Aula 12

Professores:

Felipe M. G. França

Valmir C. Barbosa

Conteúdo:

Sistema de arquivos

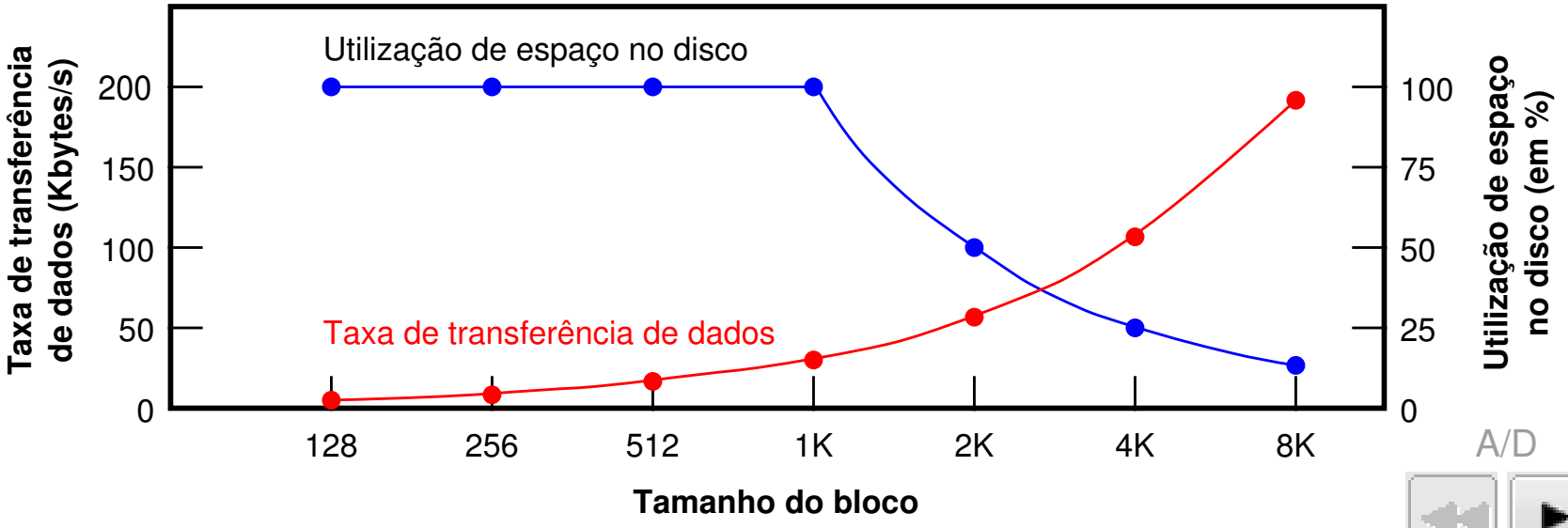
- Implementação (continuação)
- Segurança
- Mecanismos de proteção

Implementação

- Como os arquivos são armazenados em um disco, o gerenciamento do espaço do disco é importante:
  - Uma idéia é a de armazenar um arquivo de *n* bytes em *n* bytes consecutivos do disco.
  - Uma outra idéia é a de dividir o disco em blocos, e armazenar o arquivo como um conjunto (consecutivo) de blocos.
- Armazenar o arquivo em um conjunto consecutivo de bytes ou de blocos nos traz os mesmos problemas vistos com a segmentação:
  - Deveremos mover um arquivo se este aumentar de tamanho.
  - A movimentação dos arquivos no disco causa a **fragmentação externa**, e o custo da compactação é bem maior neste caso.
- Logo os sistemas, em geral, armazenam os arquivos em blocos não necessariamente consecutivos do disco.

Implementação

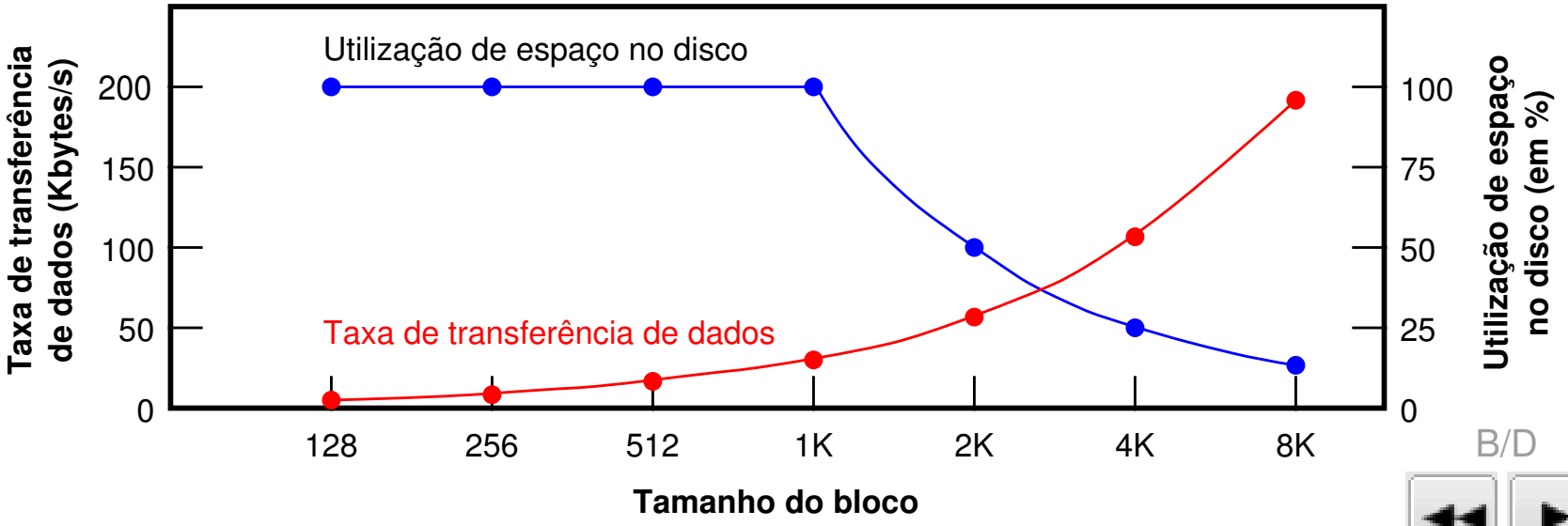
- Ao armazenarmos os arquivos como um conjunto de blocos, um detalhe a ser avaliado é o do tamanho ideal do bloco:
  - Um tamanho grande para o bloco causará um despedício do espaço do disco (com a **fragmentação interna**).
  - Um tamanho pequeno para o bloco fará com que o acesso ao arquivo seja lento, pois este possuirá muitos blocos.



Exemplo do gráfico de utilização do espaço versus taxa de transferência de dados, para um disco em que as trilhas possuem 32768 bytes, o tempo de rotação é de 16,67 ms, e o tempo médio de busca é de 30 ms. Neste disco, o tempo para lermos *k* bytes será de 30+8,335+(*k*/32768)x16,67 ms.

Implementação

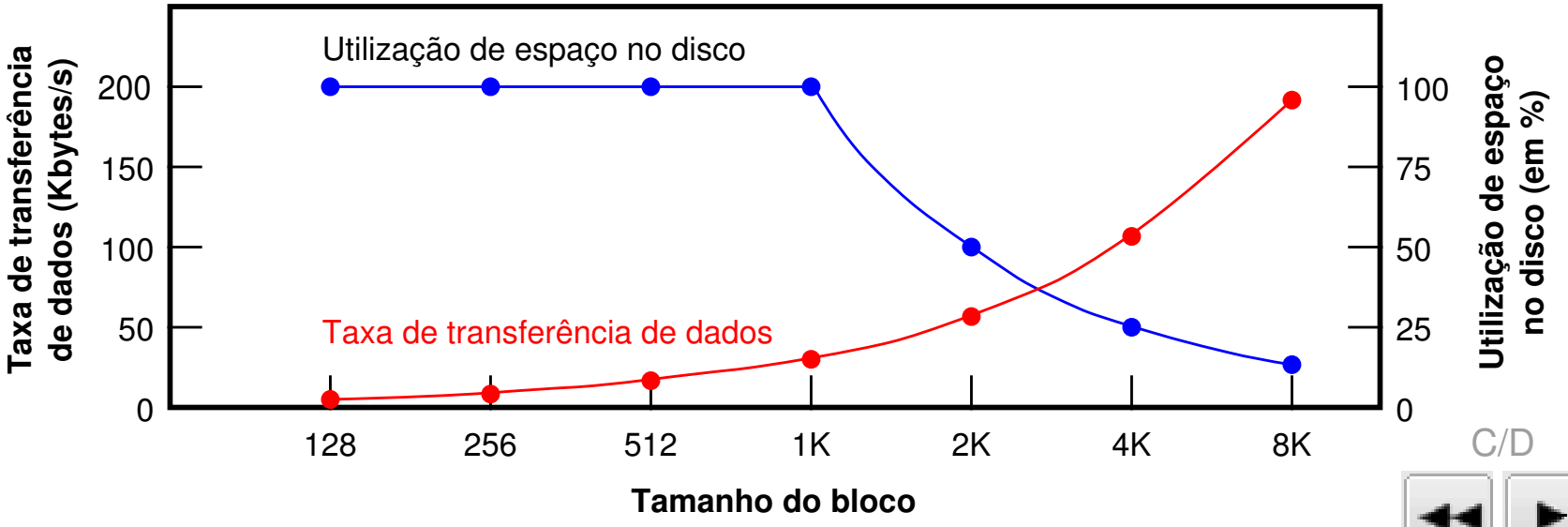
- Ao armazenarmos os arquivos como um conjunto de blocos, um detalhe a ser avaliado é o do tamanho ideal do bloco:
  - Um tamanho grande para o bloco causará um despedício do espaço do disco (com a **fragmentação interna**).
  - Um tamanho pequeno para o bloco fará com que o acesso ao arquivo seja lento, pois este possuirá muitos blocos.



Supondo que os arquivos possuem tamanho médio de 1K, se escolhermos um tamanho menor do que 2K, o desperdício de espaço será menor do que 50%. Tamanhos de blocos grandes não são bons, por exemplo, com um bloco de 8K, o desperdício de espaço no disco será de 87.5%.

Implementação

- Ao armazenarmos os arquivos como um conjunto de blocos, um detalhe a ser avaliado é o do tamanho ideal do bloco:
  - Um tamanho grande para o bloco causará um despedício do espaço do disco (com a **fragmentação interna**).
  - Um tamanho pequeno para o bloco fará com que o acesso ao arquivo seja lento, pois este possuirá muitos blocos.

