Questões Playlist SO

Aula 1

- Um programa precisa saber se outros estão utilizando o mesmo recurso (ex. memória, cpu...)? (F)
- Um programa precisa saber em que tipo de rede (wifi, ethernet...) o computador está conectado para enviar pacotes? (F)
- O sistema operacional determina a ordem de execução dos programas? (V)
- O sistema operacional pode permitir o acesso de todas as funções para qualquer aplicação? (F)
- O sistema operacional também é um programa? (V)
- Todos os sistemas operacionais oferecem as mesmas funcionalidades? (F)

Aula 2

- A busca, decodificação e execução das instruções é igual em todas as CPUs? (F)
- Os programas em execução (processos) são armazenados na RAM? (V)
- O sistema operacional gerencia o ciclo de instruções (determina qual instrução vai ser buscada/executada)? (V)
- O sistema operacional pode interromper o ciclo de execução? (V)
- As instruções são executadas sempre de forma sequencial? (F)
- Ao fim do programa, busca-se a próxima instrução na RAM? (F)
- Ao fim do programa, o sistema operacional coloca outro programa na RAM? (V)
- O Stack Pointer aponta para a próxima instrução na pilha de execução? (F)
- O Program Counter aponta para a próxima instrução a ser executada? (V)
- O Instruction Register armazena a instrução que será decodificada e executada? (V)
- O PSW (Process Status Word) armazena todo o conteúdo dos registradores das CPUs? (F)
- O PSW (Process Status Word) permite a retomada da execução após um programa ser interrompido? (V)
- Pode haver mais de um programa na RAM? (V)

Aula 3

- Troca de contexto ocorre quando um processo é substituído por outro? (V)
- A virtualização da CPU é implementada por meio de mecanismos e políticas dependentes entre si? (F)
- Espaço de endereçamento é a região de memória visível para cada processo? (V)
- A virtualização da memória é um mecanismo que permite o isolamento entre processos? (V)
- Basicamente existem 3 estados que um processo assume: executando, bloqueado e em espera. (V)
- A fila de espera é sempre uma FIFO (First In First Out)? (F)
- O kernel do sistema operacional pode ser escrito em qualquer linguagem de programação? (V)
- O uso de linguagens orientadas a eventos é uma escolha natural para design de sistemas operacionais? (V)
- O PCB (Process Control Block) é uma estrutura de dados que o sistema operacional deve manter para todos os processos, incluindo ele mesmo. (V)
- O PCB precisa ser atualizado sempre que uma transição de estados ocorre? (V)
- Um processo pode modificar o PCB de outro? (F)
- O PCB, além dos registradores de uso geral, inclui o Process Status Word?(V)

- A diferença entre o PCB e o PSW é que o primeiro armazena, além do PSW, valor dos demais registradores. (V)
- Suponha que um processo começa a efetuar uma operação de I/O e o sistema operacional coloca ele no estado de waiting. Somente quando um processo volta para o estado de running é que ele termina a operação? (F)
- O PCB fica armazenado no espaço de endereçamento do respectivo processo. (F)

Aula 4

- No Unix, o terminal é o pai de todos os processos. (F)
- Suponha que um processo cria outro no Unix. Eles possuem o mesmo espaço de endereçamento e espaço de endereçamento virtual? (F)
- O código abaixo só efetua exit(1) quando está no processo pai. (F)

```
int *pids = malloc(n*sizeof(int));
for(int i = 0; i<n; i++)
{
  if((pids[i] = fork()) < 0)
  {
    exit(1)
  }
}
```

- Ainda no código acima, caso não tenha nenhum erro, ao final teremos n+1 processos a serem executados. (V)

Aula 5

- As funções privilegiadas variam de acordo com a CPU? (V)
- Um processo pode ocupar um espaço além do seu limite da pilha? (F)
- O modo duplo de operação é um mecanismo que assegura o isolamento. (V)
- Uma das vantagens de ter mais de dois modos de operação é evitar o número de trocas de contexto. Dessa forma um processo pode executar mais instruções privilegiadas do que no modo usuário, porém nem todas como no modo kernel, mantendo assim o isolamento. (V)
- O interrupt vector table é uma tabela que gera diferentes tipos de interrupções. (F)
- O interrupt vector table pode ser modificado por qualquer programa. (F)
- Os processos para executar uma chamada de sistema colocam o ponteiro para a função do kernel na pilha. (F)

