Sistemas Operacionais

Gestão de tarefas - implementação de tarefas

Prof. Carlos Maziero

DInf UFPR, Curitiba PR

Julho de 2020

Conteúdo

Contexto de execução

Contexto

Estado interno da tarefa, que se modifica conforme a execução da tarefa evolui.

Cada tarefa possui um contexto próprio:

- Posição atual da execução (*program counter*, *stack pointer*)
- Valores das variáveis e áreas de memória
- Arquivos abertos, conexões de rede, ...

Informações do contexto

- Registradores do processador:
  - PC (*Program Counter*), indica a posição da execução
  - SP (*Stack Pointer*), topo da pilha de execução Valores de variáveis
  - Demais registradores (acumulador, etc)
  - Flags (nível usuário/núcleo, status, etc).
- Áreas de memória usadas pela tarefa
- Recursos em uso:
  - Arquivos abertos
  - Conexões de rede
  - Semáforos
  - ...

TCB - *Task Control Block*

Descritor de uma tarefa:

- Identificador da tarefa
- Estado da tarefa (nova, pronta, executando, ...)
- Informações de contexto (registradores, etc)
- Recursos usados pela tarefa (arquivos abertos, ...)
- Informações de contabilização (data de início, tempo de processamento, volume de dados lidos/escritos, etc.).
- Outras informações (prioridade, proprietário, etc.).

Geralmente implementado como um `struct` em C

Troca de contexto

Trocar a tarefa em execução por outra.

É essencial em sistemas multitarefa.

Consiste em:

1. Salvar o contexto da tarefa em execução
2. Escolher a próxima tarefa a executar
3. Restaurar o contexto da próxima tarefa

Troca de contexto

Entidades envolvidas na troca de contexto:

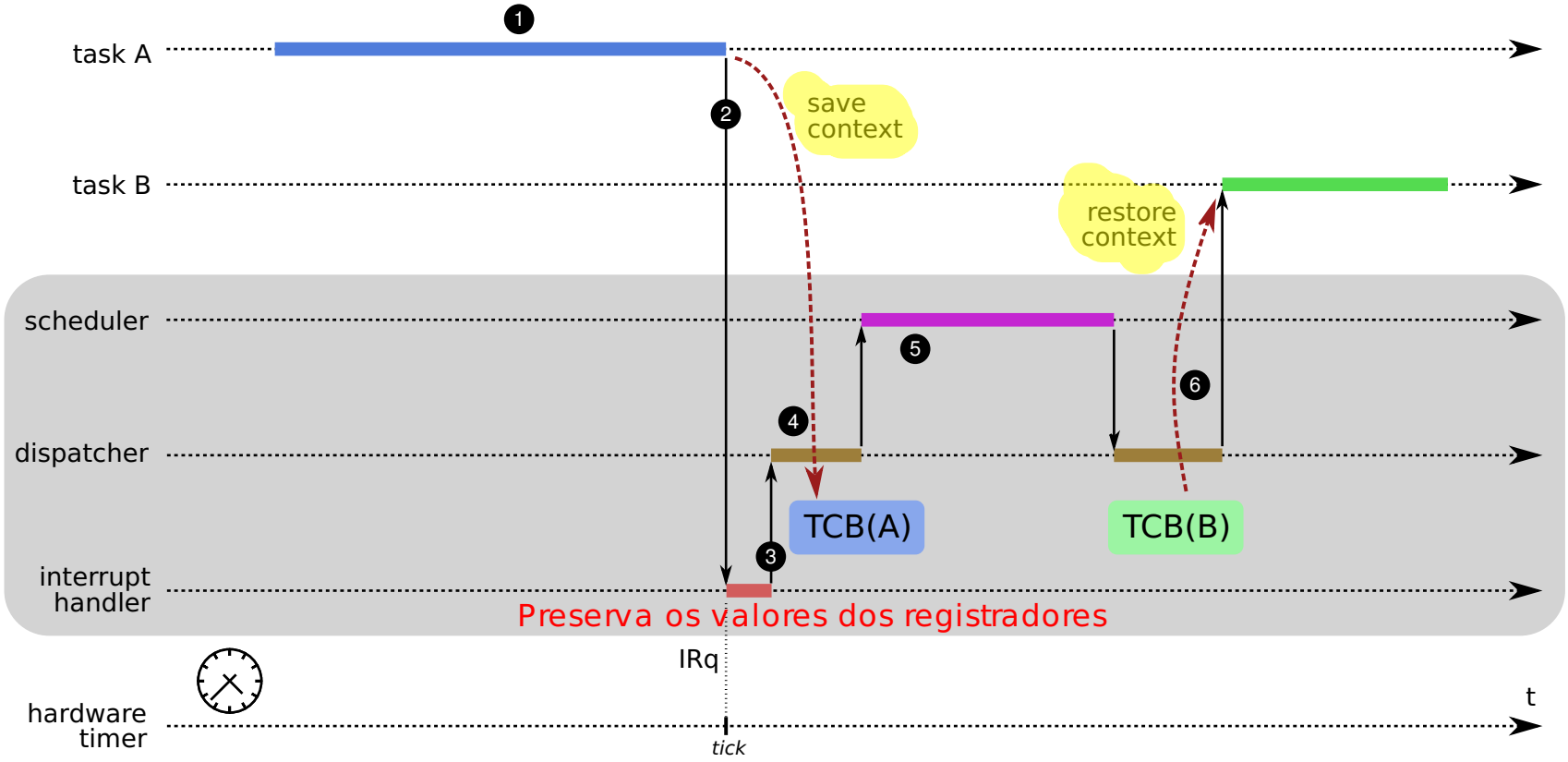
Despachante Dispatcher - quais informações tem que ser salvas, onde serão salvas e resgatá-las de volta a CPU