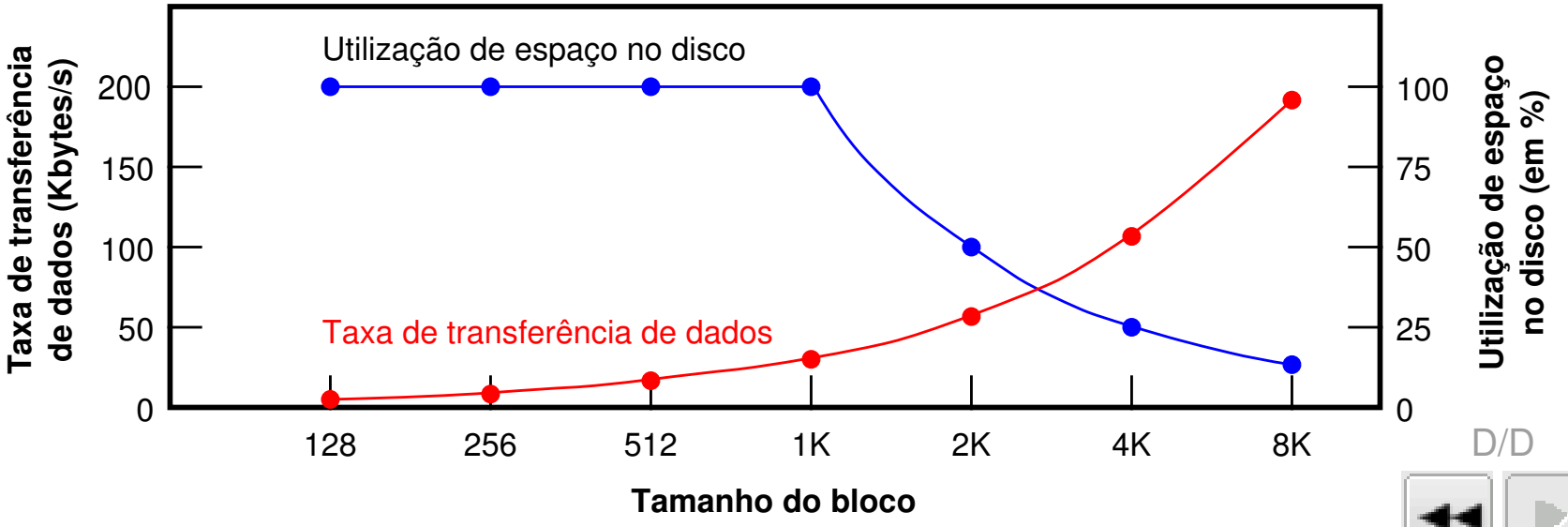


C/D

Como podemos ver pelo gráfico, quanto maior o tamanho do bloco, maior a taxa de transferência. Com um bloco de 8K, a taxa de transferência será bem próxima de 200 Kbytes/s. Logo, um tamanho de bloco pequeno reduz a taxa de transferência.

Implementação

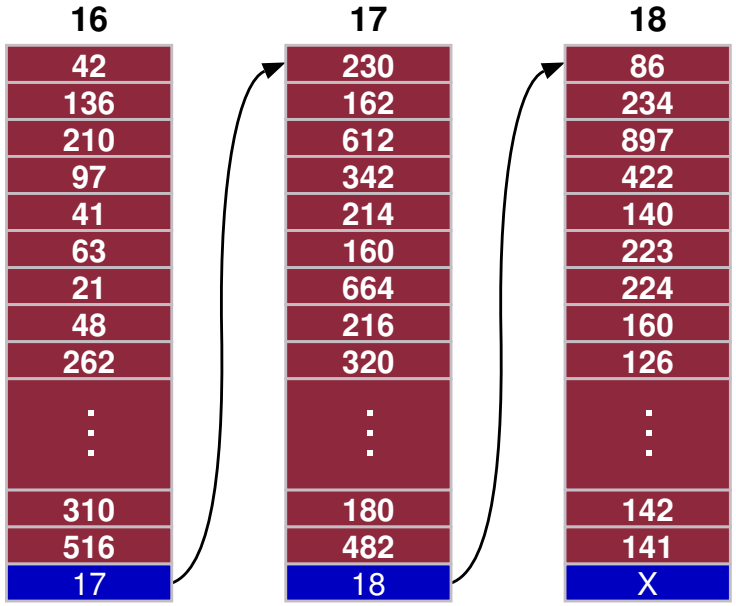
- Ao armazenarmos os arquivos como um conjunto de blocos, um detalhe a ser avaliado é o do tamanho ideal do bloco:
  - Um tamanho grande para o bloco causará um despedício do espaço do disco (com a **fragmentação interna**).
  - Um tamanho pequeno para o bloco fará com que o acesso ao arquivo seja lento, pois este possuirá muitos blocos.



Como vimos, os dois objetivos, ótima utilização de espaço e ótima taxa de transferência são conflitantes: no primeiro os blocos não podem ser muito grandes, e no segundo os blocos devem ser bem grandes. O tamanho ideal depende do tamanho médio dos arquivos, e deve sempre ser reavaliado.

Implementação

- Um outro detalhe ao dividirmos o disco em blocos é o de sabermos quais são os blocos do disco que estão livres:
  - Um modo é o de usarmos uma **lista encadeada de blocos**, onde cada bloco possui um conjunto de blocos livres do disco.
  - Um outro modo é o de usarmos um **mapa de bits**, onde cada bit informa se um bloco está ou não livre.
  - O problema do primeiro modo é que este é mais lento, e o do segundo modo é que a memória gasta pode ser significativa.



Exemplo do uso de uma lista encadeada de blocos. A lista está armazenada nos blocos 16, 17, e 18 do disco, que também são blocos livres. A última entrada de cada lista indica o próximo bloco com a lista, sendo que um 'X' indica que o bloco é o último da lista. Se o tamanho do bloco é de 1K, e se cada endereço de um bloco tem 4 bytes, então cada bloco pode armazenar até 255 blocos livres. Se um disco possui 200M e blocos de 1K, então precisaremos de 804 blocos para armazenar esta lista (pois teremos 200K blocos no disco).

Implementação

- Um outro detalhe ao dividirmos o disco em blocos é o de sabermos quais são os blocos do disco que estão livres:
  - Um modo é o de usarmos uma **lista encadeada de blocos**, onde cada bloco possui um conjunto de blocos livres do disco.
  - Um outro modo é o de usarmos um **mapa de bits**, onde cada bit informa se um bloco está ou não livre.
  - O problema do primeiro modo é que este é mais lento, e o do segundo modo é que a memória gasta pode ser significativa.

Mapa de bits

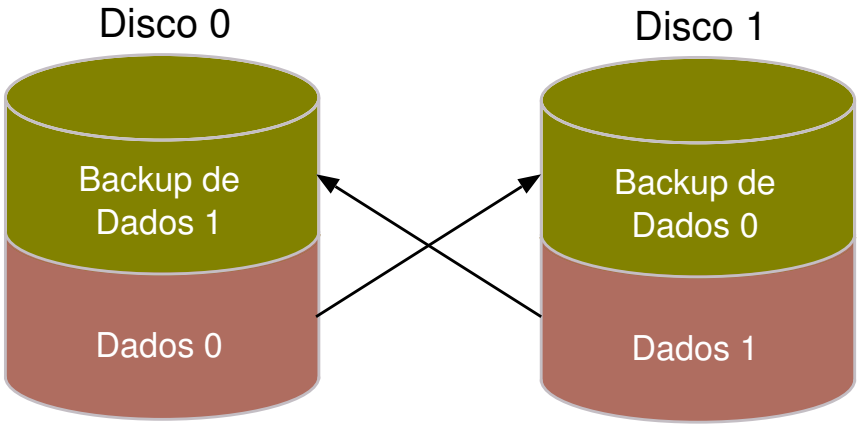
Exemplo do uso de um mapa de bits. Se o disco possuir *n* blocos, então o mapa de bits terá *n* bits. O bit *i* deste mapa está associado ao bloco cujo endereço no disco é *i*, e, ser for, por exemplo, 1, indica que este bloco está livre, e, se for 0, indica que este bloco está sendo usado por algum arquivo. No exemplo anterior de um disco de 200M com blocos de 1K, o mapa de bits deverá ter 200K bits, o que equivale a um espaço de 6.25K se o tamanho das palavras são de 32 bits. Ao alocarmos um bloco livre a um arquivo, deveremos varrer o mapa de bits do início, até encontrar um bit igual a 0.

Confiabilidade

- A perda das informações em um sistema de arquivos é, em geral, mais grave do que falhas nos dispositivos físicos do computador:
  - Se um dispositivo físico não funciona mais, sempre podemos substituí-lo por um novo dispositivo.
  - Recuperar as informações armazenadas em um sistema de arquivos dafinicado pode ser difícil, ou até, impossível.
- Os discos rígidos, em geral, vêm com alguns setores defeituosos, mas também vêm com alguns setores adicionais:
  - Os setores adicionais são usados para substituir os defeituosos.
  - A controladora é a responsável por esta substituição, que é transparente ao sistema e ao usuário.
  - Algumas controladoras informam ao sistema a existência de setores defeituosos, e este pode informar os erros aos usuários.
  - Se o número de setores adicionais se esgotar, o sistema deverá marcar os setores como defeituosos para não usá-los.

Confiabilidade

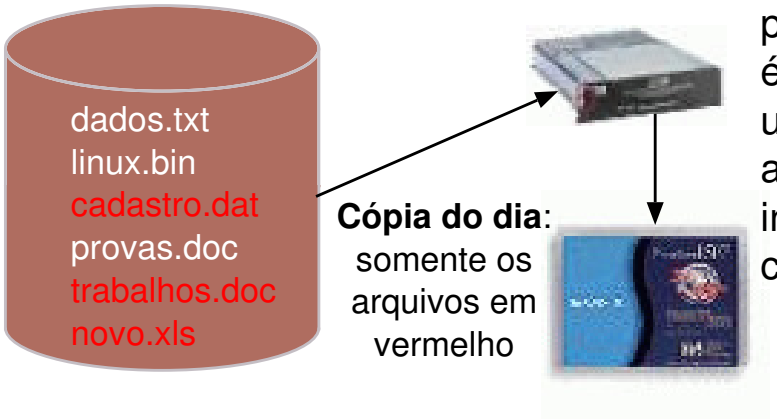
- Um modo de garantir a confiabilidade do sistema de arquivos é o de fazermos ***backups*** periódicos deste sistema:
  - O objetivo de um backup é o de armazenar todos os arquivos do sistema de arquivos em um local seguro e diferente do original.
- Se o sistema de arquivos for pequeno, o backup pode ser feito num disquete, numa fita magnética, ou em um CD.
- Se o sistema de arquivos for grande, o backup completo é lento e incômodo, e, neste caso:
  - Podemos usar dois discos com duas partes: backup e dados.
  - Podemos usar **cópias incrementais**.
  - Podemos usar a técnica do **espelhamento**.



