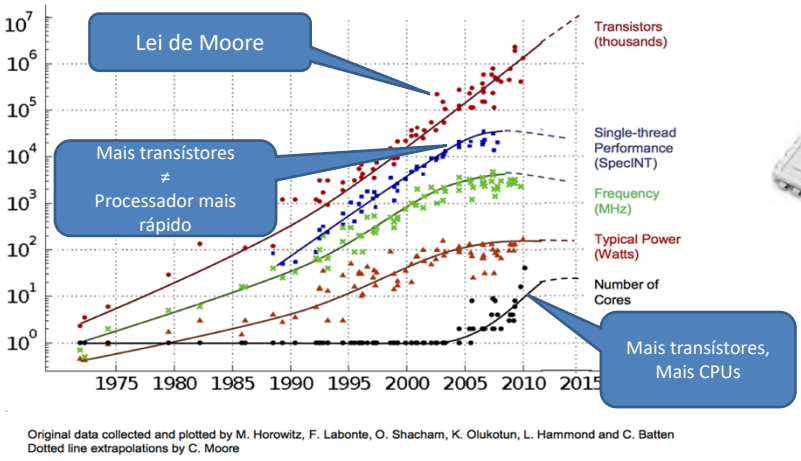
Uma tarefa é um fluxo de actividade que executa, no âmbito de um processo já existente, uma função do respectivo programa e partilha o mesmo espaço de endereçamento e os mesmos recursos do processo.

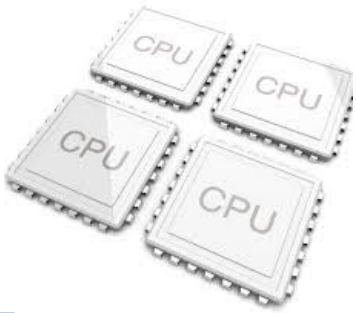
Em inglês designada task, thread ou lightweigh process

6

Programação paralela

Permite explorar de forma mais eficiente os processadores actuais com múltiplos *cores*





Como aproveitar?

7

Definição de tarefa

Uma tarefa é um fluxo de actividade que executa, no âmbito de um processo já existente, uma função do respectivo programa e partilha o mesmo espaço de endereçamento e os mesmos recursos do processo.

8

O que é partilhado entre tarefas do mesmo programa

- Código
- Variáveis globais
- Variáveis dinamicamente alocadas – *heap*
- Atributos do processo
  - PID, UID, Directório Corrente, Ficheiros Abertos,…

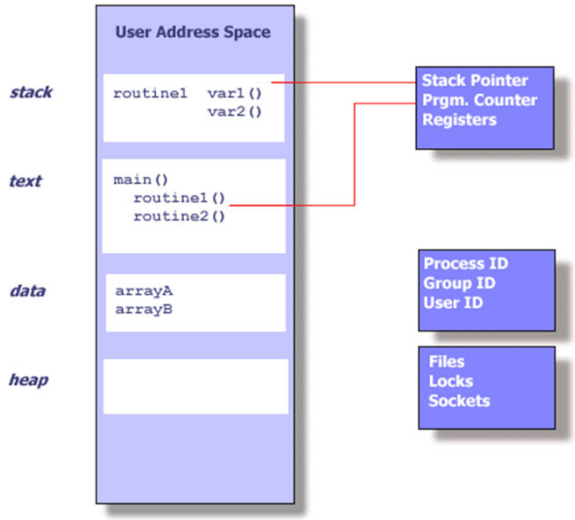
13

O que **não é** partilhado entre tarefas do mesmo processo

- Pilha (*stack*)
  - Cada tarefa tem naturalmente a sua pilha de execução
  - Mas atenção: não há isolamento entre pilhas uma vez que estão todas no mesmo espaço de endereçamento
- Estado dos registos do processador
  - Incluindo *instruction pointer*
- Atributos específicos da tarefa