Implementa a troca de contextos em baixo nível

Escalonador Scheduler

Avalia as tarefas prontas e define a ordem de execução

Trocas de contexto



Implementar tarefas: Processos

Processo

Contêiner de recursos utilizados para executar tarefas

Um processo contém:

- Áreas de memória (código, dados, pilha, ...)
- Descritores de recursos (arquivos, sockets, ...)
- Uma ou mais tarefas em execução

Os processos são isolados entre si pelo hardware:

- Isolamento de áreas de memória (MMU)
- Níveis de operação da CPU (*kernel*/*user*)

Criação de processos

Processos são criados através de chamadas de sistema.

Em Windows:

- `CreateProcess`: cria um novo processo, informando o arquivo contendo o código a executar.

Em UNIX:

- `fork()`: duplica o processo atual. Cria um processo filho identico ao pai
- `execve(file)`: substitui o código executável do processo por outro código.

Hierarquia de processos

```
1  init-+-cron
2       |-dhcpcd
3       |-getty
4       |-getty
5       |-ifplugd
6       |-ifplugd
7       |-lighttpd---php-cgi-+-php-cgi
8       |                    `-php-cgi
9       |-logsave
10      |-logsave
11      |-ntpd
12      |-openvpn
13      |-p910nd
14      |-rsyslogd-+-{rsyslogd}
15      |          `-{rsyslogd}
16      |-sshd---sshd---sshd---pstree
17      |-thd
18      `-udevd-+-udevd
19              `-udevd
```

Implementar tarefas: Threads

Thread

Fluxo de execução operando dentro de um processo ou dentro do núcleo do sistema operacional.