Na primeira técnica, o computador possui dois discos, que são divididos em duas partes: backup e dados. Em um horário de baixo processamento, como à noite, o sistema copia os dados do disco 0 no backup do disco 1, e vice-versa. O problema é que a metade de cada disco é desperdiçada. Se um disco falhar, os dados podem ser recuperados do outro disco com o backup.

Confiabilidade

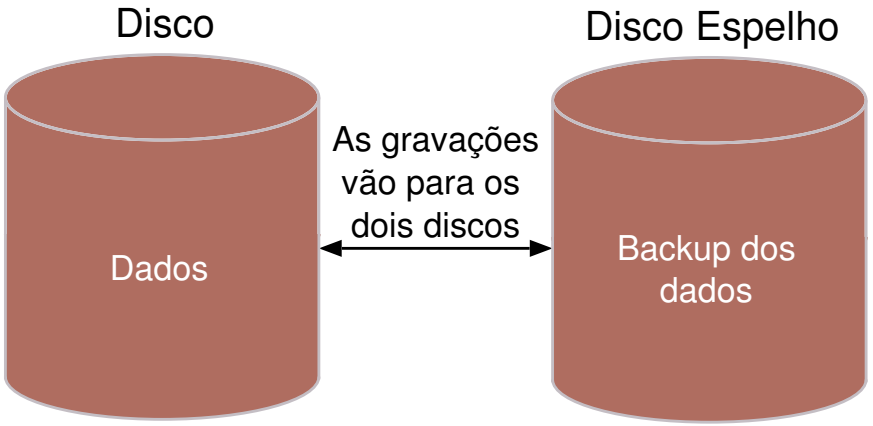
- Um modo de garantir a confiabilidade do sistema de arquivos é o de fazermos ***backups*** periódicos deste sistema:
  - O objetivo de um backup é o de armazenar todos os arquivos do sistema de arquivos em um local seguro e diferente do original.
- Se o sistema de arquivos for pequeno, o backup pode ser feito num disquete, numa fita magnética, ou em um CD.
- Se o sistema de arquivos for grande, o backup completo é lento e incômodo, e, neste caso:
  - Podemos usar dois discos com duas partes: backup e dados.
  - Podemos usar **cópias incrementais**.
  - Podemos usar a técnica do **espelhamento**.



Na cópia incremental, num longo período de tempo, como por exemplo, um mês, todo o sistema de arquivos do disco é copiado para uma media com grande capacidade, como uma fita magnética. A cada dia de um mês, somente os arquivos alterados ou criados após a cópia completa feita no início do mês serão copiados. A eficiência é melhorada se considerarmos a última cópia parcial ao invés da completa.

Confiabilidade

- Um modo de garantir a confiabilidade do sistema de arquivos é o de fazermos ***backups*** periódicos deste sistema:
  - O objetivo de um backup é o de armazenar todos os arquivos do sistema de arquivos em um local seguro e diferente do original.
- Se o sistema de arquivos for pequeno, o backup pode ser feito num disquete, numa fita magnética, ou em um CD.
- Se o sistema de arquivos for grande, o backup completo é lento e incômodo, e, neste caso:
  - Podemos usar dois discos com duas partes: backup e dados.
  - Podemos usar **cópias incrementais**.
  - Podemos usar a técnica do **espelhamento**.



Na técnica do espelhamento o computador possui dois discos idênticos. Todas as gravações são executadas nos dois discos, e a gravação no disco espelho pode ser postergada para ser executada quando o sistema estiver ocioso. O sistema sempre lerá os dados do mesmo disco. Se um dos discos falhar, poderemos substituir o disco defeituoso sem parar o sistema.

Confiabilidade

- Um modo de garantir a confiabilidade do sistema de arquivos é a de garantir a **consistência** deste sistema:
  - Para melhorar a eficiência, muitos sistemas somente gravam os blocos modificados mais tarde.
  - Se o sistema travar antes de blocos importantes serem salvos, o sistema de arquivos pode ficar inconsistente:
    - Exemplos de blocos importantes são os com os nós-i, com os diretórios, ou com a lista de blocos livres do disco.
  - A consistência é verificada através de um programa que avalia todos os blocos e diretórios do disco.
  - A verificação de consistência é feita, em geral, para cada um dos sistemas de arquivo no computador.
  - Também podemos definir regras que protejam o sistema de erros cometidos pelos usuários:

Se um usuário desejava digitar o comando ’rm *.o’ no interpretador de comandos, mas digitou, sem querer, o comando ’rm * .o’ (com um espaço entre ’*’ e ’.’), todos os arquivos do diretório serão removidos, e não somente os arquivos com a extensão o.

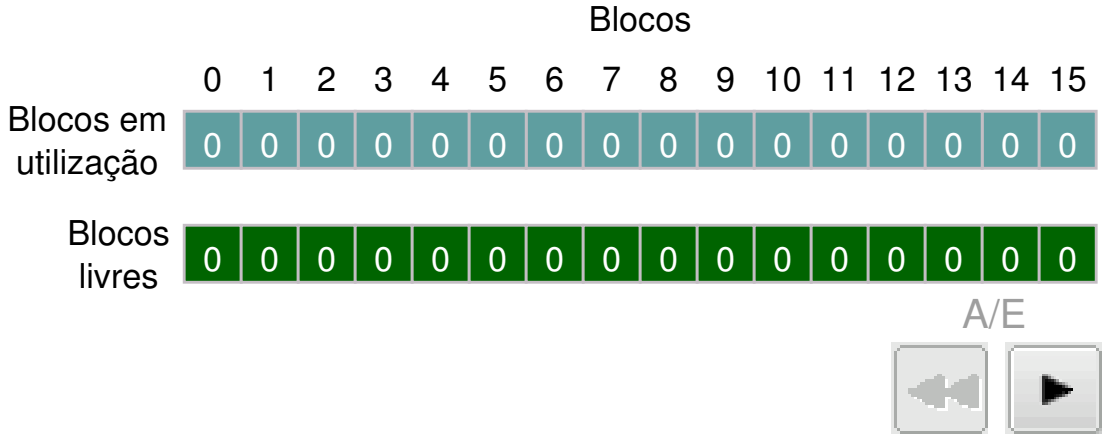
Confiabilidade

- Um modo de garantir a confiabilidade do sistema de arquivos é a de garantir a **consistência** deste sistema:
  - Para melhorar a eficiência, muitos sistemas somente gravam os blocos modificados mais tarde.
  - Se o sistema travar antes de blocos importantes serem salvos, o sistema de arquivos pode ficar inconsistente:
    - Exemplos de blocos importantes são os com os nós-i, com os diretórios, ou com a lista de blocos livres do disco.
  - A consistência é verificada através de um programa que avalia todos os blocos e diretórios do disco.
  - A verificação de consistência é feita, em geral, para cada um dos sistemas de arquivo no computador.
  - Também podemos definir regras que protejam o sistema de erros cometidos pelos usuários:

Para evitar que estes errros ocorram, alguns sistemas, como o Windows, copiam o arquivo para a Lixeira, um diretório especial. O arquivo somente será removido se a lixeira for limpa. O MS-DOS, ao remover um arquivo, marca como inválida a sua entrada no diretório, e somente o remove quando esta entrada é necessária.

Confiabilidade

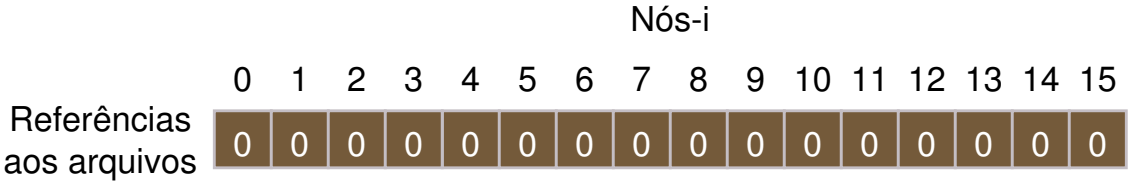
- No UNIX, as duas verificações são feitas do seguinte modo:
  - Na verificação dos blocos, duas tabelas são usadas: a tabela com os blocos em utilização, e a tabela com os blocos livres:



A/E

O algoritmo que verifica os blocos usa duas tabelas, com uma entrada para cada bloco do disco: a tabela *blocos em utilização*, com os blocos que pertencem a algum arquivo, e a tabela *blocos livres*, com os blocos que estão na lista de blocos livres. Inicialmente, as entradas são todas iguais a 0.

  - Na verificação dos diretórios, usamos uma tabela de contadores com o número de ocorrências de cada um dos arquivos:

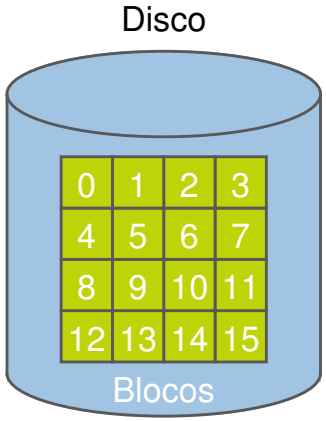


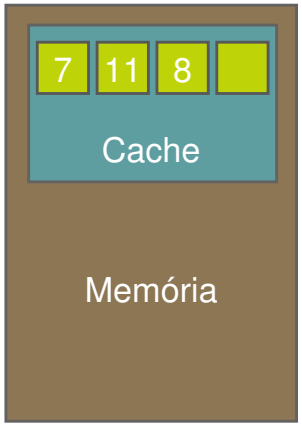
O algoritmo que verifica a consistência dos arquivos usa somente uma tabela, com uma entrada para cada nó-i: o número de vezes que este nó-i aparece em um diretório do sistema de arquivos. Inicialmente, todas as entradas são iguais a 0.

A/D

Desempenho

- O acesso ao disco é muito mais lento do que a memória, e com isso, devemos reduzir ao máximo os acessos ao disco.
- Um modo de reduzirmos os acessos ao disco é o de usarmos uma **cache de blocos** (ou **buffer**) na memória:
  - A cache, que possui blocos logicamente associados ao disco, procura armazenar os blocos mais referenciados do disco.
  - Um bloco somente será lido do disco se este não estiver na cache, pois neste caso, o bloco será lido da cache.

