**Aula 01**

**Pesquisar: microkernel, timeshare, monolítica, multitarefa, monotarefa, lote.**

SO: top down, bottle up

**bottle up, eficiência.**

escalonamento da CPU

gerência de memória

gerência E/S

**topdown, comodidade.**

**Preempção**

Em sistemas operacionais, preemptividade ou preempção é a capacidade de tirar de execução um processo em favor de outro. Esta é uma característica que não é importante apenas nos sistemas operacionais em tempo real.

**Traps**

Funções que manipulam mais o hardware precisam do modo núcleo.

Ex: T1.c contém - fprintf (interrompe) → salva o contexto → modo núcleo → lista system call → rotina → modo user → recupera o contexto → T1.c

**Interrupções**

Um evento que ocorre em nível de hardware (entrada e saída de um pendrive por exemplo). pode ter sido causado por algo que ainda não tenha sido resolvido.

Ex: Programa sofre interrupção → salva o contexto → modo núcleo → interrupção → rotina → modo user → recupera o contexto → Programa.

**Programas X Processos**

Um programa de computador é uma coleção passiva de instruções, enquanto que um processo é a execução real dessas instruções. Vários processos podem ser associados com o mesmo programa. Por exemplo, abrir várias instâncias do mesmo programa geralmente significa que mais de um processo está sendo executado.

**Programas**

Lista de instruções/ comandos

**processos**

Instância de execução do programa.

PCB → Bloco de Controle de Processos

**Término de processo**

1. Fim natural (voluntário)
2. Erro não fatal (voluntário) não atingiu o objetivo dele.
3. Morte por outro processo (involuntário)
4. Erro fatal (involuntário

**Ciclo de vida de um processo**

Novo processo → **Pronto** →

**Execução (CPU)** → (Preempção) → **Pronto**

**ou**

**Execução** → (E/S stop/wait) → **Bloqueado/ dormindo → Pronto**

**ou**

**Execução** → **Termina**

**Pronto Suspenso ←→ Pronto**

**Bloqueado/ dormindo ←→ Bloqueado/ dormindo Suspenso**

**Bloqueado/ dormindo Suspenso → Pronto Suspenso**

p1 — —

p2 — —

p3 — —

so - - - - -

p1 —

p2 —

p3 —

so -

processo — - — - — - — - — - —

**Suspensão é quando o processo vai para a memória secundária e tem menor importância.**

Threads of execution = linhas de execução

Quer execução paralela

Threads tem ciclo de vida similar ao processo, mas não tem o estado de suspensão porque quem vai ser suspenso é o processo.

**1:n:** um processo pode ter **n** threads, mas o SO enxerga como uma thread.

**n:n:** tem um conhecimento de todas as threads, cria a possibilidade de paralelismo real. Múltiplas threads do mesmo processo. Ex: Windows, Linux.

**n:m:** Identifica parcialmente as thread. Pode juntar as threads. Não vê todas as threads, faz algumas junções. Mais complexa e que divide a responsabilidade, parte é feita no SO parte no próprio processo. Ex: Solaris.

Destruir uma thread é mais rápido que um processo.

**Sistemas operacionais monoprogramados/monotarefas**

Permitem que apenas um programa seja armazenado na memória para execução. Os demais programas devem aguardar o término da execução do programa que está em memória.

* É executado por um único processador e é capaz de gerenciar a execução de um único programa (tarefa) do usuário por vez.
* Permite que o processador, a memória e os periféricos fiquem dedicados a um único usuário; são portanto monousuários (monoterminais).
* O processador fica ocioso quando o programa espera pela ocorrência de uma E/S.
* São sistemas de simples implementação.

CPU = Rápida

MEM = Intermediária

E/S = Lenta

**Sistemas operacionais multiprogramados/multitarefas**

Os sistemas multiprogramados podem ser classificados de duas formas: - Pelo número de usuários que interage com o sistema operacional; - Pela forma como as tarefas são gerenciadas;

CPU = Rápida

MEM = Intermediária

E/S = Lenta

**IPC - Condição de corrida.**

Uma condição de corrida **ocorre quando dois threads acessam uma variável compartilhada ao mesmo tempo**. O primeiro thread lê a variável e o segundo thread lê o mesmo valor da variável. (É extremamente raro de acontecer, mas é perigoso).

Thread A

printf(“IPC”)

Buffer[in] = “IPC”

in++;

antes do in++ executa o B, por isso o problema porque os dois comandos printf foram executados antes do incremento. Uma operação acabou sendo apagada.

