

CURSO SUPERIOR EM TECNOLOGIA EM REDES DE
COMPUTADORES

SISTEMAS OPERACIONAIS

Aula 08

Gerência de Processos

PROFESSOR ANTÔNIO ROGÉRIO MACHADO RAMOS

Avaliação 2 e recuperação da avaliação 1
na aula 10

O que é um programa?

É um conjunto finito de instruções dispostas em uma sequência correta para atender uma demanda ou resolver um problema.

O que é um processo?

É o programa em execução, podendo, ao contrário deste, ter uma execução infinita.

O que é um processo pesado?

É o processo ocupando memória para ele e seus dados. Cada processo pesado possui pelo menos uma linha de execução.


O que é um processo leve?

É a linha de execução que percorre o processo pesado para executar suas instruções. Cada processo leve ocupa a memória apenas para os seus dados, uma vez que percorre um processo pesado já existente. Não existe processo leve sem pelo menos um processo pesado. O processo leve ou linha de execução também é conhecido como **thread**.

Como a thread executava nos primeiros sistemas?

Cada processo pesado possuía apenas uma linha de execução e ela nem era conhecida como thread. É como se um texto impresso em uma folha fosse lido por apenas uma pessoa, representada aqui pelo indicador vermelho. Apenas um lê a folha impressa de cada vez.


```
asdasd
a
sd
asd
ad
a
dadasdasdasda
sda
sd
ds
asd
a
sd
asd
asda
sdasdasdasdasdasdasdasdasdasd
asd
asdasd
a
sd
asda
sda
sd
a
sd
as
dasdad
a
da
sd
a
sd
as
dasdasdasdasdasdasdasdasdasda
sda
```



Como a thread executa hoje (multi thread) ?

Cada thread executa o processo pesado percorrendo o trecho de acordo com processamento realizado. Por analogia vamos imaginar um texto impresso em uma folha (processo pesado) sendo lido por duas pessoas diferentes (threads vermelha e azul), onde cada pessoa lê um trecho diferente dele. A folha pode ser lida por várias pessoas ao mesmo tempo.

```
asdasd
a
sd
asd
ad
a
dadasdasdasda
sda
sd
ds
asd
a
sd
asd
asda
sdasdasdasdasdasdasdasdasdasd
asd
asdasd
a
sd
asda
sda
sd
a
sd
as
dasdad
a
da
sd
a
sd
as
dasdasdasdasdasdasdasdasdasda
sda
```



Qual o ciclo de vida de um processo?

Vamos iniciar pelo ciclo de vida mais simples, elencando as etapas na sequencia em que elas ocorrem.

1. criar
2. executar
3. terminar

O programa, que fica no meio persistente (HD, SSD, etc), é carregado na memória com seus dados, ocupando segmentos de memória. Quem faz isso é um serviço chamado loader e, para cada processo existe pelo menos uma thread que vai executá-lo.

O programa é então executado e já é chamado de processo. A thread percorre o processo executando cada instrução.

Quando a thread chega ao fim avisa ao SO que ela terminou. O SO remove os dados da memória ocupados por ela. Quando a última thread do processo termina, a área de código é então removida da memória.

No contexto deste ciclo mais simples podemos executar vários processos em batch, considerando que um roda depois do outro. Vejamos as seguintes situações.

processo 1 (p1) roda em 5 horas.

processo 2 (p2) roda em 3 horas.

processo 3 (p3) roda em 1 hora.

Se os processos forem colocados em uma fila, **p1** será o primeiro a ser atendido (**FIFO - First In First Out**) e **p3** será o último.

