CURSO SUPERIOR EM TECNOLOGIA EM REDES DE COMPUTADORES

# SISTEMAS OPERACIONAIS

## Aula 09

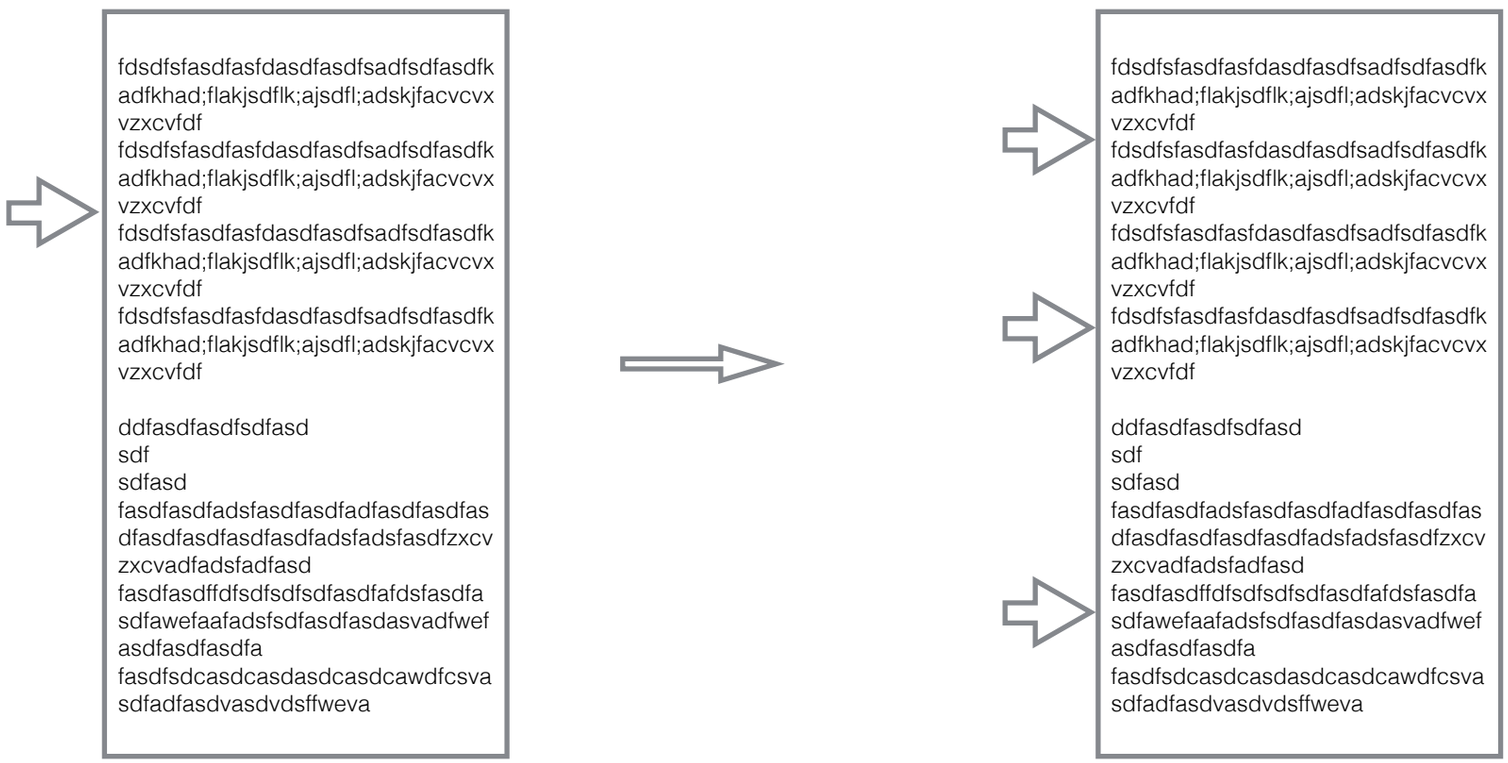
## Gerência de Processos

PROFESSOR ANTÔNIO ROGÉRIO MACHADO RAMOS

Avaliação 2 e recuperação da avaliação 1
na aula 10

# ULT - User Level Threads

**Processo pesado** com uma **única thread** controlada pelo **escalonador** de curta duração do SO (**camada kernel**).




**Três threads** executando no **processo pesado** controladas pelo **escalonador** de curta duração do processo pesado (**camada usuário**).

