

# Simple MFQS

PCS3732 – Laboratório de Processadores  
Prof: Bruno Abrantes Basseto

## Integrantes:

Dênio Araujo de Almeida

Francisco Cavaleiro Mariani

Lucas Von Ancken Garcia

NUSP: 10309013

NUSP: 11803701

NUSP: 11257592

# Agenda

1. Proposta do Projeto
2. Funcionamento do Algoritmo MFQS
3. Implementação dos Principais Componentes
4. Demonstração prática

# Proposta do Projeto



# Proposta do Projeto

Implementar, na placa Evaluator, um escalonador de processos com prioridades que atenda aos seguintes requisitos funcionais

01

Controlar o tempo de execução dos processos com **interrupções de timer**

02

Evitar que processos de mais baixa prioridade sofram **starvation**

03

Permitir que processos concedam voluntariamente tempo de execução (**yield**)

04

Permitir que processos sejam finalizados e retirados de execução (**halt**)

# Funcionamento do Algoritmo MFQS



# Escalonador MFQS

O MFQS (Multi-Level Feedback Queue Scheduler) prevê a existência de múltiplas filas de processos com diferentes prioridades

01

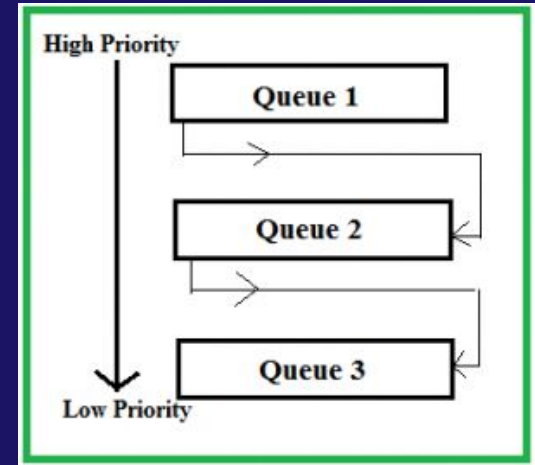
Processos de menor prioridade só poderão executar após as filas de maior prioridade terem sido esvaziadas

02

Feedback: a prioridade dos processos muda dinamicamente de acordo com o comportamento deles ao longo do tempo

03

Aging: Se um processo estiver aguardando muito tempo para ser executado (starvation), a prioridade deles pode ser aumentada



# Escalonador MFQS

Com uma estrutura multinível e de feedback, MFQS busca alcançar dois principais objetivos:

01

Diminuir o tempo médio de *turnaround*: procura-se priorizar a execução de processos mais curtos e penalizar processos mais longos

02

Tornar o SO mais responsivo: manter a prioridade de processos que abandonam a execução em CPU para operações de I/O (processos iterativos)

# Escalonador MFQS

**Algumas regras são seguidas pelo MFQS para esses objetivos...**

**01**

Em cada fila, define-se um tempo máximo para a execução contínua de um processo em termos de número de *ticks*: `quanta_limit`

**02**

Atualiza-se continuamente a quantidade de *ticks* restantes (`exec_slots`) do processo em execução

**03**

Se um processo consome muito tempo de CPU ininterruptamente, ultrapassando “`quanta_limit`”, a sua prioridade é diminuída → ajuda a evitar starvation!