Thread B

Printf(“Corrida”);

Buffer[in] = “corrida”

in++;

0 “Hello”

1 “word”

2 “so’

3 “threads”

4 IPC

5 “corrida”

in: 4 (posição)

**Exemplo em Assembly:**

saldo = 500

ld R1 saldo R1 = 500

add R1 300 R1 = 800

sto saldo r1 R1

saldo = saldo + 300

ld R1 saldo

sub r1 100

sto saldo r1

saldo = saldo - 100

Não pode ter preempção e paralelismo real entre as Threads, porque se não dão valores inconsistentes e a “condição de corrida”.

Uma operação sobrescreve a outra porque foi feita ao mesmo tempo, (problema difícil de acontecer).

**Região crítica**

Região Crítica **é a área de um código ou recurso compartilhado que depende expressivamente que o acesso à mesma seja realizado de maneira sequencial**. Porém, sabendo da possibilidade de condição de corrida, essa região deverá ser tratada de maneira especial de maneira a evitar/tratar condições de corrida entre processos.

Local do código a corrida onde faz o acesso a variável (corrida)

**Exclusão mútua**

Exclusão mútua é a propriedade de um programa que garante que somente um processo tem acesso a determinada variável compartilhada em cada momento quando isso for necessário à correção do programa. É a solução mais simples para se obter a semântica não-determinística de um programa paralelo**.**

**não pode permitir q n threads acessem**

Diferença entre elas

**Região crítica:** ocorre quando duas ou mais threads têm acesso a memória compartilhada.

**Exclusão mútua:** estratégias para garantir que não terá mais que uma thread na região crítica.

NubANK

**Regras para evitar corrida em programação concorrente**

1° Regra - Garantir a exclusão mútua.

2° Regra - Progressão - (uma thread não pode impedir outra thread de acessar a região crítica)

quem não está na crítica não pode impedir quem está de entrar

3° Regra - Espera limitada (uma thread não deve esperar para entrar na região crítica)

4° Regra - Hardware - (funcionar em qualquer sistema)

**Espera ocupada**

**Desabilitar interrupções** (eventos de fora do computador) - quebra a regra 4

NÃO FUNCIONA - problema exclusão mútua

**Variável de trava**

NÃO FUNCIONA - problema espera limitada

**Alternância**

NÃO FUNCIONA - problema de progressão (mas não ocorre corrida), nunca vai poder executar em paralelo as Threads.

**Solução de Peterson**

FUNCIONA, MAS NÃO 100% - erros pequenos

**Semáforo**

Contador positivo

Incremento e decremento

FUNCIONA.

Contador de recursos

syncronized void rc (){

}

Uma solução mais simples e exclusiva para E/S de região crítica.

**Semáforo de propósito real**

Problema do produtor consumidor / buffer limitado

buffer como vetor

| buffer | 0 1 2 3 4 | | | | |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |

semaphore free = new semaphore

semaphore occupied = new semaphore

**Jantar dos filósofos**

Deadlock

Impasse

O que ocasiona o deadlock?(pergunta

1 - Exclusão

2 - Uso e espera

3 - Não preempção de recursos

4 - Espera circular

**PERGUNTAS DA APOSTILA**

1. Quais os dois principais objetivos de um sistema operacional?

2. Quais as diferenças entre interrupções, exceções e traps?

Programa: é um conjunto de uma ou mais sequências de instruções escritas para resolver um problema específico, constituindo assim uma aplicação ou utilitário.

Tarefa: é a execução sequencial, por um processador, da sequência de instruções definidas em um programa para realizar seu objetivo

**Definição de thread**

Uma thread é definida como sendo um fluxo de execução independente. Um processo pode conter uma ou mais threads, cada uma executando seu próprio código e compartilhando recursos com as demais threads localizadas no mesmo processo. Cada thread é caracterizada por um código em execução e um pequeno contexto local, o chamado Thread Local Storage (TLS), composto pelos registradores do processador e uma área de pilha em memória, para que a thread possa armazenar variáveis locais e efetuar chamadas de funções.