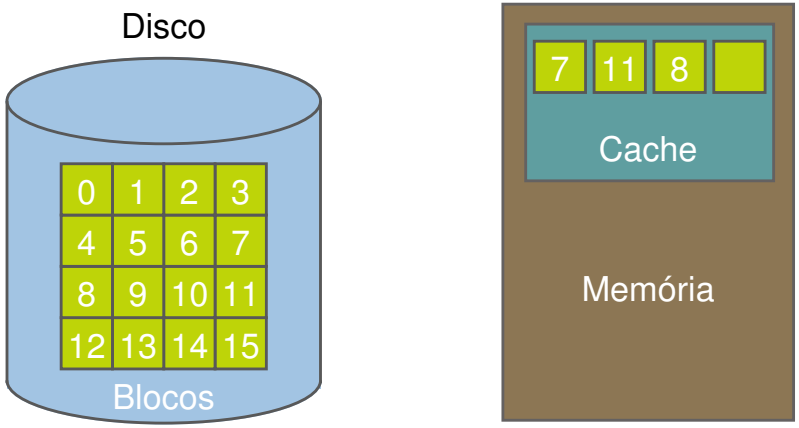




Exemplo de um sistema que possui um disco com 16 blocos, numerados de 0 à 15, e uma cache na memória que pode armazenar até 4 blocos do disco. Ao ler um bloco, a cache sempre será verificada antes de iniciarmos uma operação de leitura no disco.

Desempenho

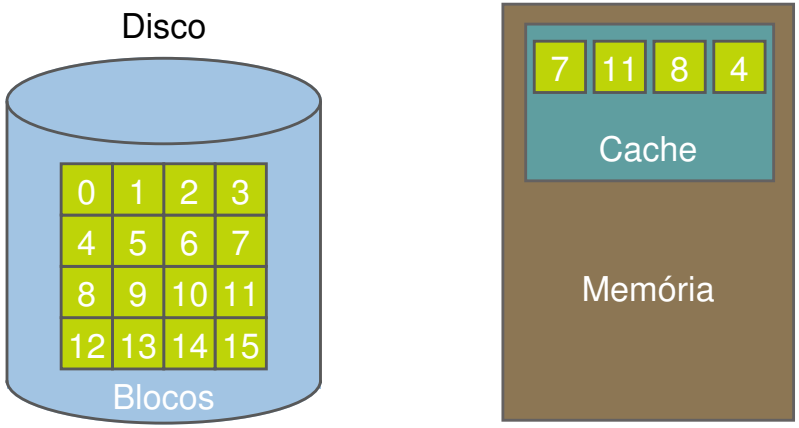
- O acesso ao disco é muito mais lento do que a memória, e com isso, devemos reduzir ao máximo os acessos ao disco.
- Um modo de reduzirmos os acessos ao disco é o de usarmos uma **cache de blocos** (ou **buffer**) na memória:
  - A cache, que possui blocos logicamente associados ao disco, procura armazenar os blocos mais referenciados do disco.
  - Um bloco somente será lido do disco se este não estiver na cache, pois neste caso, o bloco será lido da cache.



Se o sistema precisa acessar, por exemplo, o bloco 11, não será necessário ler este bloco do disco, pois este está na cache, e, como sabemos, o acesso à memória é bem mais rápido do que o acesso ao disco.

Desempenho

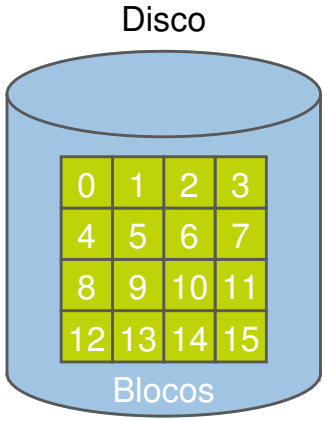
- O acesso ao disco é muito mais lento do que a memória, e com isso, devemos reduzir ao máximo os acessos ao disco.
- Um modo de reduzirmos os acessos ao disco é o de usarmos uma **cache de blocos** (ou **buffer**) na memória:
  - A cache, que possui blocos logicamente associados ao disco, procura armazenar os blocos mais referenciados do disco.
  - Um bloco somente será lido do disco se este não estiver na cache, pois neste caso, o bloco será lido da cache.

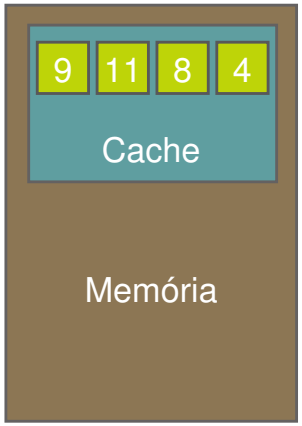


Se o sistema acessar agora o bloco 4, então o sistema deverá ler o bloco do disco, pois este não está na cache. Para que futuras referências ao bloco 4 sejam satisfeitas pela cache, este será copiado na entrada livre da cache.

Desempenho

- O acesso ao disco é muito mais lento do que a memória, e com isso, devemos reduzir ao máximo os acessos ao disco.
- Um modo de reduzirmos os acessos ao disco é o de usarmos uma **cache de blocos** (ou **buffer**) na memória:
  - A cache, que possui blocos logicamente associados ao disco, procura armazenar os blocos mais referenciados do disco.
  - Um bloco somente será lido do disco se este não estiver na cache, pois neste caso, o bloco será lido da cache.

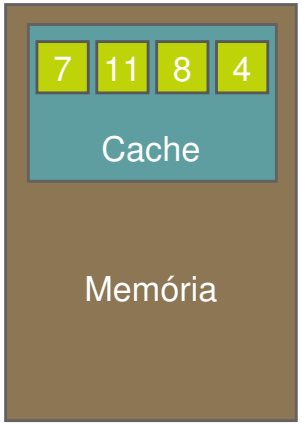




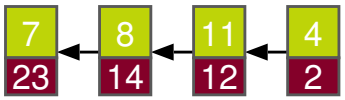
Se agora o sistema tentar ler outro bloco do disco que não está na cache, como o 9, deveremos ler, assim como antes, o bloco do disco. Mas como a cache agora está cheia, deveremos remover um dos blocos da cache, e salvá-lo novamente no disco, se necessário. Podemos usar qualquer um dos algoritmos de substituição de página que vimos para escolher o bloco a ser removido. No exemplo, o bloco 7 foi removido da cache.

Desempenho

Para garantir a consistência do sistema de arquivos, podemos usar a seguinte versão do algoritmo LRU:



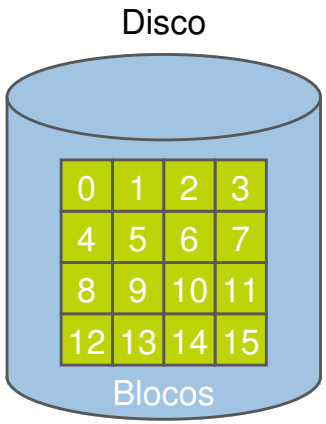
Lista LRU (blocos da cache)

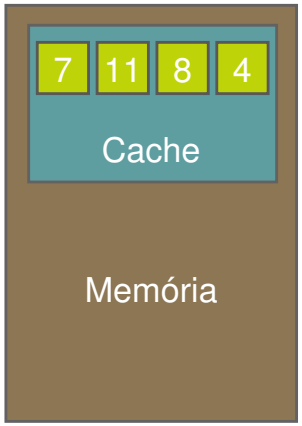


Como a cache em disco não é muito referenciada, podemos usar um algoritmo LRU exato, porque manter a lista de uso dos blocos será viável. O problema é que podemos ter inconsistências, pois um bloco com um nó-i ou uma lista de livres pode ser alterado, e ser salvo no disco somente após um longo período de tempo.

A/D

Mesmo assim, o sistema não pode manter os blocos modificados na cache por muito tempo, para evitar a perda de dados do usuário:



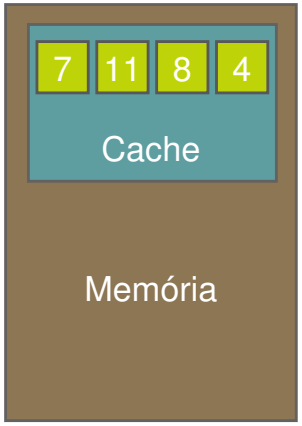


Em relação aos blocos alterados que estejam na cache do disco, podemos salvar imediatamente o bloco no disco assim que este for alterado, ou salvar o bloco somente quando este for retirado da cache. Vamos a seguir ver brevemente cada uma das abordagens.

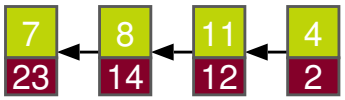
A/C

Desempenho

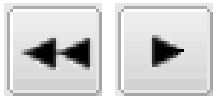
Para garantir a consistência do sistema de arquivos, podemos usar a seguinte versão do algoritmo LRU:



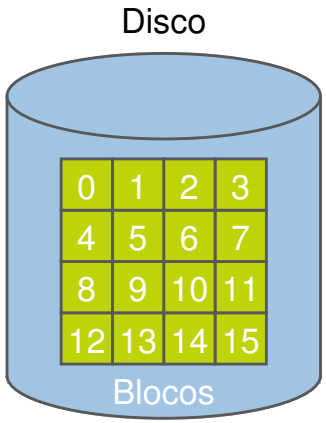
Lista LRU (blocos da cache)

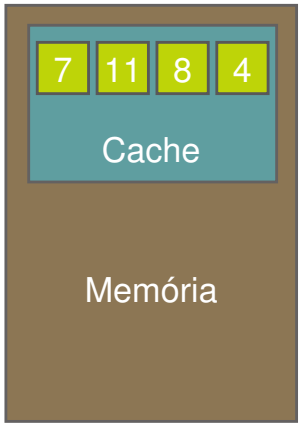


Além das inconsistências, que seriam geradas se, por exemplo, o bloco 4 fosse crítico à consistência do sistema, e fosse alterado sem ser salvo. O algoritmo também pode ser ineficaz, se, por exemplo, o bloco 7 estiver parcialmente usado, e o bloco 11 não for muito referenciado, como um bloco indireto duplo.



Mesmo assim, o sistema não pode manter os blocos modificados na cache por muito tempo, para evitar a perda de dados do usuário:

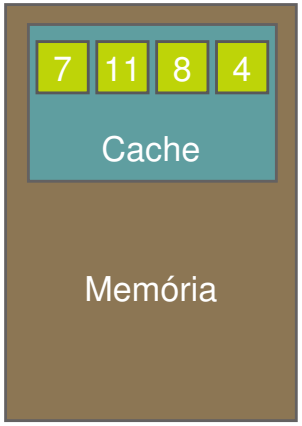




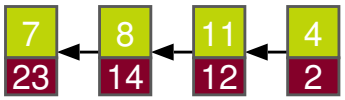
Em relação aos blocos alterados que estejam na cache do disco, podemos salvar imediatamente o bloco no disco assim que este for alterado, ou salvar o bloco somente quando este for retirado da cache. Vamos a seguir ver brevemente cada uma das abordagens.