# Escalonador MFQS

## Mais algumas regras...

04

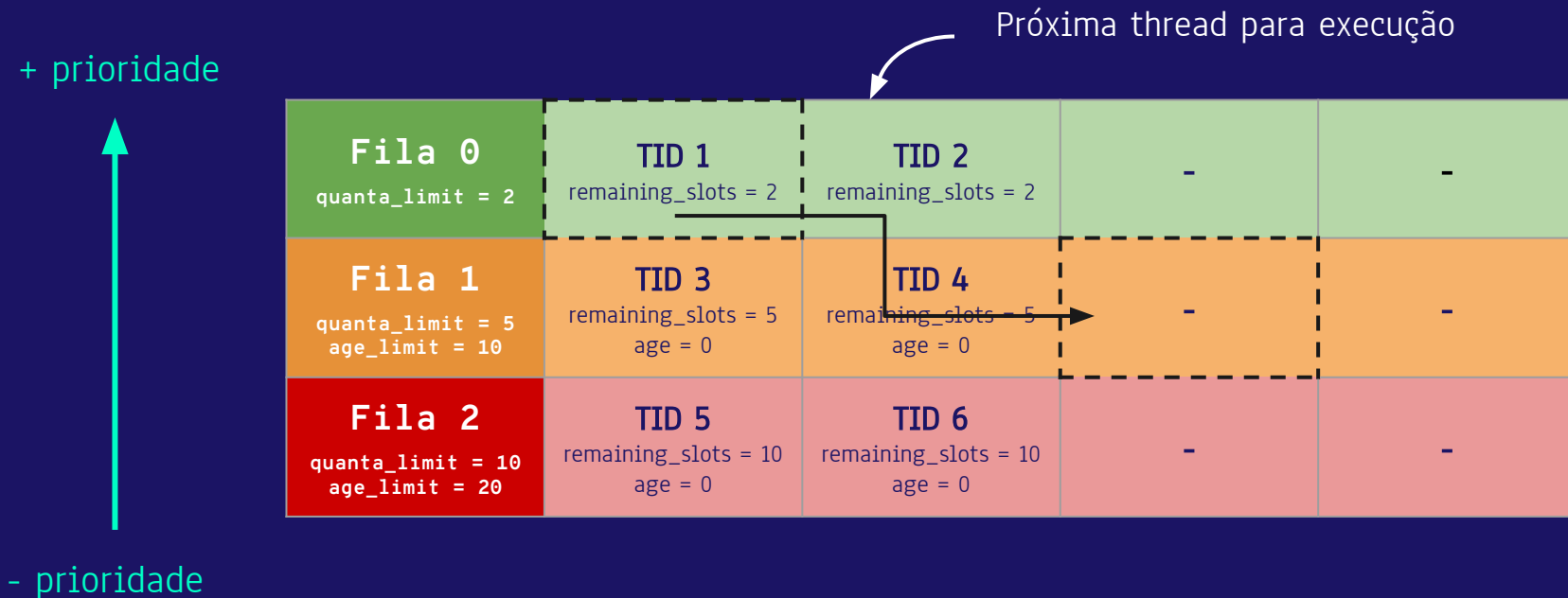
Toda vez que processo deixa a execução antes do esgotamento de `exec_slots`, devido a uma operação I/O por exemplo, ele é retirado da frente da fila, mas sua prioridade é preservada

05

O tempo de espera dos processos (`age`) é atualizado a cada tick. A prioridade dele é aumentada se este tempo supera um limite pré-definido para a fila (`age_limit`)

# Funcionamento do Algoritmo MFQS

Exemplo de Downgrade: Tempo de execução da thread 1 ultrapassou limite de 2 slots de tempo