Data: 19/10

Escalonamento de Processos

Algoritmos de escalonamento

algoritmos preemptivos, não preemptivos(pesquisar)

Algoritmo de escalonamento

**Não preemptivo**

**FCFS (First comf first served) ou FIFO, processo mais antigo**

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 4 | 4 | 4 | 0 | 34/5 = 6,8 | 19/5 = 3,8 |
| 2 | 1 | 3 | 7 | 6 | 3 |
| 3 | 2 | 1 | 8 | 6 | 5 |
| 4 | 3 | 2 | 10 | 7 | 5 |
| 5 | 4 | 5 | 15 | 11 | 6 |

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 4 | 4 | 4 | 0 |  | 19/5 = 3,5 |
| 2 | 1 | 5 | 9 | 8 | 3 |
| 3 | 2 | 2 | 11 | 9 | 7 |
| 4 | 0 | 3 | 14 | 14 | 11 |

1 0 0 4

2 0 1 5

3 0 2 2

4 1 0 3

AT arrival time - tempo de chegada

BT bash time - tempo de execução

CT completion time - tempo de conclusão

TAT turn arround time - tempo no sistema TAT = CT - AT

WT wait time - tempo de espera WT = TAT - BT

**Gráfico de Gantt**

p1 p2 p3 p4 p5

0 4 7 8 10 15

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | 0 | 16 | 16 | 16 | 0 |  |
| B | 0 | 7 | 23 | 23 | 16 |  |
| C | 0 | 3 | 26 | 26 | 23 | 39/3=13 |

Tempo médio de espera

1. 7,66
2. 26
3. 13 opção correta
4. 11,5
5. 5

A B C

0 16 23 26

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 3 | 3 | 3 | 0 | 41/6=6,83 | 24/6 = 4 |
| 2 | 1 | 2 | 5 | 4 | 2 |
| 3 | 2 | 1 | 6 | 4 | 3 |
| 4 | 3 | 4 | 10 | 7 | 3 |
| 5 | 4 | 5 | 15 | 11 | 6 |
| 6 | 5 | 2 | 17 | 12 | 10 |

1 2 3 4 5 6

0 3 5 6 10 15 17

**SJF = shortest job first**

**Não preemptivo - Menor tempo de execução**

**inviável, porque o tempo de execução não é previamente conhecido**

é mais teórico e não prático

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 7 | 8 | 7 | 0 | 11 | 6,4 |
| 2 | 2 | 5 | 16 | 14 | 9 |
| 3 | 3 | 1 | 9 | 6 | 5 |
| 4 | 4 | 2 | 11 | 9 | 7 |
| 5 | 5 | 8 | 24 | 19 | 11 |

BT Não é conhecido previamente **Pergunta de prova**

1 3 4 2 5

0 1 8 9 11 16 24

menor tempo de execução = p1(7) p2(5) p3(1) p4(2) p5(6)

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 4 | 4 | 4 | 0 | 6,2 | 3,2 |
| 2 | 1 | 3 | 10 | 9 | 6 |
| 3 | 2 | 1 | 5 | 3 | 2 |
| 4 | 3 | 2 | 7 | 4 | 2 |
| 5 | 4 | 5 | 15 | 11 | 6 |

1 3 4 2 5

0 4 5 7 10 15

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 4 | 9 | 9 |  |  | 19/5 = 3,5 |
| 2 | 1 | 5 | 14 | 13 |  |
| 3 | 2 | 2 | 5 | 3 |  |
| 4 | 0 | 3 | 3 | 3 |  |

P4 P3 P1 P2

0 3 5 9 14

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 3 | 3 | 3 | 0 | 5,8 | 3 |
| 2 | 1 | 2 | 6 | 5 | 3 |
| 3 | 2 | 1 | 4 | 2 | 1 |
| 4 | 3 | 4 | 12 | 9 | 5 |
| 5 | 4 | 5 | 17 | 13 | 8 |
| 6 | 5 | 2 | 8 | 3 | 1 |

p1 p3 p2 p6 p4 p5

0 3 4 6 8 12 17

p1(3) p2(2) p3(1) p4(4) p5(5) p6(2)

A regra de desempate caso existam dois iguais é o AT.