Desempenho

Para garantir a consistência do sistema de arquivos, podemos usar a seguinte versão do algoritmo LRU:



Lista LRU (blocos da cache)

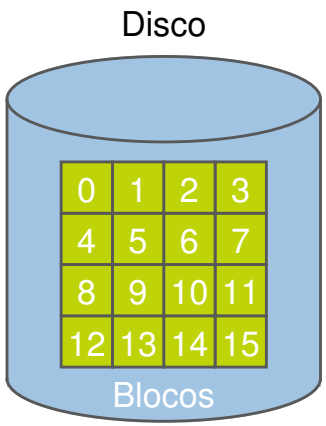


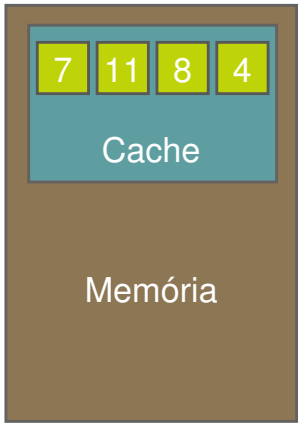
Para corrigir o algoritmo LRU, podemos levar em consideração dois fatores sobre cada bloco, o que nos permite construir um algoritmo LRU adequado:

1 - O bloco será usado novamente em breve?
2 - O bloco é essencial à consistência do sistema de arquivos?

C/D

Mesmo assim, o sistema não pode manter os blocos modificados na cache por muito tempo, para evitar a perda de dados do usuário:



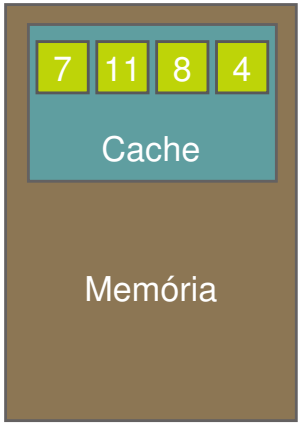


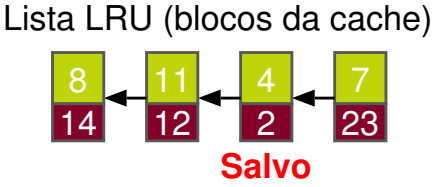
Em relação aos blocos alterados que estejam na cache do disco, podemos salvar imediatamente o bloco no disco assim que este for alterado, ou salvar o bloco somente quando este for retirado da cache. Vamos a seguir ver brevemente cada uma das abordagens.

A/C

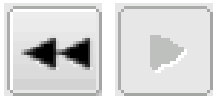
Desempenho

Para garantir a consistência do sistema de arquivos, podemos usar a seguinte versão do algoritmo LRU:



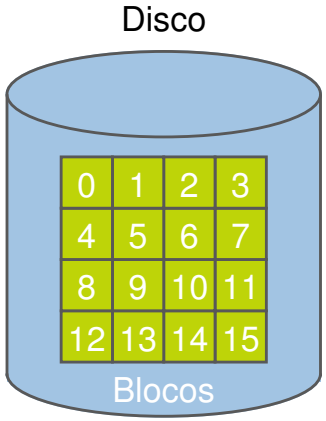


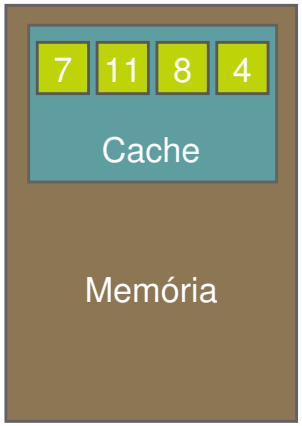
Usando estes fatores, podemos colocar os blocos que serão provavelmente usados no final da lista, os que não serão usados no início, e deveremos sempre salvar os blocos críticos, não importando a sua posição na lista. No exemplo anterior, o bloco 7 deveria ir para o final da lista, o 11 deveria ficar próximo ao início da lista, e o 4 deveria ser salvo.



D/D

Mesmo assim, o sistema não pode manter os blocos modificados na cache por muito tempo, para evitar a perda de dados do usuário:



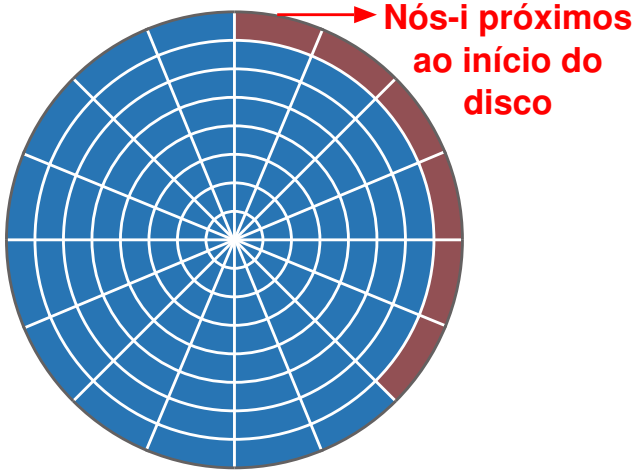


Em relação aos blocos alterados que estejam na cache do disco, podemos salvar imediatamente o bloco no disco assim que este for alterado, ou salvar o bloco somente quando este for retirado da cache. Vamos a seguir ver brevemente cada uma das abordagens.

A/C

Desempenho

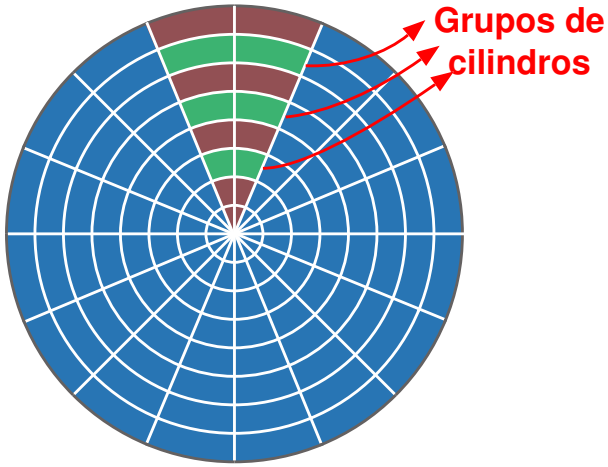
- Um outro modo de melhorar o desempenho do sistema é o de reduzir o deslocamento do braço mecânico do disco:
  - Cada bloco do arquivo, ao ser alocado, é colocado o mais próximo possível do bloco anterior do arquivo.
  - Para melhorar a eficiência, poderemos usar, no caso da lista encadeada de blocos livres, dois ou mais blocos consecutivos.
- Se levarmos em conta o posicionamento rotacional do disco, devemos alocar os blocos de forma intercalada no disco.
- Nos sistemas que usam estruturas similares aos nós-i, podemos fazer o seguinte, para melhorar o desempenho:



Num sistema que usa estruturas como os nós-i, deveremos fazer pelo menos dois acessos ao disco ao ler um arquivo: um para ler o nó-i, e outro para ler o bloco com os dados. Como, em geral, os nós-i estão no início do disco, a distância média entre este nó-i e este bloco será a metade do número de cilindros do disco, o que fará com que o acesso ao arquivo seja lento. Colocar os nós-i no meio do disco somente reduzirá o tempo de acesso pela metade.

Desempenho

- Um outro modo de melhorar o desempenho do sistema é o de reduzir o deslocamento do braço mecânico do disco:
  - Cada bloco do arquivo, ao ser alocado, é colocado o mais próximo possível do bloco anterior do arquivo.
  - Para melhorar a eficiência, poderemos usar, no caso da lista encadeada de blocos livres, dois ou mais blocos consecutivos.
- Se levarmos em conta o posicionamento rotacional do disco, devemos alocar os blocos de forma intercalada no disco.
- Nos sistemas que usam estruturas similares aos nós-i, podemos fazer o seguinte, para melhorar o desempenho:



Para melhorar o acesso ao arquivo, podemos dividir o disco em grupos de cilindros, cada um possuindo um conjunto diferente de nós-i, de blocos com dados, e lista de blocos livres. Quando um arquivo é criado, usamos qualquer nó-i de qualquer grupo, mas, quando alocamos um novo bloco para um arquivo, sempre escolhemos um bloco no grupo ao qual pertence o nó-i associado ao arquivo, ou no grupo o mais próximo possível deste grupo.

Segurança

- O sistema de arquivos deve fornecer os meios para proteger as informações dos usuários que este armazena:
  - Devemos garantir a integridade das informações dos usuários.
  - Devemos impedir o uso não autorizado das informações de um usuário por um intruso do sistema.
- A **segurança** estuda o problema da proteção das informações, e trata tanto da perda como do acesso a estas informações.
- A perda das informações pode ocorrer devido aos seguintes fatos, e pode ser tratada através do uso de backups:
  - **Ações *divinas***: fatos que independem do hardware, do software e do usuário, como incêndios, inundações, etc.
  - **Erros de hardware ou de software**: fatos que dependem de falhas do hardware ou de erros no software.
  - **Erros de humanos**: fatos que dependem de erros cometidos pelos usuários, como a perda de um disquete.

Segurança

- O problema do uso não autorizado das informações por intrusos é mais interessante, pois podemos ter dois tipos de intrusos:
  - Os **intrusos passivos** somente tentam ler as informações.
  - Os **intrusos ativos** causam danos reais, pois tentam alterar ou destruir as informações.
- Tornar o sistema seguro contra intrusos depende do tipo de intruso que tenta acessar o sistema:
  - **Bisbilhotice casual por usuários não-técnicos**: tentativa de acesso a arquivos e *emails* num sistema sem proteção.
  - **Espionagem por pessoas de dentro**: tentativa de acesso não autorizada pelos próprios usuários do sistema.
  - **Tentativa determinada de fazer dinheiro**: obtenção de lucro com o uso de contas inativas e alterações nos softwares.
  - **Espionagem comercial ou militar**: tentativa de roubo, por um intruso externo, de informações importantes e sigilosas.

Segurança

- Um outro aspecto é o da **privacidade**, isto é, proteger os usuários do uso abusivo das suas informações armazenadas no sistema.
- Para ter um sistema de arquivos seguro, deveremos ser capazes de combater ou evitar os seguintes ataques:
  - Tentativa de obter informações através da solicitação de páginas de memória, espaço em disco, ou espaço na fita.
  - Tentativa de executar chamadas ao sistema ilegais, ou então chamadas legais com parâmetros incorretos ou incomuns.
  - Tentativa de manipular o acesso ao sistema ao logar-se.
  - Tentativa de manipular as estruturas do sistema operacional armazenadas no espaço do usuário.
  - Tentativa de enganar os usuários que usam o sistema.
  - Tentativa de usar possíveis falhas documentadas do sistema.
  - Tentativa de manipular um programador do sistema.
  - Tentativa de obter acesso ao sistema através de suborno.