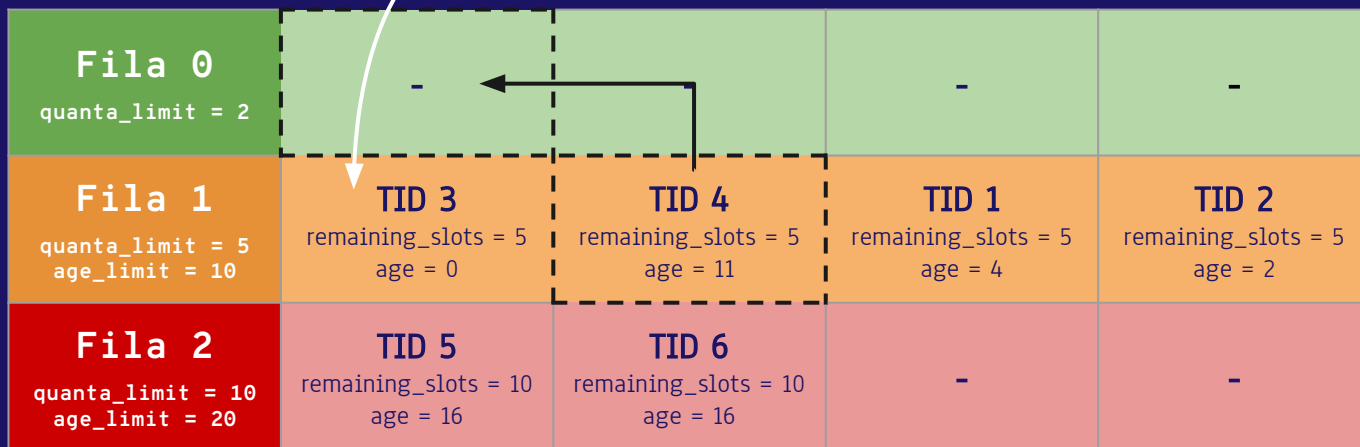


# Funcionamento do Algoritmo MFQS

Exemplo de Upgrade: Tempo de espera da thread 4 ultrapassou limite de 10 slots de tempo (aging)

+ prioridade

Thread 3 em execução, mas será substituída pela thread 4



- prioridade

# Funcionamento do Algoritmo MFQS

Exemplo de Yield ou I/O: Caso a thread 3 seja retirada de execução da CPU antes do esgotamento do slot de tempo, ela permanece na mesma fila

+ prioridade



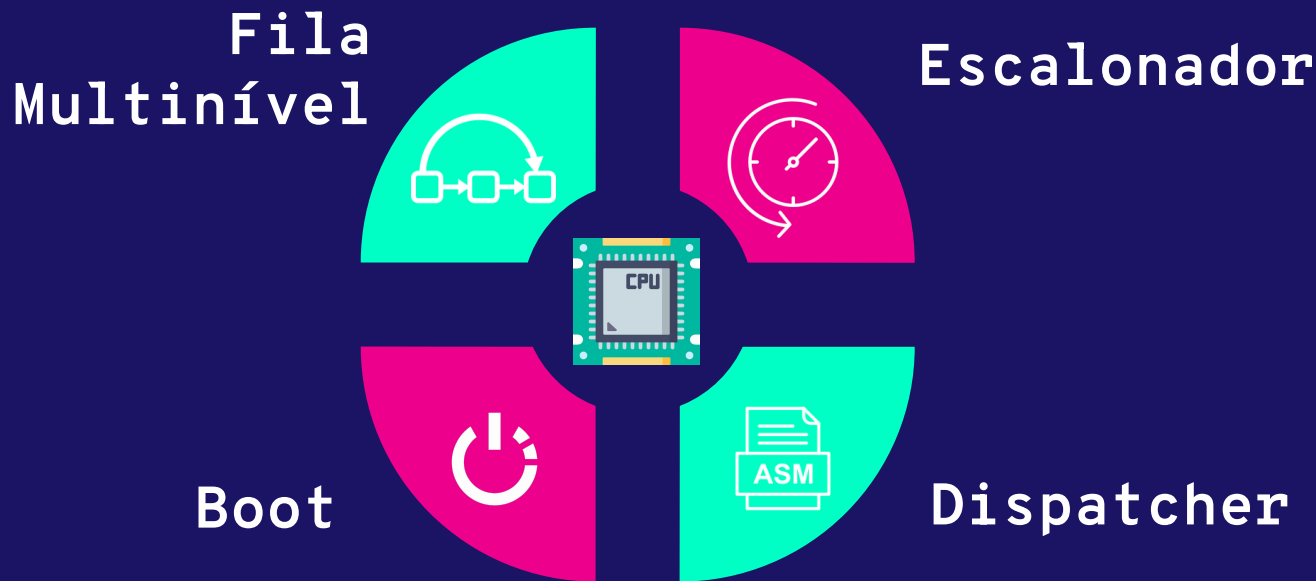
<b>Fila 0</b> quanta_limit = 2				
<b>Fila 1</b> quanta_limit = 5 age_limit = 10	<b>TID 3</b> remaining_slots = 5 age = 0	<b>TID 4</b> remaining_slots = 5 age = 4	<b>TID 2</b> remaining_slots = 5 age = 4	-
<b>Fila 2</b> quanta_limit = 10 age_limit = 20	<b>TID 5</b> remaining_slots = 10 age = 16	<b>TID 6</b> remaining_slots = 10 age = 16	-	-