# ULT - User Level Threads

O **controle das threads** é feito pelo **processo** que está **executando**. Este processo possui um pequeno **escalonador** de curta duração **para as threads**. O SO não precisa ser multithread. Ele, inclusive, executa o processo pesado com uma única thread (camada kernel) e esse processo, com seu escalonador e suas threads executam na camada de cima (usuário).

A **vantagem do ULT** é que o SO **não precisa** estar orientado a trabalhar com **várias threads** por processo pesado (multi thread). O processo do usuário fica mais independente do sistema operacional. No caso de linguagens que possuem uma máquina virtual (Java), essa máquina possui um escalonador de threads (JVM).

A **desvantagem** é que se uma **thread gerar** uma **trap**, o SO vai tratar como se o processo pesado gerasse essa trap, abortando sua execução. Considerando que o processo do usuário é que faz o multithreading, **todas as threads são abortadas**. Isso é recorrente em navegadores como o IE, por exemplo. Quando uma aba apresenta um erro fatal, todo o navegador (e todas as demais abas abertas) é fechado.

# ULT - User Level Threads

**KERNEL**

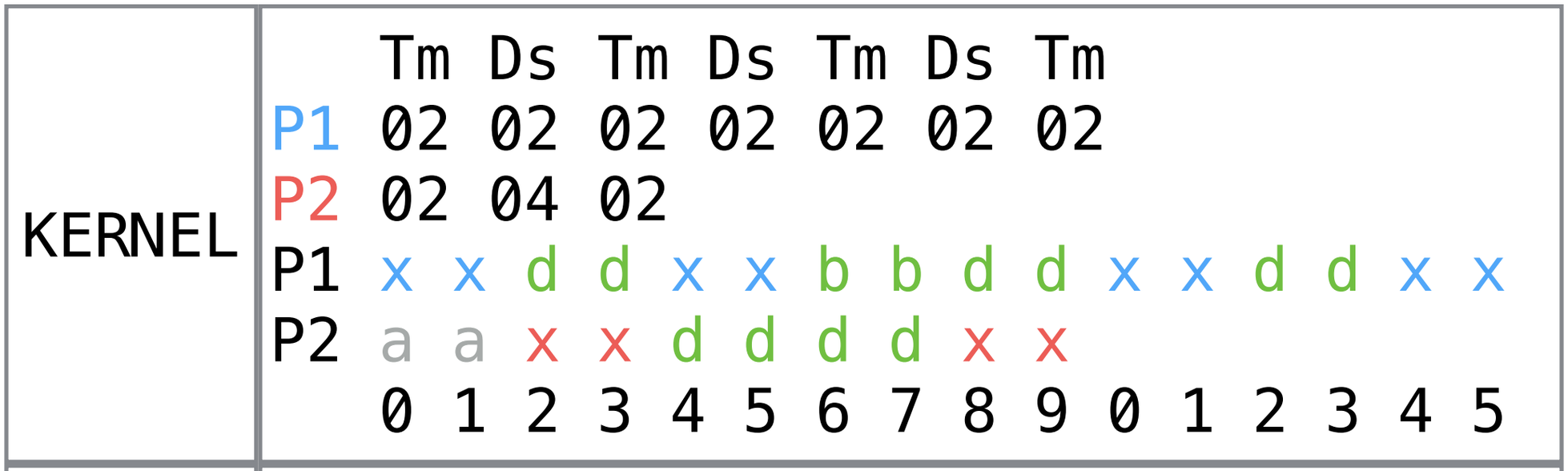
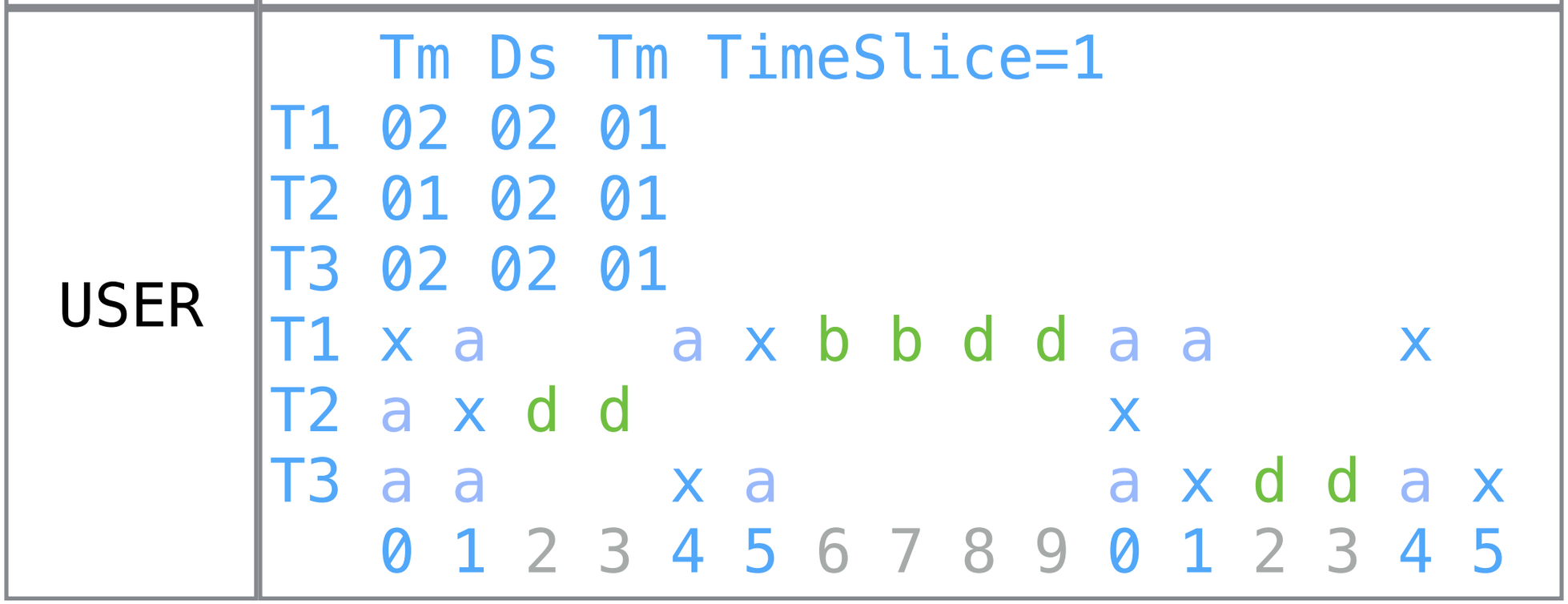
```
   Tm ..........Escalonamento..........
P1 10 x x x a a a x x x a a a x x x a a x
P2 08 a a a x x x a a a x x x a a a x x
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
TimeSlice=3
```