14

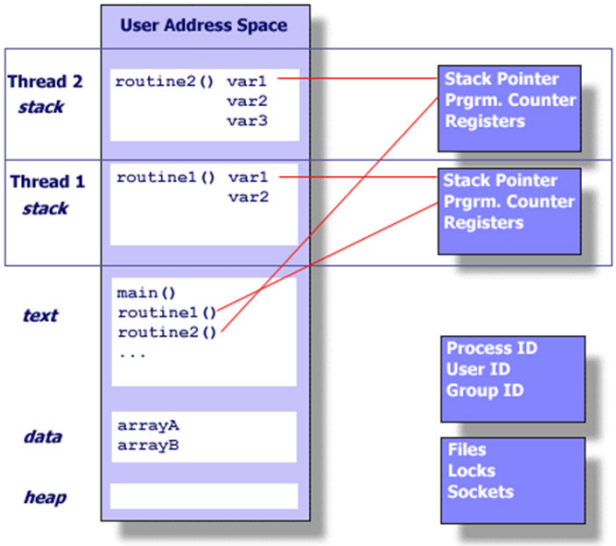
Um processo Unix



em Posix Programming threads - https://computing.llnl.gov/tutorials/pthreads/

15

Um processo unix com duas tarefas (threads)



em Posix Programming threads - https://computing.llnl.gov/tutorials/pthreads/

16

Objecto Tarefa

Propriedades

- Identificador
- Procedimento
- Prioridade

Operações

- CriarTarefa
- EliminarTarefa
- TransferirControlo
- EsperarTarefa
- LerEstado

17

Operações sobre Tarefas

IdTarefa = CriarTarefa(Procedimento);

A tarefa começa a executar o procedimento dado como parâmetro e que faz parte do programa previamente carregado em memória

EliminarTarefa (IdTarefa);

EsperaTarefa (IdTarefa);

Bloqueia-se à espera da terminação de outra tarefa ou da tarefa referenciada no parâmetro Idtarefa

18

Interface POSIX de tarefas

19

Especificação Posix de tarefas

- No Unix original não existia um modelo multitarefa, pelo que diversos ambientes de programação acrescentaram a interface de programação baseada em tarefas inicialmente numa abordagem de co-rotinas ou pseudoparalelismo
- O modelo multitarefa da especificação POSIX pretendeu normalizar o respectivo modelo computacional nos sistemas operativos Unix, mas tem vindo a ser adoptada como referência noutros sistemas.

20

Funções Posix

Tarefas	Criar	Sincronizar com a Terminação	Transferir Controlo	Transferir/ Adormecer	Terminar
POSIX	pthread_create	pthread_join	pthread_yield	sleep	pthread_exit

21

Interface POSIX: criar tarefa

```
int pthread_create(pthread_t *thread,
                   attr *attr, void *(*start), void *arg)
```

0 - sucesso
>0 - erro

identificador da tarefa (apontador)

Define atributos da tarefa (apontador para uma estrutura)

Ponteiro para a função a executar

Ponteiro para um parâmetro da função (pode ser um ponteiro para uma estrutura)

A tarefa começa a sua execução chamando a função referenciada pelo ponteiro para função `start` com o argumento `arg`

22

Interface POSIX: terminação de tarefa

`int pthread_exit(void *value_ptr)`

- Tarefa termina
- Retorna ponteiro para resultado da função
- Semelhante a fazer `return()` da função, contudo `pthread_exit()` pode ser chamado em qualquer ponto do código

`int pthread_join(pthread_t thread,`
`void **value_ptr)`

- Tarefa espera até a tarefa indicada ter terminado
- O retorno da função da tarefa terminada é colocado na variável referenciada por `(*value_ptr)`

23

Conclusão

- As tarefas são a forma de criar paralelismo dentro de um processo
- As tarefas partilham o espaço de endereçamento e o contexto do processo
- São contudo atividades independentes com a sua pilha, *instruction pointer*, contexto de hardware
- O modelo de criação e sincronização com a terminação tem muita semelhanças com o dos processos

24