- prioridade

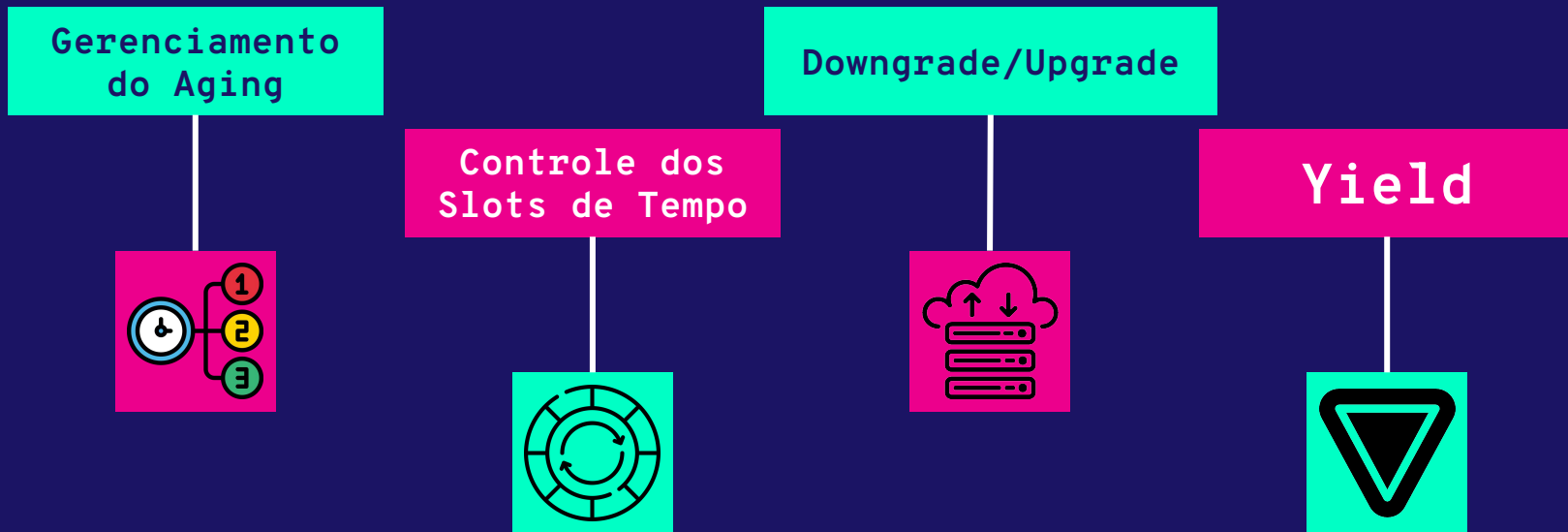
# Detalhes Relevantes da Implementação



# Principais Componentes



# Implementação do Escalonador



# Task Control Block (TCB)

Além do contexto, também são armazenados na TCB os parâmetros específicos de cada thread utilizados pelo escalonador

```
typedef struct {  
    uint32_t regs[17];    // contexto (17 registradores)  
    uint32_t tid;         // identificador da thread  
    uint32_t priority;     // prioridade atual da thread  
    uint32_t exc_slots;    // número de execuções restantes  
    uint32_t age;         // tempo de espera da thread na fila (aging)  
    uint32_t cpu_time;     // quantidade total de execuções de slots de tempo da thread  
} tcb_t;
```