**USER**

```
   Tm ..........Escalonamento..........
T1 03 x x a       a a a       x
T2 03 a a x       x a a       a x
T3 04 a a a         x x       a a x     x
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
TimeSlice=2
```

Os tempos em cinza referem-se aos tempos em que P1 não está executando. Nenhuma thread está executando nestes tempos, pois eles não são percebidos pelo escalonador do processo do usuário.

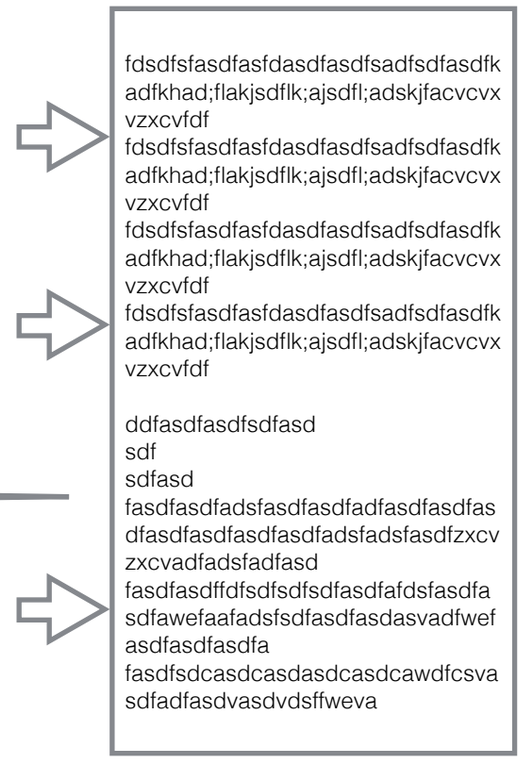
# ULT - User Level Threads

| | |
|---|---|
| KERNEL |  |
| USER |  |

# KLT - Kernel Level Threads

O **SO** tem um **escalonador multithread** nativo e executa os **processos pesados** que tenham **uma ou mais threads**. Os SOs modernos operam desta forma. Para confirmar, entre no gerenciador de tarefas do windows  e veja quantos processadores ele mostra no gráfico. Geralmente é exibido o número de processadores lógicos, que é o dobro (no caso de processadores intel) do número de processadores físicos. Isso ocorre porque cada núcleo HT (hyper threading), que permite a execução de 2 threads ao mesmo tempo.

A **vantagem** é que, se uma thread gerar um **erro fatal**, **apenas ela** será **abortada**.




Três threads executando no SO controladas pelo escalonador de curta duração do SO.

# KLT - Kernel Level Threads

KERNEL

```
   Tm ..........Escalonamento..........
T1 03 x x a a a a a a x
T2 03 a a x x a a a a a x
T3 04 a a a a x x a a a a x x
P2 08 a a a a a a x x a a a a x x x x x x
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
TimeSlice=2

Processo P1: 3 threads (T1, T2 e T3)
Processo P2: 1 thread  (P1)

Quando o processo tem apenas uma linha de
execução (thread), ela tem a mesma
identificação do processo.
```

**Como é que um programa vira um processo?**

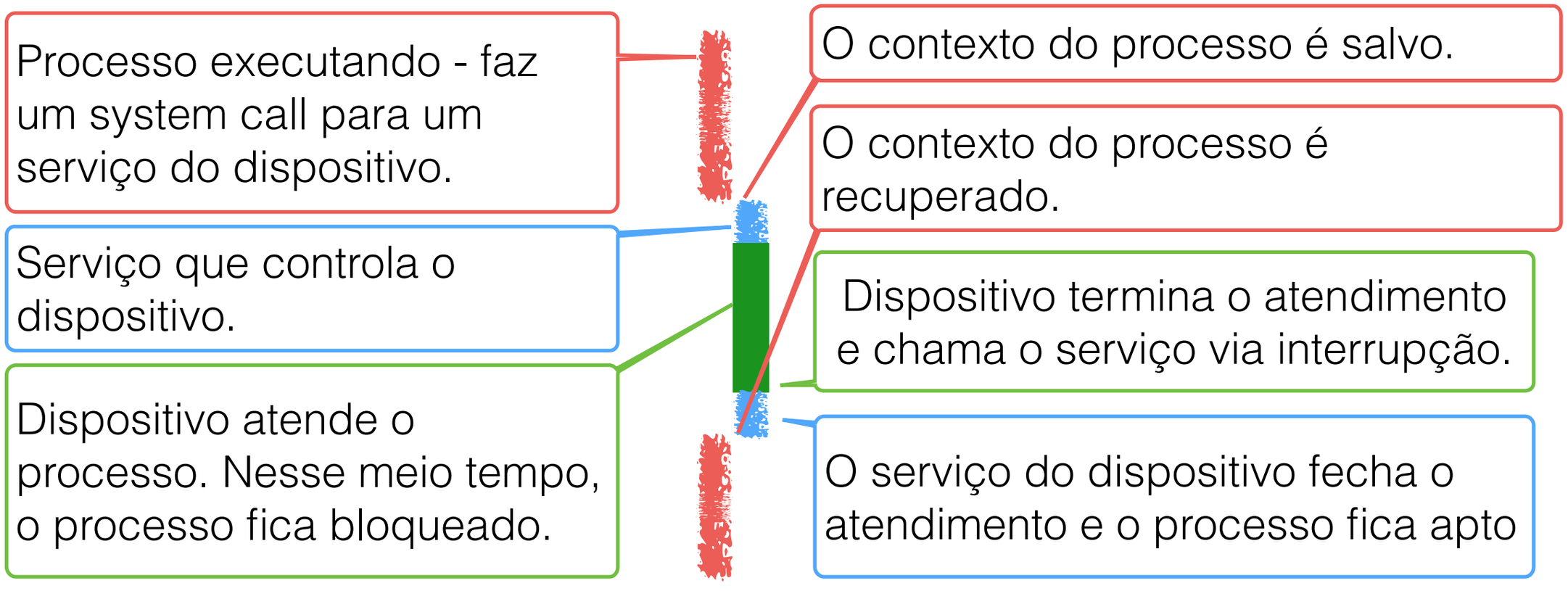
Quando solicitamos ao SO que um programa seja executado, um serviço (**loader**) é disparado. Ele opera junto com o gerenciador de memória para fazer a **alocação** de páginas para o **programa** ser **carregado**.

