

Resumo 50 - 1^a prova -

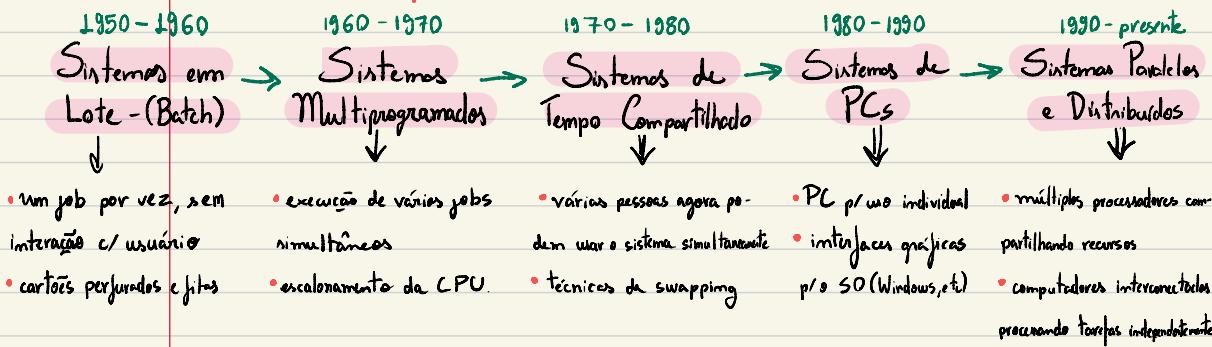
Sophia Carrazza

Job = conjunto de instruções, processos
 ou tarefas que o SO deve executar
 (um programa em execução)
 Job = processo sequencial

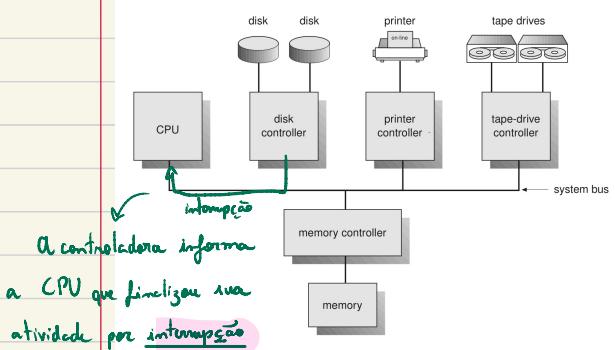
Introdução - SO:

↳ SO: É um programa que age como intermediário entre o usuário e o hardware, facilitando o uso dos recursos.
 - ele aloca e gerencia os recursos eficientemente, controlando o uso do hardware entre diversas aplicações.

↳ Linha Temporal dos SOs:



↳ Estrutura de um sistema computacional:



Obs: trap é uma interrupção que normaliza erros!

• Dispositivos de I/O e a CPU podem executar de forma concorrente

↳ quando múltiplos tarefas procedem de forma intercalada.

• Cada controladora tem um buffer local, e a CPU transfere os dados entre esses buffers e a RAM.

* SOs modernos são baseados em interrupções!

↳ elas transferem o controle para a rotina de serviço da interrupção.

Hardware:

DMA → Acesso direto à memória. Usado p/ permitir que dispositivos I/O transfiram dados em velocidade próxima à memória (direto do buffer p/ a memória, sem interrupção da CPU).

Caching → Cópia de informação em diferentes níveis de memória.

Modo Monitor / Supervisor → Executa à favor do S.O., e não do usuário (modo usuário). Instruções privilegiadas só podem ser executadas em modo supervisor.

- ↳ 1 bit de modo indica o modo atual (usuário = 0 / supervisor = 1)
 - ↳ Instruções privilegiadas:
 - instruções de I/O
 - load para os registradores de base e limite
 - carregar o timer
- menor endereço físico legal
tamanho da faixa legal
interrupção o captador depois não irá mais ser gerada
de um tempo da memória

Escalonamento:

Escalonamento de longo prazo: seleciona quais processos devem ser levados p/ memória na fila de processos prontos.

Escalonamento de curto prazo: seleciona qual processo deve ser executado e alocado à CPU.

Swapping: inserção e remoção do processo na memória

Overhead: tempo necessário p/ a troca de contexto, que é um gasto extra de tempo.

Utilização máxima da CPU: só pode ser obtida através da multiprogramação, com mais de um processo em execução.

precisamos  

Critérios de escalonamento:

- > Utilização da CPU = manter a CPU o + ocupada possível \oplus
- > Throughput = # de processos que completaram sua execução por un. de tempo \oplus
- > Tempo de Retorno = qtd de tempo p/ executar um processo \ominus
- > Tempo de Espera = qtd de tempo que um processo gasta na fila de processos prontos \ominus
- > Tempo de Resposta = qtd de tempo gasto entre a requisição e a produção da 1^a resposta \ominus
- > Quantum = intervalo máx de tempo que um processo pode executar antes que o escalonador interrompa e passe p/ o próximo processo.
- > Vazão = número de processos que conseguimos atender por un. de tempo
- > Eficiência = $\frac{\text{Tempo útil CPU}}{\text{Tempo total CPU}}$ (executando algum processo)
- > Ciclos de Susto = é a execução de um processo. O burst time é o tempo de execução.