# Implementação da Fila Multinível

Em nossa implementação do MFQS, utilizam-se 3 filas de prioridade construídas com listas duplamente ligadas. Cada nó guarda uma referência para uma TCB.

```
#define NUM_OF_QUEUES 3

typedef struct {
    queue_t* queues[NUM_OF_QUEUES];
    tcb_t* next_thread;
} multiqueue_t;
```

```
// Estrutura da tabela de TCBs
typedef struct {
    node_t* head;
    uint32_t quanta_limit;
    uint32_t age_limit;
} queue_t;
```

```
typedef struct node_t {
    tcb_t* tcb;
    struct node_t* next_node;
    struct node_t* previous_node;
} node_t;
```

# Implementação do Boot

Assim que a placa Evaluator é ligada, executa-se uma função de *boot* responsável por criar as estruturas de fila e inicializar as threads a serem executadas

01

As estruturas TCB e da fila multinível são criadas dinamicamente com uso da função *malloc*

02

Pode ser criadas tanto threads que terminam a sua execução quanto threads que executam indefinidamente.

```
void boot() {  
  
    os_tcb = create_tcb(0, 0, 2, 0, (uint32_t)os_thread);  
  
    tcb_t* user_tcb1 = create_tcb(1, 0, 2, 0, (uint32_t)user_thread);  
    tcb_t* user_tcb2 = create_tcb(2, 0, 2, 0, (uint32_t)user_thread);  
    tcb_t* user_tcb3 = create_tcb(3, 1, 4, 0, (uint32_t)user_thread);  
    tcb_t* user_tcb4 = create_tcb(4, 1, 4, 0, (uint32_t)user_thread);  
    tcb_t* user_tcb5 = create_tcb(5, 2, 6, 0, (uint32_t)user_thread);  
    tcb_t* user_tcb6 = create_tcb(6, 2, 6, 0, (uint32_t)user_thread);  
  
    queue_t *queue0 = (queue_t*)malloc(sizeof(queue_t));  
    queue0->quanta_limit = 2;  
    queue0->age_limit = 5;  
    queue0->head = NULL;  
  
    queue_t *queue1 = (queue_t*)malloc(sizeof(queue_t));  
    queue1->quanta_limit = 4;  
    queue1->age_limit = 10;  
    queue1->head = NULL;  
  
    queue_t *queue2 = (queue_t*)malloc(sizeof(queue_t));  
    queue2->quanta_limit = 6;  
    queue2->age_limit = 21;  
    queue2->head = NULL;  
}
```

# Funções Yield e Halt

## Função Yield:

Permite visualizar os efeitos de uma concessão de tempo de execução antes do esgotamento do slot de tempo no MFQS, optou-se por utilizar a interrupção de botão da Evaluator para essa finalidade.

## Função Halt:

De modo a permitir que uma thread em execução seja finalizada, criou-se uma chamada de sistema para esse propósito: `halt()`. Ela promove a retirada do nó correspondente a thread em execução da estrutura da fila multinível

# Implementação do Halt

De modo a permitir que uma thread em execução seja finalizada, criou-se uma chamada de sistema para esse propósito: *halt()*

O tratamento dessa chamada de sistema envolve:

01

Remover nó correspondente a thread em execução da estrutura da fila multinível

02

Realizar um novo escalonamento e troca de contexto

```
void __attribute__((naked)) halt(void) {  
    asm volatile("mov r0, #5 \n\t"  
                 "swi #0 \n\t");  
}
```

```
/* Finaliza a thread atual e realiza troca de contexto */  
handle_halt:  
    bl finish_current_thread // atualização de estado da thread  
    bl mfqs_scheduler // current_tcb é atualizada  
    b context_change
```

# Implementação do Yield

De modo a permitir a visualização dos efeitos de uma concessão de tempo de execução antes do esgotamento do slot de tempo no MFQS, optou-se por utilizar a interrupção de botão da Evaluator para essa finalidade.