**H R R N**

Highest response ratio next

RR = (WT + BT) / BT

WT atual.

seleciona o que tem maior tempo de resposta

maior problema que o bt nao e conhecido

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 3 | 3 | 3 | 0 | 5,8 | 3 |
| 1 | 2 | 6 | 9 | 7 | 1 |
| 2 | 4 | 4 | 15 | 11 | 6 |
| 3 | 6 | 5 | 22 | 16 | 11 |
| 4 | 8 | 2 | 17 | 9 | 7 |

Causa starvation

p0 p1 p2 p4 p3

0 3 9 13 17 22

p0 = 3 + 0 / 3 = 1

p1 = 6 + 1 / 6 = 1,1

p2 = 4 + 5 / 4 = 2

p3 = 5 + 3 / 5 = 1,6

p4 = 2 + 1 / 2 = 1,5

p3 = 5 + 9 / 5 = 2,8

p4 = 2 + 7 / 2 = 4,5

**Data 26/10**

**SRTN - shortest remaining time next**

SJT preemptivo

menor tempo de execução

p1 p2 p3 p4 p5 p1 p5

0 1 2 3 4 6 10 16 24

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 7 | 16 | 15 | 8 | 9 | 4,4 |
| 2 | 2 | 5 | 10 | 8 | 3 |
| 3 | 3 | 1 | 4 | 1 | 0 |
| 4 | 4 | 2 | 6 | 2 | 0 |
| 5 | 5 | 3 | 24 | 19 | 11 |

Exercicio

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 4 | 5 | 5 | 1 | 7 | 2  3 |
| 2 | 1 | 3 | 10 | 9 | 6 |
| 3 | 2 | 1 | 3 | 1 | 0 |
| 4 | 3 | 2 | 7 | 4 | 2 |
| 5 | 4 | 5 | 15 | 11 | 6 |

p1(3) p3(2) p1(0) p4(p0) p2(0) p5(0)

0 2 3 5 7 10 15

AT arrival time - tempo de chegada

BT bash time - tempo de execução

CT completion time - tempo de conclusão

TAT turn arround time - tempo no sistema TAT = CT - AT

WT wait time - tempo de espera WT = TAT - BT

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 3 | 3 | 3 | 0 |  |  |
| 2 | 1 | 2 | 6 | 5 | 3 |
| 3 | 2 | 1 | 4 | 2 | 1 |
| 4 | 3 | 4 | 12 | 9 | 5 |
| 5 | 4 | 5 | 17 | 13 | 8 |
| 6 | 5 | 2 | 8 | 3 | 1 |

p1(2) p3(1) p2(0) p6(0) p4(0) p5(0)

0 3 4 6 8 12 17

| **ID** | **AT** | **BT** | **CT** | **TAT** | **WT** | **AVG TAT** | **AVG WT** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 3 | 3 | 3 | 0 |  |  |
| 1 | 2 | 6 | 15 | 13 | 7 |
| 2 | 4 | 4 | 8 | 4 | 0 |
| 3 | 6 | 5 | 20 | 14 | 7 |
| 4 | 8 | 2 | 10 | 2 | 0 |

p0(2) p1(1) p2(0) p4(0) p1(0) p3(0)

0 3 4 8 10 15 20

Prioridade Preemptivo e não preemptivo

Estática e dinâmica: com o processo vem uma informação de prioridade, escolha baseada na prioridade.

SJF, SRTN

**Gerência de memória**

Recursos limitados

Executar instrução

mantém somente o que é usado na memória principalvirtur

espaço suspenso da memória principal para secundária, (SWAP)

Mapeamento da memória

**bit map, lista encadeada que o SO tem para gerenciar os espaços ocupados e livres pela memória.**

**First fit =** Mais rápido e simples, cabe ali, vai ali mesmo.

**Next fit =** Continua a partir do último espaço livre.

**Best fit =**  Decide o que sobrar menos blocos, procura pelo melhor encaixa

**Worst fit =** A sobra pode ser útil no futuro, já que se sobrar apenas um espaço, aquele espaço dificilmente será usado novamente

**Virtualização da memória.**

**Paginação =** Tabela de páginas, a memória principal é dividida em quadros e um quadro tem o tamanho de uma páginas, é muito comum que elas tenham tamanho de 4kb,.

**Segmentação =**

**o que é abstração de memória**

virtual fisicos

1 2 3 4 5 —----------> 57

1 4 1 4 1 —---------->1833

2 4 5 6 7 \_\_\_\_\_\_\_\_> 16375

4 6 0 0 0 —---------->

3 2 7 8 0

divide por 4096 para saber em que quadro está

paginação por demanda SO

algoritmo ótimo, o melhor resultado possível = só funciona para simulações, sempre vai gerar a melhor paginação

alogoritmo real = FIFO

lidos por humanos

lidos por máquinas

comunicação

CPU - MEMÓRIA - DISPOSITVOS E/S

Serial

Paralela

PONTO NORTE

CPU

PONTO SUL

PONTO

CPU

MULTIPONTO

programava

interrupção

DMA

questões de entrada e saída