# MAC5753 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 19 de Novembro de 2020

# Roteiro



## Algoritmos para alocação de memória

## Memória Virtual

# Algoritmos para alocação de memória

# Como alocar o espaço livre?



- Podem existir diversos locais para acomodar um processo na memória. Qual deles é o melhor local vai depender do histórico de execuções de processos na máquina, da quantidade de memória disponível e do método usado (lista encadeada ou mapa de bits)
- Os algoritmos apresentados a seguir fazem sentido tanto com mapa de bits quanto com lista encadeada

# First Fit

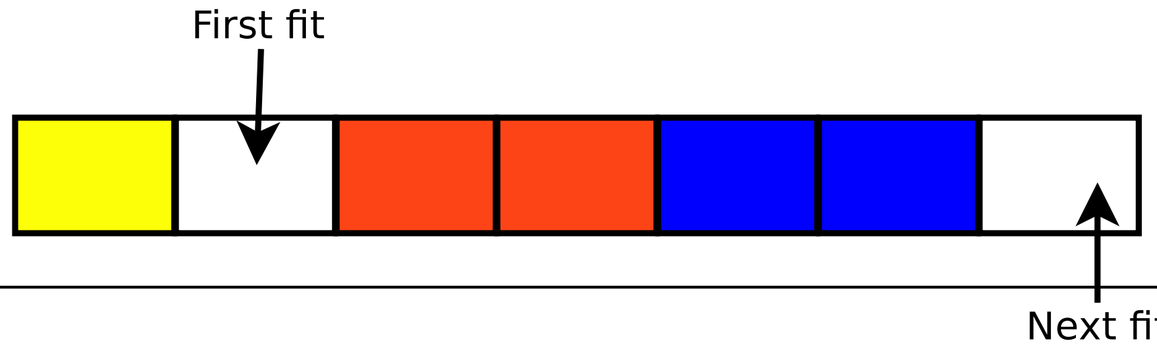


- Busca a partir da primeira posição de memória, de forma sequencial, e aloca para o processo o primeiro espaço suficiente encontrado. **Mais simples**

# Next Fit



- Similar ao First Fit com a diferença de que nem sempre começa da primeira posição. Vai começar a partir do ponto onde a busca terminou ao alocar espaço para o último processo. **A ideia é evitar buscar espaço livre onde provavelmente não há.** No exemplo abaixo, com um espaço livre na segunda unidade de alocação de memória, se um processo solicitando duas unidades tiver sido alocado na posição 5, a alocação para um próximo processo exigindo apenas 1 unidade seria feita na segunda unidade de alocação com o First Fit e na unidade de alocação 7 com Next Fit.



# Best Fit



- Busca a partir da primeira posição de memória e varre a memória inteira em busca do espaço livre que tenha o menor tamanho suficiente para acomodar o processo. **A ideia é evitar quebrar espaços livres muito grandes – que podem ser necessários no futuro e raros de encontrar. Apesar de fazer um uso melhor da memória, é ruim pelo fato de ter que varrer a memória inteira**.
- Mas no fim das contas pode ser pior por criar buracos muito pequenos que não servirão para nada
- Obs.: na prática pode ser que a ideia dos algoritmos não consigam ser cumpridas. Apenas simulações com tempos de chegada e tamanhos de processos próximos ao real dirão o melhor algoritmo.

# Worst Fit



- Já que na prática o Best Fit pode criar muitos “buracos” inúteis, o Worst Fit vai na contramão e seleciona o maior espaço dentro todos os possíveis. **Assim como no caso do Best Fit, tem que varrer a memória inteira.**

# Quick Fit



- Mantém listas com os endereços dos espaços livres que comportam os tamanhos mais requisitados
- Poderia ter uma lista dessas listas. Por exemplo, uma lista dos locais na memória onde cabem 4K, uma lista dos locais onde cabem 8K, etc...
- **Eficiente em tempo pois é rápido encontrar espaços para os tamanhos mais pedidos**. Porém, é ineficiente em espaço pela necessidade de manipular uma lista de listas.
- Ineficiente também quando for necessário unir "buracos" sempre que um processo é finalizado. Se não fizer a união a memória ficará fragmentada rapidamente.

# Otimização para todos os algoritmos



- Pode manter listas dos intervalos ocupados e dos intervalos livres
- As listas de intervalos ocupados podem ser ordenados pelo tamanho dos intervalos
- As listas podem ser mantidas na própria memória, nos espaços vazios

# Memória Virtual

# Motivação



- O hardware evolui (mais memória) e com isso os programadores passam a usar mais memória (Se tem, porque não usar?)
- Se todas as exigências de memória de todos os processos tivessem que ser atendidas de fato, dificilmente seria possível rodar todos eles pois a soma das exigências de memória na grande maioria das vezes é maior do que a totalidade de memória física (memória RAM) de um computador
- **Seria interessante permitir que um SO executasse programas que ocupassem mais espaço do que a memória livre num momento**

# Motivação



- Já vimos como rodar múltiplos programas se não há memória física para todos: Usando swap
- Mas swap não é uma boa opção a ser usada sempre porque o acesso ao disco não é tão rápido quanto o acesso à memória
- Na verdade não há como fugir de usar o disco para armazenar os processos que não estejam em execução mas seria interessante reduzir a granularidade do que está na memória e do que está no disco

# Solução inicial



- Nos anos 60: dividir o programa em *overlays*
- Ao inicializar um programa: carrega apenas o gerente de *overlays*
- O gerente de *overlays* carregava e rodava o *overlay* 0
- Sempre que um *overlay* terminava de rodar, carregava o próximo sobre o anterior (se já terminou de rodar então pode ser sobrescrito) ou em outra posição de memória se houvesse espaço livre
- Os *overlays* ficavam em disco e swap era usado aqui mas como não havia necessidade de carregar todo o programa, isso tendia a rodar mais rápido do que se fosse feito swap de tudo de uma vez

# Problema do overlay



- O gerente de *overlays* tinha que ser programado pelo programador do software (O SO só tratava de swap)
- Difícil e muito sensível a erros lidar com um esquema assim
- Seria muito melhor se a responsabilidade fosse passada para o SO