01

Tratamento específico para interrupção de botão

02

Nesse tratamento, a thread é movida para o final da mesma fila caso ainda reste slots de execução (prioridade mantida nesse caso)

# Outros Aspectos Importantes

## Condição de Fila Multinível vazia:

O sistema operacional executa indeterminadamente uma thread padrão até que uma nova thread surja na fila multinível

## Chamadas de Sistema Auxiliares:

De modo a permitir uma thread acesse dados de sua TCB, tornou-se necessário implementar chamadas de sistema para que o sistema operacional disponibilize essas informações.

```
int __attribute__((naked)) getpid(void) {  
    asm volatile("push {lr} \n\t"  
                 "mov r0, #2 \n\t"  
                 "swi #0 \n\t"  
                 "pop {pc}");  
}
```

```
int __attribute__((naked)) get_cpu_time(void) {  
    asm volatile("push {lr} \n\t"  
                 "mov r0, #4 \n\t"  
                 "swi #0 \n\t"  
                 "pop {pc}");  
}
```

```
int __attribute__((naked)) get_priority(void) {  
    asm volatile("push {lr} \n\t"  
                 "mov r0, #3 \n\t"  
                 "swi #0 \n\t"  
                 "pop {pc}");  
}
```

# Demonstração Prática



# Demonstração

- **Placa Evaluator-7T**
  - Display de 7 segmentos exhibe **TID da thread**
  - LEDs exibem fila onde a thread está:
    - **verde** é a fila mais prioritária
    - **amarelo** é a fila intermediária
    - **vermelho** é a fila menos prioritária



# Demonstração

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread X remaining_slots = a age = b cpu_time = c	-	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	-	-	-	-	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 0

<b>Fila 0</b> quanta_limit = 2	Thread 1 remaining_slots = 2 cpu_time = 0	Thread 2 remaining_slots = 2 cpu_time = 0	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 3 remaining_slots = 4 age = 0 cpu_time = 0	Thread 4 remaining_slots = 4 age = 0 cpu_time = 0	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 0 cpu_time = 0	Thread 6 remaining_slots = 6 age = 0 cpu_time = 0	-	-	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 1

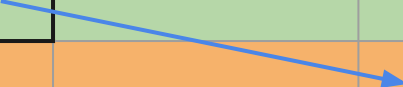
<b>Fila 0</b> quanta_limit = 2	Thread 1 remaining_slots = 1 cpu_time = 1	Thread 2 remaining_slots = 2 cpu_time = 0	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 3 remaining_slots = 4 age = 1 cpu_time = 0	Thread 4 remaining_slots = 4 age = 1 cpu_time = 0	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 1 cpu_time = 0	Thread 6 remaining_slots = 6 age = 1 cpu_time = 0	-	-	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 2

<b>Fila 0</b> quanta_limit = 2	Thread 1 remaining_slots = 0 cpu_time = 2	Thread 2 remaining_slots = 2 cpu_time = 0	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 3 remaining_slots = 4 age = 2 cpu_time = 0	Thread 4 remaining_slots = 4 age = 2 cpu_time = 0	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 2 cpu_time = 0	Thread 6 remaining_slots = 6 age = 2 cpu_time = 0	-	-	-	-



\*thread termina quando cpu\_time == 9

# Demonstração

t = 2

<b>Fila 0</b> quanta_limit = 2	<b>Thread 2</b> remaining_slots = 2 cpu_time = 0	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 3</b> remaining_slots = 4 age = 2 cpu_time = 0	<b>Thread 4</b> remaining_slots = 4 age = 2 cpu_time = 0	<b>Thread 1</b> remaining_slots = 4 age = 0 cpu_time = 2	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 age = 2 cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 age = 2 cpu_time = 0	-	-	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 3


<b>Fila 0</b> quanta_limit = 2	Thread 2 remaining_slots = 1 cpu_time = 1	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 3 remaining_slots = 4 age = 3 cpu_time = 0	Thread 4 remaining_slots = 4 age = 3 cpu_time = 0	Thread 1 remaining_slots = 4 age = 1 cpu_time = 2	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 3 cpu_time = 0	Thread 6 remaining_slots = 6 age = 3 cpu_time = 0	-	-	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 4

<b>Fila 0</b> quanta_limit = 2	Thread 2 remaining_slots = 0 cpu_time = 2	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 3 remaining_slots = 4 age = 4 cpu_time = 0	Thread 4 remaining_slots = 4 age = 4 cpu_time = 0	Thread 1 remaining_slots = 4 age = 2 cpu_time = 2	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 4 cpu_time = 0	Thread 6 remaining_slots = 6 age = 4 cpu_time = 0	-	-	-	-