Também é feita uma alocação em páginas para os dados de processamento.

**Cada thread** possui os **seus dados** de processamento alocados na **memória**, mas existe **apenas uma área** de código para cada **processo pesado**.

Uma área de código.

2 áreas de dados (uma para cada thread



**O que é o contexto de um processo?**
Cada thread (processo leve) possui um conjunto de dados sobre o seu status. A seguir são apresentados alguns destes dados:

- Valor de cada registrador
- Valor do PSW (registrador de status)
- Valor dos ponteiros das pilhas
- Valor do PC (contador de programa)
- Número do processo (PID)

Essas informações fazem parte de uma linha em uma tabela chamada **PCB - Process Control Block**. Nessa tabela ficam os **dados sobre todos os processos** que estão **executando** no SO.

PCB
dados do processo 1
dados do processo 2
dados do processo 3
dados do processo 4
...
dados do processo n

O serviço que acessa o PCB é o **chaveador de contexto**. Ele é responsável por **salvar o contexto** do processo quando ele **perde a CPU** e também de **recuperar esse contexto** quando o processo vai **executar**. Desta forma a CPU retoma o processo de onde havia parado.

**Serviços envolvidos no contexto do processo**.

- Chaveador de contexto - atua em 2 frentes:
  - Quando o processo perde a CPU por fim do time-slice ou por systemcall, salvando o contexto do processo no PCB.
  - Quando o processo retoma a CPU para executar, recuperando o contexto do processo e atualizando os dados na CPU e na memória.
- Driver do dispositivo - atua em 2 frentes:
  - Quando acessa o dispositivo chamado via systemcall.
  - Alertando sobre o fim do processamento do dispositivo - ativado por interrupção de hardware.

Esses serviços são fundamentais para o funcionamento do escalonador de curta duração e são muito pequenos, não sendo considerados nas tabelas de escalonamento desenvolvidas nos exercícios em aula.

Existem mais 2 tipos de escalonadores:
**Média duração (EMD)**: faz swaping nos processos.
**Longa duração (ELD)**: verifica qual processo é carregado e qual é terminado.

**Escalonador de curta duração (ECD)**:
É o conjunto de serviços que faz o escalonamento dos processos na execução, determinando as etapas do ciclo de vida de cada um deles.

ELD: processo inteiro

EMD: página/segmento

ECD: processo CPU

O que acontece em P1 também acontece em P2 e P3. Cada um dos processos possui somente uma thread.

No ULT, o contexto de cada thread também é salvo pelo escalonador do processo.