$$\text{Tempo de Retorno} = \text{Tempo de término} - \text{tempo de chegada}$$

$$\text{Tempo de Espera} = \text{Tempo de retorno} - \text{tempo de execução}$$

$$\text{Tempo de Resposta} = \text{tempo p/ a 1^a resposta} - \text{tempo de chegada}$$

Escalonamento Preemptivo vs Escalonamento Não-Preemptivo

- ↳ a CPU pode ser retirada de um processo em execução e entregue a outro mais prioritário
- ↳ RR, SJF preemptivo e Prioridade preemptiva

Escalonamento Não-Preemptivo

- ↳ um processo, uma vez em execução, continua até terminar (ou entrar em estado de espera).
- ↳ FCFS, SJF e Prioridade preemptiva

FCFS (First Come, First Served):

- ↳ o primeiro a chegar é servido
- ↳ processos executados na ordem em que chegaram.
- ↳ efeito comboio: pequenos processos atráçã de longos processos

RR (Round Robin):

Quantum!

→ cada processo tem uma quantidade de tempo p/ ser executado (um quantum). Se esse tempo for alcançado, o processo é retirado p/ o fim da fila de pronto.

→ para n processos e um quantum q:

- cada processo terá $\frac{1}{n}$ de CPU em pacotes de no máximo q unidades de tempo por vez.
- nenhum processo esperará mais que $(n-1) \cdot q$ unidades de tempo.

SJF Não - Preemptivo (Shortest Job First)

→ o processo c/ menor tempo de execução é escolhido primeiro

→ o processo executa até o fim, sem interrupções

SJF Preemptivo

→ mesmo que o anterior, mas se, se um novo processo com tempo menor chegar, ele interrompe o atual e faz a troca.

→ se não houver fila de pronto

SJR

* SJF é ótimo
(tempe dás o mínimo tempo médio de espera p/ o cagunto de processos).

Escalonamento Por Prioridade

→ a CPU é alocada p/ o processo com maior prioridade (menor valor = maior prioridade)

→ o não-preemptivo executa cada processo ate' o fim

→ o preemptivo faz a troca se chegar um + prioritário

→ ex: SJF

→ problema → Starvation (alguns processos com baixa prioridade podem nunca serem executados)

→ solução: aging (a prioridade aumenta c/ o tempo).

Escalonamento de Processos Periódicos:

* Um sistema é escalonável se todos os processos puderem ser executados dentro dos seus respectivos períodos.

- ↳ cada processo deve finalizar execução antes do final de seu período
- ↳ o tempo total necessário p/ executar todos os processos não pode exceder o intervalo de tempo em que todos os períodos se repetem (o MMC dos períodos).

1°	240, 200, 100	4
*	60, 50, 25	2) 8
	30, 25, 25	2 / 16
	15, 25, 25	5)
	3, 5, 5	5)
	3, L, L	3 /
	L, L, L	1200

Exemplo



2° * Determinaremos a carga total de sistema (tempo necessário de execução)

b) p/ cada Ponto:

$$\rightarrow \text{nº de repetições} = \frac{\text{MMC}}{\text{Período}}$$

$$\rightarrow \text{tempo necessário} = \text{nº de rep.} \cdot \text{temp. exec.}$$

3° * Depois, calculamos o tempo total necessário de todo sistema incluindo o de X.

$$\text{MMC} \cdot T_T = T_{P_1} + T_{P_2} + T_{P_3} + T_{P_X}$$

4° * Agora p/ encontrar X, voltamos na função de

$$\frac{\text{MMC}}{X} \leq \frac{T_P}{\text{temp. exec.}}$$



Grafo de Alocação de Recursos

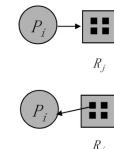
♦ Processo



♦ Tipo de recurso com 4 instâncias



♦ P_i solicita instância de R_j



♦ P_i mantém instância de R_j



Dead lock:

↳ Para ocorrer um deadlock, as seguintes condições devem ser verdadeiras:

- > exclusão mútua = só um processo por vez pode usar o recurso
- > máx - preempção = um recurso não pode ser liberado se ele finalizar sua execução, voluntariamente.
- > passe e espera = um processo em passe de polo tem um recurso e está esperando por recursos adicionais em posse de outros processos
- > espera circular = existe um círculo de processos em espera em que P₀ espera por um recurso em posse de P₁, P₁ de P₂, ... P_n de P₀. Assim, forma-se uma espera circular de recursos.

Semáforo: mecanismo de sincronização provida pelo SO, de maior nível que thread e o lock.

↳ é uma variável inteira, operada somente a partir de duas operações atómicas:

- wait() = tenta decrementar o valor do semáforo
- signal() = incrementa o semáforo. Se ele for = 0, irá verificar quais processos estavam esperando e vai desbloquear os processos que estavam bloqueados pelo wait().

```
• wait(s)
    se s > 0
    então s = s - 1
    senão o processo é suspenso
```

```
• signal(s)
    se algum processo P está suspenso
    então acorde P
    senão s = s + 1
```