Segurança

- Um problema bem conhecido de segurança é o ataque ao sistema de arquivos feito por um **vírus** de computador:
  - Um **vírus** é um fragmento de programa que se une a algum programa legítimo com a intenção de infectar outros programas:

Programa infectado

```
1: jmp 1000
2: mov ax, 0
   ⋮
1000: mov bx,0
   ⋮
1499: mov cx,0
1500: jmp 2
```

Programa não-infectado

```
1: call 500
2: inc cx
   ⋮
```

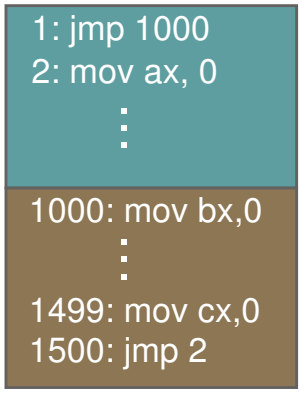
Quando um usuário executa um arquivo infectado, este não notará que o vírus está sendo executado, pois este altera a primeira instrução do arquivo para saltar para o código do vírus, e, ao final do seu código, o vírus executa a primeira instrução original, e depois salta para a segunda instrução. Diferentemente do vírus, um **verme** é um programa completo, que deve ser executado pelo usuário.

  - A procura por infecções de vírus e vermes no sistema pode ser feita através do uso de programas anti-vírus.
  - Podemos também manter um registro, checado periodicamente, com as somas de checagem de todos os arquivos do sistema.
  - A prevenção pode ser feita instalando-se no sistema somente os programas de fontes confiáveis.

Segurança

- Um problema bem conhecido de segurança é o ataque ao sistema de arquivos feito por um **vírus** de computador:
  - Um **vírus** é um fragmento de programa que se une a algum programa legítimo com a intenção de infectar outros programas:

Programa infectado



Programa infectado

Ao ser executado, o código do vírus procurará por outros arquivos executáveis ainda não infectados, e os infectará, copiando o código do vírus ao final do código do arquivo, e alterando a primeira instrução para saltar para o código do vírus, que executará, ao seu final, a primeira instrução do código original, e depois saltará para a segunda instrução deste código.

  - A procura por infecções de vírus e vermes no sistema pode ser feita através do uso de programas anti-vírus.
  - Podemos também manter um registro, checado periodicamente, com as somas de checagem de todos os arquivos do sistema.
  - A prevenção pode ser feita instalando-se no sistema somente os programas de fontes confiáveis.

Segurança

- Um desenvolvedor deve levar em conta os seguintes fatores ao projetar um sistema seguro:
  - O projeto do sistema deve ser de domínio público.
  - Por padrão, não deve ser permitido acessar o sistema.
  - Sempre verificar a autoridade ao permitir o acesso.
  - Ao executar um novo processo, o sistema deve sempre dar o menor privilégio de acesso possível a este processo.
  - O sistema deve possuir um mecanismo de proteção simples e uniforme, implementado nas camadas mais baixas do sistema.
  - O esquema de proteção deve ser psicologicamente aceitável, ou seja, não deve causar desconforto aos usuários.
- Um outro problema é o da **autenticação dos usuários**, em que o sistema verifica se o usuário é realmente quem diz ser:
  - Autenticação por senhas.
  - Autenticação por identificação física.
  - Uso de contramedidas.

Segurança

- Na **autenticação por senhas**, o usuário deve informar uma senha ao acessar o sistema:
  - Esta autenticação é fácil de entender e de implementar.
  - O problema é que a autenticação é facilmente quebrada, pois os usuários, em geral, usam senhas triviais.
  - O sistema pode melhorar a confiabilidade do seguinte modo:
    - Encorajar ou obrigar o usuário a criar senhas melhores, ou então exigir que a senha seja alterada regularmente.
    - Não mostrar e nem armazenar as senhas dos usuários de modo não criptografado no sistema.
  - Um exemplo é o da autenticação no sistema UNIX:

Arquivo de senhas criptografadas

```
ast:xDE3X4
joao:cEd34
bia: 2XedC
   ⋮
```

No UNIX, as senhas são criptografadas e armazenadas em um arquivo conjuntamente com os nomes dos usuários. Quando o usuário acessa o sistema, um programa de login pede o nome do usuário e a sua senha, criptografa esta senha, e permite o acesso somente se esta senha é igual a dada no arquivo. O problema é que um intruso pode criptografar as senhas mais comuns, e tentar obter acesso ao sistema.

Segurança

- Na **autenticação por senhas**, o usuário deve informar uma senha ao acessar o sistema:
  - Esta autenticação é fácil de entender e de implementar.
  - O problema é que a autenticação é facilmente quebrada, pois os usuários, em geral, usam senhas triviais.
  - O sistema pode melhorar a confiabilidade do seguinte modo:
    - Encorajar ou obrigar o usuário a criar senhas melhores, ou então exigir que a senha seja alterada regularmente.
    - Não mostrar e nem armazenar as senhas dos usuários de modo não criptografado no sistema.
  - Um exemplo é o da autenticação no sistema UNIX:

Arquivo de senhas criptografadas

```
ast:xDE3X4:23
joao:cEd34:2
bia: 2XedC:77
⋮
```

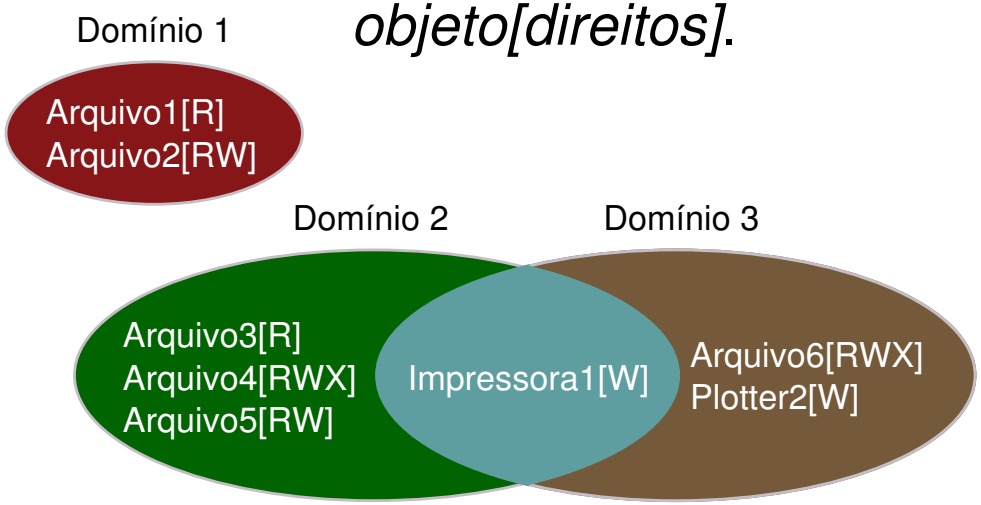
Para dificultar o ataque, o UNIX associa, a cada senha, um número aleatório que é alterado sempre que a senha é alterada, e que é salvo no arquivo de senhas sem ser criptografado. A senha é concatenada a este número antes de ser criptografada e salva no arquivo de senhas, e, ao verificar uma senha, o sistema a concatena com o número dado no arquivo e a compara com o que foi criptografado no arquivo.

Segurança

- Uma autenticação alternativa é a de fazer perguntas particulares aos usuários, ou então propor desafios conhecidos pelos usuários.
- A **autenticação por identificação física** usa algum item ou alguma característica física do usuário ao autenticá-lo:
  - Um exemplo é o uso de um cartão magnético e de uma senha ao acessar uma conta de um banco.
  - Um outro exemplo seria o uso da impressão digital do usuário.
  - Seja qual for o método usado de autenticação, este não deve ser incômodo aos usuários que acessam o sistema.
- As **contramedidas** melhoram a segurança da autenticação:
  - O sistema pode restrigir o acesso dos usuários a certas horas do dia, ou a partir de lugares pré-estabelecidos.
  - O sistema deve monitorar todos os acessos, para informar o último acesso ao usuário, e reportar tentativas de invasão.
  - O sistema pode estabelecer armadilhas para tentar capturar os intrusos que tentam acessar o sistema.

Mecanismos de proteção

- Os **mecanismos de proteção** são técnicas usadas para proteger os recursos do sistema operacional, como os arquivos dos usuários.
- O **monitor de referências** pode ser usado para garantir que o acesso a um dado recurso é permitido.
- O acesso aos recursos usa o conceito de **domínios de proteção**:
  - O sistema possui vários **objetos**, que podem ser componentes do hardware ou do software.
  - Para cada objeto, deveremos definir o conjunto de possíveis operações ao acessar estes objetos, chamadas de **direitos**.
  - Um **domínio de proteção** é um conjunto de pares do tipo *objeto[direitos]*.



Exemplo com 3 domínios de proteção e 8 objetos: 6 arquivos, uma impressora, e um plotter. Os direitos são 'R' para ler o conteúdo do objeto, 'W' para gravar algo no objeto, e 'X' para executar o objeto. No exemplo, o objeto Arquivo1 pode ser lido somente no domínio 1, e podemos gravar (imprimir) na Impressora1 somente dentro dos domínios 2 ou 3.

Mecanismos de proteção

- Ao usarmos os domínios de proteção, cada processo deverá estar, em um dado intervalo de tempo, em um dos possíveis domínios.
- Um processo pode mudar o seu domínio dinamicamente durante a execução (as regras de comutação dependerão do sistema).
- Um exemplo do uso de domínios é o do sistema UNIX:
  - O domínio do processo é definido pelos identificadores UID e GID (identificadores do usuário e do grupo a que este pertence).
  - Para cada domínio (UID, GID) podemos determinar quais são os objetos que podem ser acessados, e os direitos de acesso.
- Um processo do UNIX sempre pertecente a um domínio, e:
  - Possui duas partes: a do **usuário** e a do **núcleo**, que possuem acesso a diferentes objetos do sistema.
  - Pode adquirir um novo UID e GID, isto é, comutar para um novo domínio, ao executar um programa com a chamada **exec**.

Mecanismos de proteção

- A monitoração de quais objetos pertencem a quais domínios, e de quais operações são permitidas, pode ser feita por uma matriz:
  - Os próprios domínios podem ser considerados objetos, e com isso, a matriz pode modelar também a comutação de domínios.
  - Dado o objeto e o domínio do processo, o sistema pode saber quais operações podem ser executadas sobre este objeto.