Este método de organizar os arquivos não é muito eficiente porque o usuário de **p1** vai ter que esperar muito mais do que a duração de seu processo para ter o resultado dele. Esse problema é denominado **turnaround** que é o tempo que a gente espera pelo resultado desde a hora que o processo foi disparado até o seu término. E no tempo médio de espera todo mundo perde.

horas:	123456789	turnaround:
p1 -	xxxxx	5 horas
p2 -xxx	8 horas
p3 -x	9 horas

5+8+9=22 média: 22/3=7.3

Se soubéssemos de antemão qual processo demora mais, poderíamos utilizar o método **SJF - Shortest Job First**. Infelizmente isso é muito difícil de precisar, embora existam algumas métricas para estimar o tamanho tais como verificar o tamanho do programa (nem sempre dá certo), acesso à dispositivos, número de loops, etc. O que se faz também é dar uma primeira passada (execução) no processo, cronometrar o tempo, e utilizar o resultado para definir o tamanho. O problema é que, dependendo dos dados que o processo manipula, esses tempos podem variar muito...

Exemplo do SJF para 3 processos, onde a ordem de execução é do menor para o maior. Note que o tempo médio de espera fica bem menor.

processo 1 (p1) roda em 5 horas.

processo 2 (p2) roda em 3 horas.

processo 3 (p3) roda em 1 hora.

horas:	123456789	turnaround:
p3 -	x	1 hora
p2 -	.xxx	4 horas
p1 -xxxxx	9 horas

1+4+9=14 média: 14/3=4.7

Também pode-se definir **prioridades**. Os processos são colocados em uma fila e a ordem de atendimento será pela prioridade, ou ordem de importância, que cada processo tem. Essa prioridade era definida pelo operador com base em fatores técnicos - o tempo do processo já é conhecido pela experiência - ou fatores externos - surgiu uma urgência para determinado processo e ele deve ser executado antes dos outros. Pode-se dizer que a prioridade troca logicamente os processos de lugar em uma fila física.

processo 1 (p1) roda em 5 horas - prioridade baixa.

processo 2 (p2) roda em 3 horas - prioridade alta.

processo 3 (p3) roda em 1 hora - prioridade média.

Os sistemas operacionais são divididos em duas categorias - os que definem prioridade numérica em que quanto maior o número, maior a prioridade, e os que definem gentileza em que quanto maior o número, menor a prioridade.

horas:	123456789	turnaround:	
p2 -	xxx	3 horas	
p3 -	...x	4 horas	3+4+9=16 média:16/3=5.3
p1 -xxxxx	9 horas	

Um ciclo de vida mais completo - e complicado.

Este ciclo envolve chamadas de sistema (system call) e interrupção.

1. criado

2. executando

2.1. system call (chamada para um dispositivo)

2.2. bloqueado - o processo vai para fila de processos bloqueados

2.3. interrupção (o dispositivo entrega por DMA, a requisição feita)

2.4. apto - o processo sai da fila de bloqueados e vai para fila de aptos

2.5. executando - ao chegar a sua vez, o processo volta a executar

3. terminando

O processo pode ainda gerar uma **trap** no passo 2.1 e ir direto para o passo 3. A trap é uma interrupção de emergência causada por alguma violação feita pelo processo.

Enquanto o processo está bloqueado ele não está executando. Nesse meio tempo, a CPU está executando um outro processo para não ficar ociosa. Quando o outro processo ficar bloqueado, o processo que estava esperando para executar (apto) volta para a ação (executando). Essa técnica de executar vários processos de forma concorrente é denominada **multiprogramação** ou **programação concorrente**.

As primeiras versões do Windows (3.1 e 3.11) trabalhavam desta forma. Considerando que quase todos os processos utilizavam muito dispositivo, todos conseguiam executar. Quando algum processo monopolizava a CPU, os demais processos ficava bloqueados.

Pr	Tm	Ds	Tm	Cpu	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
p1	04	04	01	x	x	x	x	d	d	d	d	a	x								
p2	02	04	02	a	a	a	a	x	x	b	b	d	d	d	d	x	x				
p3	03	04	01	a	a	a	a	a	a	x	x	x	b	b	b	d	d	d	d	x	
unid.	tempo:			0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	

Pr: processo
Tm: unidade de tempo
Ds: dispositivo
x: executando
d: dispositivo
a: apto
b: bloqueado

Um ciclo de vida com fatias de tempo - round robin.