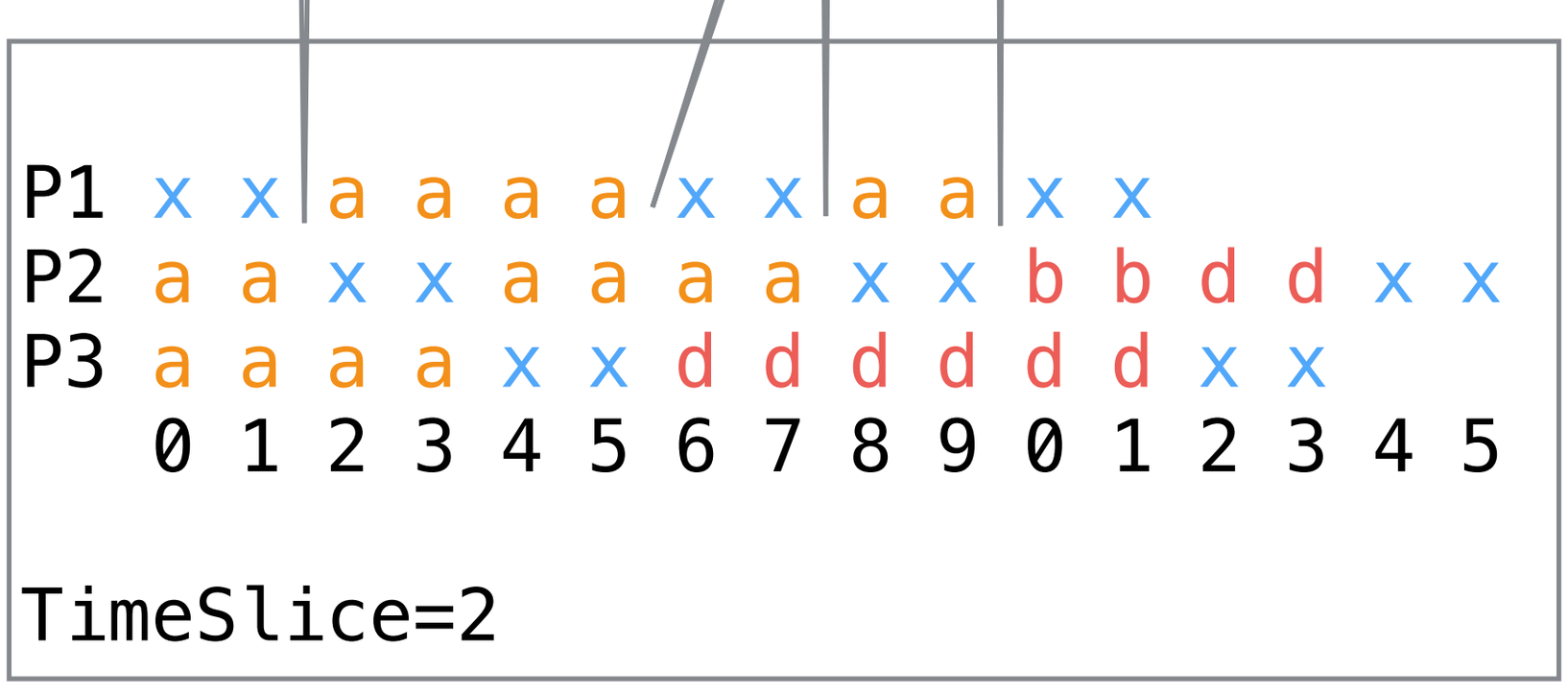
No KLT, o contexto de cada thread também é salvo pelo escalonador do kernel.

Recuperando o contexto de P1

Salvando o contexto de P1

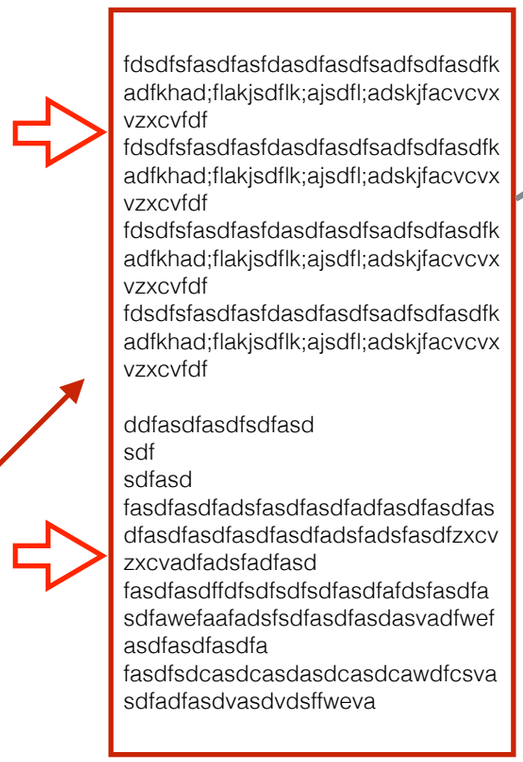
Salvando o contexto de P1

Recuperando o contexto de P1



Um Kernel com suporte KLT pode operar com processos ULT. Nesta modalidade, cada thread de um processo em KLT pode representar um processo com várias threads em ULT.
Confuso? Veja o desenho abaixo.

KLT - executando 2 threads no processo pesado.


ULT - executando 2 threads derivadas da thread no KLT.

ULT - executando 3 threads derivadas da thread no KLT.