Domínio \ Objeto	A	B	C	D	E	F	G	H
0	R	RW						
1			R	**RWX**	RW		W	
2						RWX	W	W

Objetos

A: Arquivo1
B: Arquivo2
C: Arquivo3
D: Arquivo4
E: Arquivo5
F: Arquivo6
G: Impressora1
H: Plotter2

Matriz para o exemplo anterior com 3 domínios e 8 objetos. Cada entrada da matriz informa quais operações o domínio da linha desta entrada pode executar sobre o objeto da coluna da entrada. Se a entrada está vazia, o domínio não pode acessar o objeto. Por exemplo, o domínio 1 pode ler (R), gravar (W), ou executar (X) o Arquivo4 (o objeto D).

Mecanismos de proteção

- A monitoração de quais objetos pertencem a quais domínios, e de quais operações são permitidas, pode ser feita por uma matriz:
  - Os próprios domínios podem ser considerados objetos, e com isso, a matriz pode modelar também a comutação de domínios.
  - Dado o objeto e o domínio do processo, o sistema pode saber quais operações podem ser executadas sobre este objeto.

Domínio \ Objeto	A	B	C	D	E	F	G	H	I	J	K
0	R	RW								Enter	
1			R	RWX	RW		W				
2						RWX	W	W			

Objetos

A: Arquivo1
B: Arquivo2
C: Arquivo3
D: Arquivo4
E: Arquivo5
F: Arquivo6
G: Impressora1
H: Plotter2
I: Domínio0
J: Domínio1
K:Domínio2

Como os domínios podem ser considerados objetos, a matriz anterior pode ser estendida para indicar uma comutação de domínios. Neste caso, se na entrada existir a operação ***Enter***, isso indica que podemos sair do domínio da linha desta entrada e entrar no domínio da coluna desta entrada. Por exemplo, podemos sair do domínio 0 e entrar no domínio 1.

Mecanismos de proteção

- Como armazenar toda a matriz é custoso, podemos armazená-la por colunas, usando as **listas de controle de acesso** (**ACL**):
  - Somente as entradas não vazias das colunas são armazenadas.
  - Cada objeto possui uma lista com os domínios que podem acessá-los, e com as operações permitidas para cada domínio:

Objeto	Lista com os dominios e as operações
Arquivo0	(Jan, *, RWX)
Arquivo1	(Jan, sistema, RWX)
Arquivo2	(Jan, *, RW-), (Els, staff,RW-),(Maaike,*,RW-)
Arquivo3	(*, estudante,R--)
Arquivo4	(Jelle,*,---),(*, estudante, R--)

Exemplo das listas de controle de acesso para 5 objetos, sendo que cada entrada de uma lista é uma tripla **(uid, gid, op)**, onde **(uid, gid)** é um domínio similar ao do UNIX (**uid** é a **UID** do usuário e **gid** é a **GID** do grupo deste usuário), e **op** é as operações **RWX** (**R** - leitura, **W** - gravação, **X** - execução), sendo que um **'-'** na posição da operação indica que esta é negada ao domínio.

- As permissões de acesso dadas em uma ACL podem sempre ser alteradas pelo proprietário do objeto.

Mecanismos de proteção

- Como armazenar toda a matriz é custoso, podemos armazená-la por colunas, usando as **listas de controle de acesso** (**ACL**):
  - Somente as entradas não vazias das colunas são armazenadas.
  - Cada objeto possui uma lista com os domínios que podem acessá-los, e com as operações permitidas para cada domínio:

Objeto	Lista com os dominios e as operações
Arquivo0	(Jan, *, RWX)
Arquivo1	(Jan, sistema, RWX)
Arquivo2	(Jan, *, RW-), (Els, staff,RW-),(Maaike,*,RW-)
Arquivo3	(*, estudante,R--)
Arquivo4	(Jelle,*,---),(*, estudante, R--)

Como um exemplo, considere a única entrada da lista para o Arquivo0. Ela indica que somente o domínio **(Jan, *)** pode ler, gravar e executar este arquivo, sendo que um **'*'** em um dos campos do domínio indica que este pode assumir qualquer valor válido, isto é, um usuário Jan de qualquer grupo.

  - As permissões de acesso dadas em uma ACL podem sempre ser alteradas pelo proprietário do objeto.

Mecanismos de proteção

- Como armazenar toda a matriz é custoso, podemos armazená-la por colunas, usando as **listas de controle de acesso** (**ACL**):
  - Somente as entradas não vazias das colunas são armazenadas.
  - Cada objeto possui uma lista com os domínios que podem acessá-los, e com as operações permitidas para cada domínio:

Objeto	Lista com os dominios e as operações
Arquivo0	(Jan, *, RWX)
Arquivo1	(Jan, sistema, RWX)
Arquivo2	(Jan, *, RW-), (Els, staff,RW-),(Maaike,*,RW-)
Arquivo3	(*, estudante,R--)
Arquivo4	(Jelle,*,---),(*, estudante, R--)

Um exemplo interessante é a lista para o Arquivo4: a primeira entrada indica que o usuário **Jelle** não pode acessar o arquivo, e a segunda entrada diz que todos os usuários do grupo **estudante**, com exceção do **Jelle**, podem somente ler o arquivo. Devido a isso, podemos negar completamente o acesso de um domínio a um objeto, sem proibir os outros domínios de acessarem este objeto.

- As permissões de acesso dadas em uma ACL podem sempre ser alteradas pelo proprietário do objeto.

Mecanismos de proteção

- Como armazenar toda a matriz é custoso, podemos armazená-la por colunas, usando as **listas de controle de acesso** (**ACL**):
  - Somente as entradas não vazias das colunas são armazenadas.
  - Cada objeto possui uma lista com os domínios que podem acessá-los, e com as operações permitidas para cada domínio:

Arquivo	Atributos do arquivo
arvore.obj	rw-r-r
teste	rwxrx-r
boot.img	r-x------
oowrite	--x--x--x
estranho.txt	**---rwx---**

Os arquivos do UNIX usam 9 bits para implementar a lista de modo compacto: os três primeiros bits dão a permissão de acesso para o dono do arquivo, os três seguintes dão a permissão de acesso para o grupo do dono, e os três últimos dão a permissão de acesso aos outros usuários dos outros grupos. A lista do último arquivo é suspeita, pois dá total acesso a qualquer usuário de outro grupo.

  - As permissões de acesso dadas em uma ACL podem sempre ser alteradas pelo proprietário do objeto.

Mecanismos de proteção

- Um outro modo de armazenarmos a matriz é por linhas, em listas chamadas de **listas de capacitação (listas C)**:
  - Cada domínio possui uma lista com os objetos que este pode acessar, e as operações permitidas sobre cada objeto.
  - Assim como antes, somente armazenaremos as entradas não vazias das linhas, que são chamadas de **capacidades**.

Posição	Tipo	Direitos	Objeto
0	Arquivo	R--	Ponteiro para Arquivo3
1	Arquivo	RWX	Ponteiro para Arquivo4
2	Arquivo	RW	Ponteiro para Arquivo5
3	Ponteiro	-W-	Ponteiro para Impressora1

Exemplo da Lista C para o Domíno 2 que vimos anteriormente. Cada entrada da lista é uma capacidade e possui 4 campos: a posição da capacidade na lista, o tipo do objeto, os possíveis direitos de acesso (os mesmos que vimos anteriormente), e um ponteiro para o objeto associado à capacidade.

A/D

  - As listas C devem ser protegidas contra alterações por usuários não autorizados:

Memória entiquetada

0	062774
1	1212324
1	9781232
⋮	⋮
0	98237364541

Um possível modo de proteger a lista C é o projeto do hardware ser baseado na **memória etiquetada**: cada palavra da memória possui um bit adicional que indica se a palavra está ou não associada a uma lista C. O bit não é usado pelo processador, e somente pode ser alterado no modo supervisor pelo sistema operacional.

A/C

Mecanismos de proteção

- Um outro modo de armazenarmos a matriz é por linhas, em listas chamadas de **listas de capacitação (listas C)**:
  - Cada domínio possui uma lista com os objetos que este pode acessar, e as operações permitidas sobre cada objeto.
  - Assim como antes, somente armazenaremos as entradas não vazias das linhas, que são chamadas de **capacidades**.

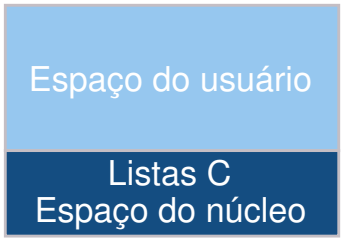
Posição	Tipo	Direitos	Objeto
0	Arquivo	R--	Ponteiro para Arquivo3
1	Arquivo	RWX	Ponteiro para Arquivo4
2	Arquivo	RW	Ponteiro para Arquivo5
3	Ponteiro	-W-	Ponteiro para Impressora1

Exemplo da Lista C para o Domíno 2 que vimos anteriormente. Cada entrada da lista é uma capacidade e possui 4 campos: a posição da capacidade na lista, o tipo do objeto, os possíveis direitos de acesso (os mesmos que vimos anteriormente), e um ponteiro para o objeto associado à capacidade.

A/D

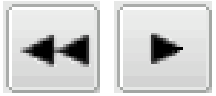
  - As listas C devem ser protegidas contra alterações por usuários não autorizados:

Memória



Um outro modo de proteger uma lista C é a de colocá-la no espaço do núcleo do sistema operacional, e definir que todos os acessos a um objeto sejam feitos a partir das posições das capacidades associadas a este objeto nas listas C.

B/C



Mecanismos de proteção

- Um outro modo de armazenarmos a matriz é por linhas, em listas chamadas de **listas de capacitação (listas C)**:
  - Cada domínio possui uma lista com os objetos que este pode acessar, e as operações permitidas sobre cada objeto.
  - Assim como antes, somente armazenaremos as entradas não vazias das linhas, que são chamadas de **capacidades**.

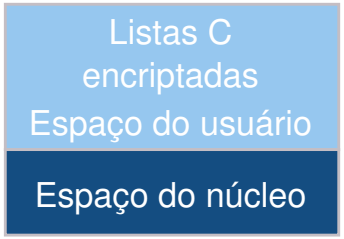
Posição	Tipo	Direitos	Objeto
0	Arquivo	R--	Ponteiro para Arquivo3
1	Arquivo	RWX	Ponteiro para Arquivo4
2	Arquivo	RW	Ponteiro para Arquivo5
3	Ponteiro	-W-	Ponteiro para Impressora1

A/D 

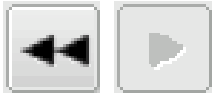
Exemplo da Lista C para o Domíno 2 que vimos anteriormente. Cada entrada da lista é uma capacidade e possui 4 campos: a posição da capacidade na lista, o tipo do objeto, os possíveis direitos de acesso (os mesmos que vimos anteriormente), e um ponteiro para o objeto associado à capacidade.