Este ciclo envolve chamadas de sistema (system call) e interrupção.

1. criado

2. executando

2.1. system call (chamada para um dispositivo)

2.2. bloqueado - o processo vai para fila de processos bloqueados

2.3. interrupção (o dispositivo entrega por DMA, a requisição feita)

2.4. apto - o processo sai da fila de bloqueados e vai para fila de aptos

2.5. executando - ao chegar a sua vez, o processo volta a executar

3. terminando

O processo pode ainda...

...gerar uma **trap** no passo 2 e ir direto para o passo 3. A trap é uma interrupção de emergência causada por alguma violação feita pelo processo (divisão por 0, etc).

...esgotar sua fatia de tempo (**time slice**) e passar do passo 2 para o passo 2.4 onde vai esperar sua vez de executar novamente.

Um processo está apto quando ele tem todas as condições de executar, mas não o faz porque tem outro processo executando no momento. No **round robin** o processo executa por uma fatia de tempo que é medida em milissegundos.

As versões mais recentes do Windows, com kernel NT (XP, Vista, 7, 8, 10, 20, 30...), trabalham desta forma.

Pr	Tm	Ds	Tm	TimeSlice=2										CpuOciosa=4Tm									
p1	04	04	01	x	x	a	a	a	a	x	x	d	d	d	d	x							
p2	02	04	02	a	a	x	x	d	d	d	d	x	x										
p3	03	04	01	a	a	a	a	x	x	a	a	a	a	x	b	d	d	d	d	x			
unid.	tempo:			0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6			

Pr: processo
Tm: unidade de tempo
Ds: dispositivo
x: executando
d: dispositivo
a: apto
b: bloqueado

Um ciclo de vida com fatias de tempo - round robin e preemptivo.

Este ciclo envolve chamadas de sistema (system call) e interrupção.

- 1. criado
- 2. executando
 - 2.1. system call (chamada para um dispositivo)
 - 2.2. bloqueado - o processo vai para fila de processos bloqueados
 - 2.3. interrupção (o dispositivo entrega por DMA, a requisição feita)
 - 2.4. apto - o processo sai da fila de bloqueados e vai para fila de aptos
 - 2.5. executando - ao chegar a sua vez, o processo volta a executar
- 3. terminando

O processo pode ainda...

- ...gerar uma **trap** no passo 2 e ir direto para o passo 3. A trap é uma interrupção de emergência causada por alguma violação feita pelo processo (divisão por 0, etc).
- ...esgotar sua fatia de tempo (**time slice**) e passar do passo 2 para o passo 2.4 onde vai esperar sua vez de executar novamente.
- ...ser **bonificado** com uma **prioridade maior**, sendo executado mais do que os outros, pois só **libera a cpu** para processos de menor prioridade **se este processo estiver bloqueado**.
- ...ser **penalizado** com uma **prioridade menor**, sendo executado menos do que os outros, pois **só executa se todos os processos de prioridade maior estiverem bloqueados**.

O processo de **prioridade menor** é **preemptivo**, pois **dá a preferência** para o processo de **prioridade maior** executar na frente dele.

Pr	Tm	Ds	Tm	TimeSlice=2										CpuOciosa=5Tm										Prior
p1	04	04	01	x	x	x	x	d	d	d	d	x								alta				
p2	02	04	02	a	a	a	a	x	x	b	b	d	d	d	d	x	x			baixa				
p3	03	04	01	a	a	a	a	a	a	x	x	b	x	b	b	d	d	d	d	x	baixa			
unid.	tempo:			0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6				

Pr: processo
Tm: unidade de tempo
Ds: dispositivo
x: executando
d: dispositivo
a: apto
b: bloqueado