\*thread termina quando cpu\_time == 9

# Demonstração

t = 4

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 3</b> remaining_slots = 4 <b>age = 0</b> cpu_time = 0	<b>Thread 4</b> remaining_slots = 4 <b>age = 4</b> cpu_time = 0	<b>Thread 1</b> remaining_slots = 4 <b>age = 2</b> cpu_time = 2	<b>Thread 2</b> remaining_slots = 4 age = 0 cpu_time = 2	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 <b>age = 4</b> cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 <b>age = 4</b> cpu_time = 0	-	-	-	-

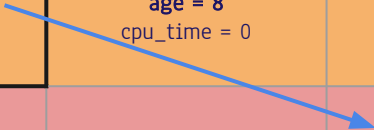
\*thread termina quando cpu\_time == 9



# Demonstração

t = 8

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 3</b> remaining_slots = 0 age = 0 cpu_time = 4	<b>Thread 4</b> remaining_slots = 4 age = 8 cpu_time = 0	<b>Thread 1</b> remaining_slots = 4 age = 6 cpu_time = 2	<b>Thread 2</b> remaining_slots = 4 age = 4 cpu_time = 2	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 age = 8 cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 age = 8 cpu_time = 0	-	-	-	-



\*thread termina quando cpu\_time == 9

# Demonstração

t = 8

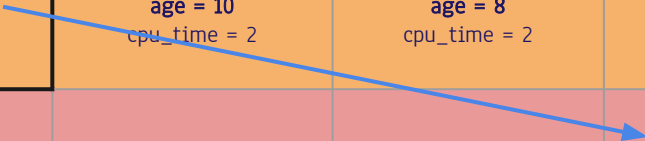
<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 4</b> remaining_slots = 4 <b>age = 0</b> cpu_time = 0	<b>Thread 1</b> remaining_slots = 4 <b>age = 6</b> cpu_time = 2	<b>Thread 2</b> remaining_slots = 4 <b>age = 4</b> cpu_time = 2	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 <b>age = 8</b> cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 <b>age = 8</b> cpu_time = 0	<b>Thread 3</b> <b>remaining_slots = 6</b> age = 0 cpu_time = 4	-	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 12

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 4</b> remaining_slots = 0 age = 0 cpu_time = 4	<b>Thread 1</b> remaining_slots = 4 age = 10 cpu_time = 2	<b>Thread 2</b> remaining_slots = 4 age = 8 cpu_time = 2	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 age = 12 cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 age = 12 cpu_time = 0	<b>Thread 3</b> remaining_slots = 6 age = 4 cpu_time = 4	-	-	-



\*thread termina quando cpu\_time == 9

# Demonstração

t = 12

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 1</b> remaining_slots = 4 <b>age = 0</b> cpu_time = 2	<b>Thread 2</b> remaining_slots = 4 <b>age = 8</b> cpu_time = 2	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 <b>age = 12</b> cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 <b>age = 12</b> cpu_time = 0	<b>Thread 3</b> remaining_slots = 6 <b>age = 4</b> cpu_time = 4	<b>Thread 4</b> <b>remaining_slots = 6</b> age = 0 cpu_time = 4	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 15

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 1</b> remaining_slots = 1 age = 0 cpu_time = 5	<b>Thread 2</b> remaining_slots = 4 age = 11 cpu_time = 2	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 age = 15 cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 age = 15 cpu_time = 0	<b>Thread 3</b> remaining_slots = 6 age = 7 cpu_time = 4	<b>Thread 4</b> remaining_slots = 6 age = 3 cpu_time = 4	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 15

<b>Fila 0</b> quanta_limit = 2	Thread 2 remaining_slots = 2 age = 0 cpu_time = 2	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 1 remaining_slots = 1 age = 0 cpu_time = 5	-	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 15 cpu_time = 0	Thread 6 remaining_slots = 6 age = 15 cpu_time = 0	Thread 3 remaining_slots = 6 age = 7 cpu_time = 4	Thread 4 remaining_slots = 6 age = 3 cpu_time = 4	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 17