  - As listas C devem ser protegidas contra alterações por usuários não autorizados:

Memória

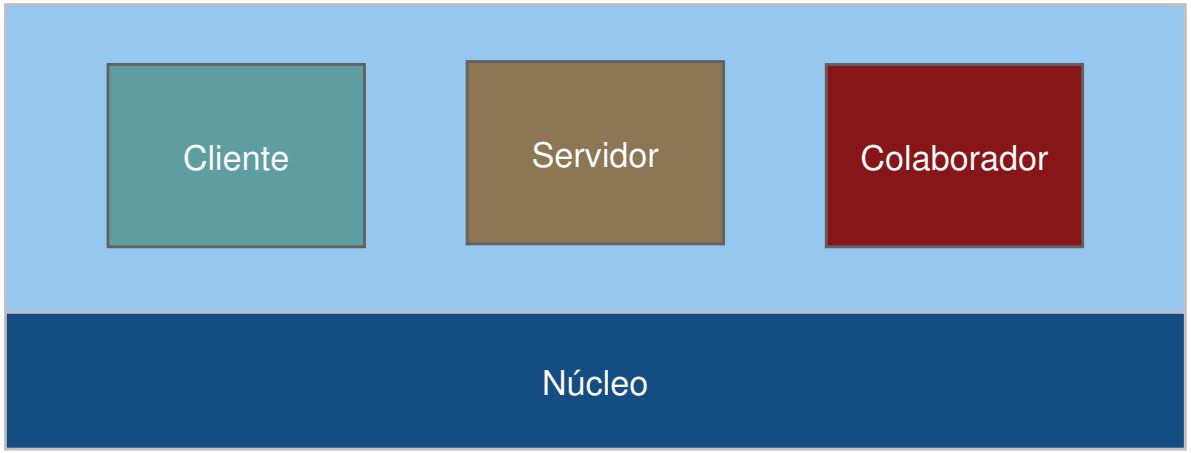


Podemos também colocar as listas C no modo usuário, mas encriptar cada capacidade com uma chave secreta que somente o sistema operacional conhece, o que impedirá um programa do usuário de alterá-la. Esta abordagem é usada pelos sistemas distribuídos.

C/C 

Mecanismos de proteção

- Mesmo se usarmos um dos mecanismos de proteção que vimos, falhas poderão ocorrer, como no **problema do confinamento**:
  - Neste problema temos 3 processos: um **cliente**, um **servidor**, e um **colaborador**.
  - O cliente requisita serviços ao servidor, que além de executar o serviço rouba dados do cliente com a ajuda do colaborador.

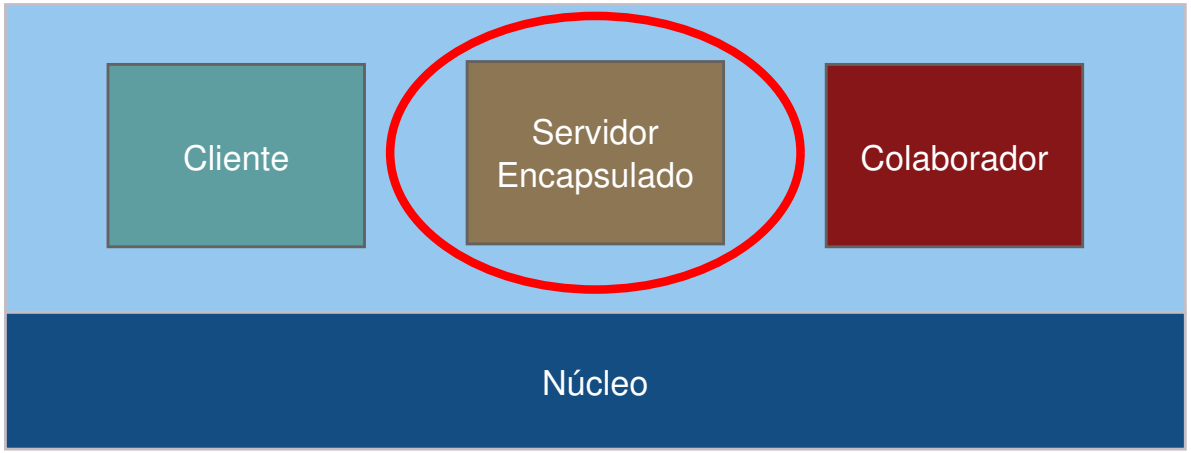


A/D

Se o sistema não impor nenhuma restrição de acesso ao processo servidor, este poderá se comunicar livremente com o colaborador e roubar os dados do usuário dono do processo cliente. O problema de impedir o processo servidor de passar os dados do processo cliente ao processo colaborador é chamado de o **problema do confinamento**.

Mecanismos de proteção

- Mesmo se usarmos um dos mecanismos de proteção que vimos, falhas poderão ocorrer, como no **problema do confinamento**:
  - Neste problema temos 3 processos: um **cliente**, um **servidor**, e um **colaborador**.
  - O cliente requisita serviços ao servidor, que além de executar o serviço rouba dados do cliente com a ajuda do colaborador.

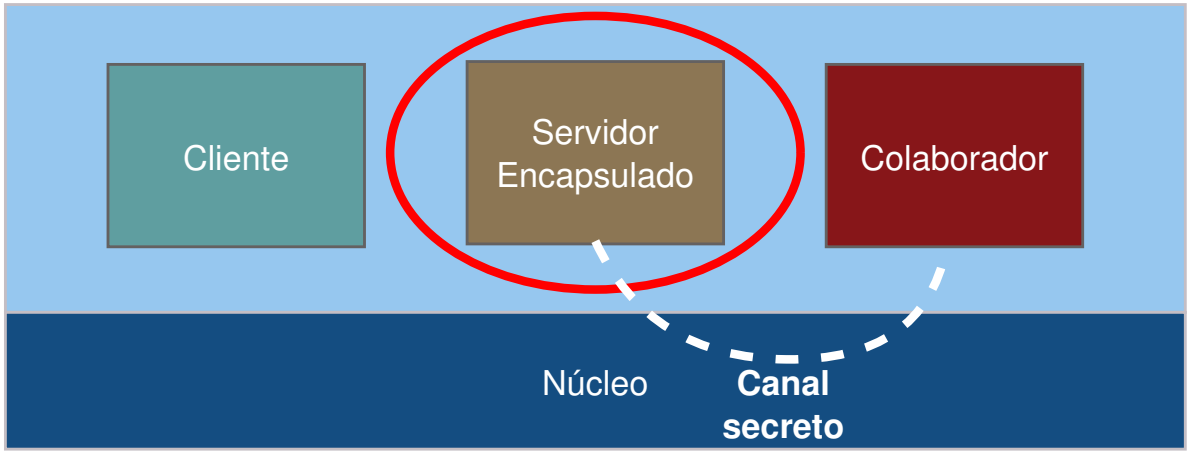


B/D

Encapsular o processo servidor, impedindo que este se comunique com o processo colaborador através do uso das primitivas de comunicação ou do compartilhamento de recursos (como um arquivo) não irá funcionar, pois os processos podem se comunicar usando os **canais secretos**.

Mecanismos de proteção

- Mesmo se usarmos um dos mecanismos de proteção que vimos, falhas poderão ocorrer, como no **problema do confinamento**:
  - Neste problema temos 3 processos: um **cliente**, um **servidor**, e um **colaborador**.
  - O cliente requisita serviços ao servidor, que além de executar o serviço rouba dados do cliente com a ajuda do colaborador.

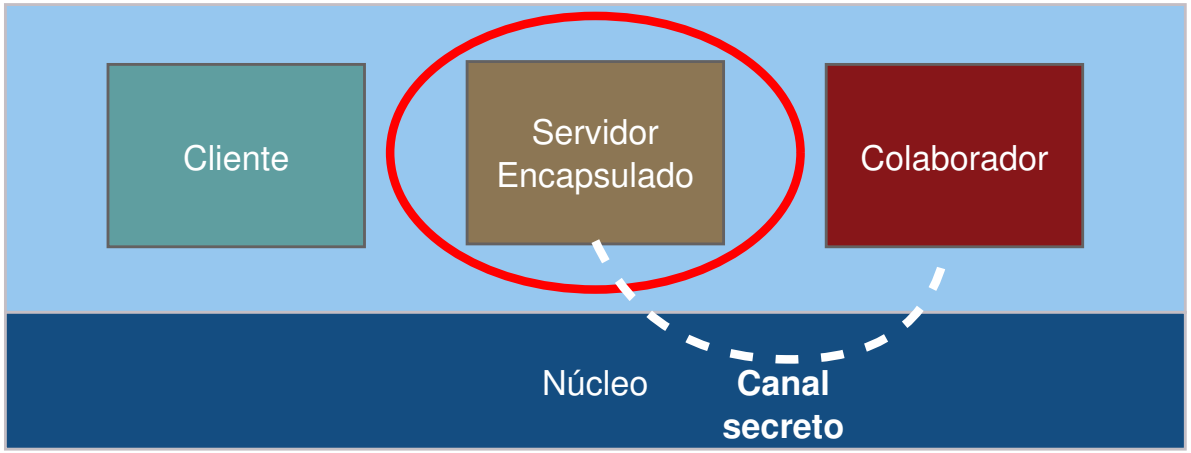


C/D

Um **canal secreto** é usado por dois processos para a troca de dados, bit a bit, através da degradação do sistema, de modo temporizado. Por exemplo, o processo servidor poderia ficar ocioso para enviar um bit 0, e executar muito trabalho para enviar um bit 1, e o processo colaborador poderia descobrir o bit monitorando o tempo de resposta do servidor a uma requisição de serviço.

Mecanismos de proteção

- Mesmo se usarmos um dos mecanismos de proteção que vimos, falhas poderão ocorrer, como no **problema do confinamento**:
  - Neste problema temos 3 processos: um **cliente**, um **servidor**, e um **colaborador**.
  - O cliente requisita serviços ao servidor, que além de executar o serviço rouba dados do cliente com a ajuda do colaborador.



D/D

Outros modos de transmissão de bits poderiam ser o monitoramento da taxa de falhas de página do servidor, o bloqueio e a liberação de arquivos ou de recursos, etc. O processo servidor também pode vazar informações ao seu proprietário. A detecção destes canais é difícil, e a redução da banda destes canais através da degradação do sistema não é aceitável.