Por default, cada processo contém um(a) *thread*

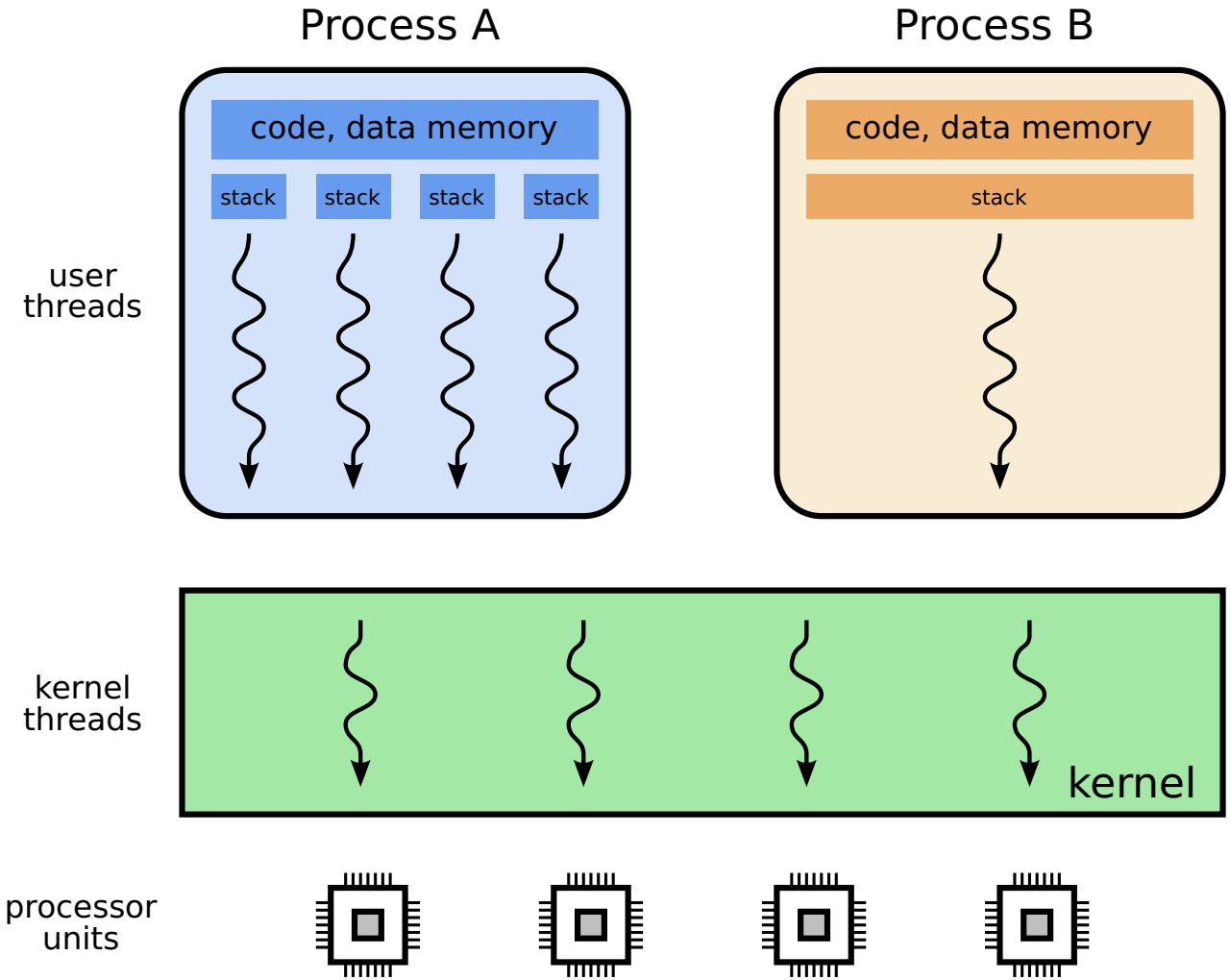
Threads

Tipos de *threads*:

- Threads de usuário: fluxos de execução dentro de um processo, associados à execução da aplicação.
- Threads de núcleo: fluxos de execução dentro do núcleo; representam threads de usuário ou atividades do núcleo.

~~Modelos de implementação: N:1, 1:1 ou N:M~~

Threads

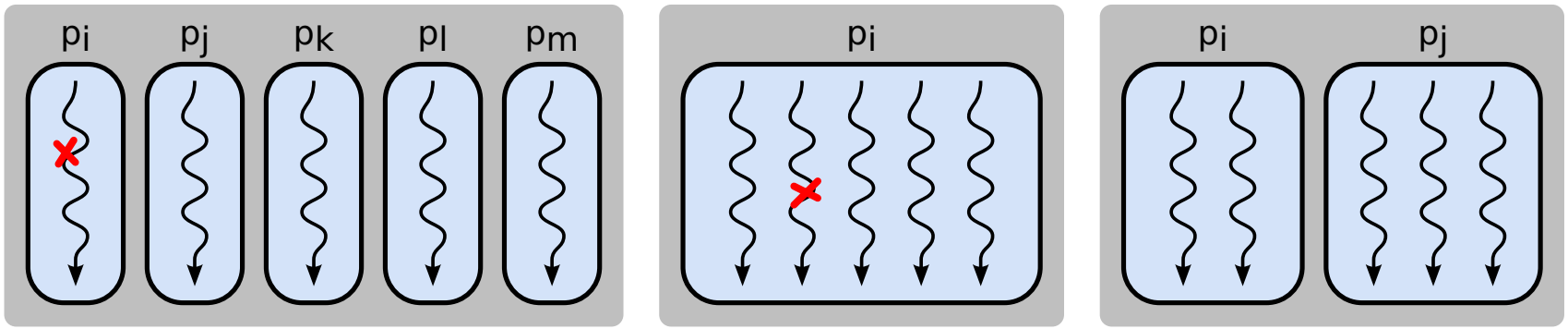

Process A
Process B
code, data memory
stack
stack
stack
stack
code, data memory
stack
user threads
kernel threads
kernel
processor units

Usar *threads* ou processos?

Ao implementar um programa com várias tarefas:

- Colocar cada tarefa em um processo distinto
- Colocar tarefas como threads no mesmo processo
- Híbrido: usar vários processos com várias threads

Comunicação mais fácil



com processos | com threads | híbrido

Vantagem: robustez (erro) e segurança

Desvantagem: ocupa mais memória do que a abordagem com threads

Um erro em uma thread pode derrubar todo o processo

Usar *threads* ou processos?

Característica	**Com processos**	**Com *threads* (1:1)**	**Híbrido**
Custo de criação de tarefas	alto	baixo	médio
Troca de contexto	lenta	rápida	variável
Uso de memória	alto	baixo	médio
Compartilhamento de dados entre tarefas	canais de comunicação e áreas de memoria compartilhada.	variáveis globais e dinâmicas.	ambos.
Robustez	um erro fica contido no processo.	um erro pode afetar todas as *threads*.	um erro pode afetar as *threads* no mesmo processo.
Segurança	cada processo pode executar com usuários e permissões distintas.	todas as *threads* herdam as permissões do processo onde executam.	*threads* com as mesmas permissões podem ser agrupadas em um mesmo processo.
Exemplos	Apache 1.*, PostGres	Apache 2.*, MySQL	Chrome, Firefox, Oracle