<b>Fila 0</b> quanta_limit = 2	Thread 2 remaining_slots = 0 age = 0 cpu_time = 4	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 1 remaining_slots = 1 age = 2 cpu_time = 5	-	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 17 cpu_time = 0	Thread 6 remaining_slots = 6 age = 17 cpu_time = 0	Thread 3 remaining_slots = 6 age = 9 cpu_time = 4	Thread 4 remaining_slots = 6 age = 5 cpu_time = 4	-	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 17

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 1</b> remaining_slots = 1 <b>age = 0</b> cpu_time = 5	<b>Thread 2</b> remaining_slots = 4 age = 0 cpu_time = 4	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 <b>age = 17</b> cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 <b>age = 17</b> cpu_time = 0	<b>Thread 3</b> remaining_slots = 6 <b>age = 9</b> cpu_time = 4	<b>Thread 4</b> remaining_slots = 6 <b>age = 5</b> cpu_time = 4	-	-

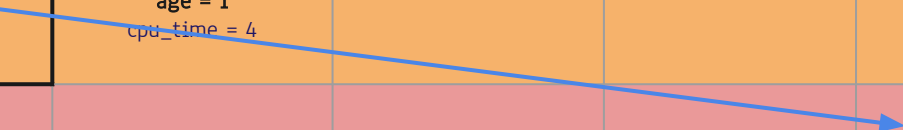
\*thread termina quando cpu\_time == 9



# Demonstração

t = 18

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	Thread 1 remaining_slots = 0 age = 0 cpu_time = 6	Thread 2 remaining_slots = 4 age = 1 cpu_time = 4	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	Thread 5 remaining_slots = 6 age = 18 cpu_time = 0	Thread 6 remaining_slots = 6 age = 18 cpu_time = 0	Thread 3 remaining_slots = 6 age = 10 cpu_time = 4	Thread 4 remaining_slots = 6 age = 6 cpu_time = 4	-	-



\*thread termina quando cpu\_time == 9

# Demonstração

t = 18

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 2</b> remaining_slots = 4 <b>age = 0</b> cpu_time = 4	-	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 <b>age = 18</b> cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 <b>age = 18</b> cpu_time = 0	<b>Thread 3</b> remaining_slots = 6 <b>age = 10</b> cpu_time = 4	<b>Thread 4</b> remaining_slots = 6 <b>age = 6</b> cpu_time = 4	<b>Thread 1</b> remaining_slots = 6 age = 0 cpu_time = 6	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 22

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 2</b> remaining_slots = 0 age = 0 cpu_time = 8	-	-	-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 5</b> remaining_slots = 6 age = 22 cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 age = 22 cpu_time = 0	<b>Thread 3</b> remaining_slots = 6 age = 14 cpu_time = 4	<b>Thread 4</b> remaining_slots = 6 age = 10 cpu_time = 4	<b>Thread 1</b> remaining_slots = 6 age = 4 cpu_time = 6	-

\*thread termina quando cpu\_time == 9

# Demonstração

t = 22

<b>Fila 0</b> quanta_limit = 2	-	-	-	-	-	-
<b>Fila 1</b> quanta_limit = 4 aging_limit = 10	<b>Thread 5</b> remaining_slots = 6 age = 0 cpu_time = 0	<b>Thread 6</b> remaining_slots = 6 age = 0 cpu_time = 0		-	-	-
<b>Fila 2</b> quanta_limit = 6 age_limit = 21	<b>Thread 3</b> remaining_slots = 6 age = 14 cpu_time = 4	<b>Thread 4</b> remaining_slots = 6 age = 10 cpu_time = 4	<b>Thread 1</b> remaining_slots = 6 age = 4 cpu_time = 6	<b>Thread 2</b> remaining_slots = 0 age = 0 cpu_time = 8		-

\*thread termina quando cpu\_time == 9