Aula 6

- Existem várias métricas utilizadas para medir a eficiência de uma política de escalonamento. Suponha o contexto em que as tarefas possuem duração distinta. Se um escalonador priorizar o throughput, sempre ele alcança também a justiça?(F)
- Existem várias métricas utilizadas para medir a eficiência de uma política de escalonamento. Suponha o contexto em que as tarefas possuem duração distinta. Se um escalonador priorizar o baixo overhead, também terá um alto throughput?(F)
- No Round Robin, quanto menor o quantum melhor.(F)
- No Round Robin, o quantum grande é sempre ruim?(F)

Aula 7

- O escalonamento da MultiLevel Feedback Queue, quando tem processos de tamanhos diferentes, possui um desempenho similar ao Shortest Time to Complete First. (V)
- Colocar o quantum baixo e igual em cada fila da MLFQ faz com que ela tenha resultado similar a Round Robin. (V)
- Caso não houvesse um boost de prioridade para as tarefas que já estão na fila de menor prioridade, poderia ocorrer starvation.(V)
- O boost de prioridade também resolve o problema da transição de processos CPU-bound para I/O bound. (V)
- Um dilema para setar o período S após o qual o boost de prioridade é dado se refere a: se muito grande, ocorre starvation das tarefas maiores, se pequeno então os programas I/O bound não usariam uma fatia justa da CPU.(V)
- Caso não fosse armazenado o tempo de execução de uma thread enquanto estivesse numa fila, o escalonador estaria sujeito a um ataque no qual um processo simula uma ação de I/O antes de ceder a CPU. Dessa forma o processo continuaria com prioridade na CPU enquanto outros sofrem starvation.(V)
- Suponha que a prioridade de uma tarefa seja dada pela equação $priority = 1/(Exec_{time} + \in)$, onde ∈ é uma constante muito pequena e $Exec_{time}$ o tempo que aquela tarefa já executou. Um escalonador baseado nessa equação consegue atender ao desempenho desejado pela MLFQ? (F)
- Suponha que a prioridade de uma tarefa seja dada pela equação $priority = (1 + \Delta)/(Exec_{time} + \in)$, onde ∈ é uma constante muito pequena, $Exec_{time}$ o tempo que aquela tarefa já executou e Δ o tempo decorrido desde a última execução da tarefa.Um escalonador baseado nessa equação consegue atender ao desempenho desejado pela MLFQ?(V)

Aula 8

- Suponha o caso em que todos os processos possuem a mesma prioridade e o quantum pode assumir qualquer valor. No CFS, um problema latente é o caso de haver muitos processos esperando execução, o que leva ao quantum possuir um valor baixo. Com isso teremos mais troca de contexto, prejudicando o desempenho do sistema. (V)
- O quantum, no CFS, pode assumir qualquer valor. (F)
- No Linux, a variável Niceness determina a prioridade de cada tarefa. Quanto maior esse valor, maior a prioridade. (F)
- O CFS mantém todos os processos na árvore rubro negra. (F)
- Suponha que a prioridade de uma tarefa seja dada pela equação $priority = (1 + \Delta)/(vruntime + \in)$, onde ∈ é uma constante muito pequena, $vruntime$o tempo que aquela tarefa já executou e Δ o tempo decorrido desde a última execução da tarefa. Imagine que alguns processos estão na lista de espera. Para que um escalonador tenha desempenho similar a CFS basta atualizar o vruntime deles com o menor valor que está na fila de ready? (F)
- A decisão de quando escalonar as tarefas ocorre de tempos em tempos conforme sched_latency. (V)
- A variável sched_latency pode ser acessada por qualquer programa?(F)
- Suponha que a prioridade de uma tarefa seja dada pela equação $priority = (1 + \Delta)/(vruntime + \in)$, onde ∈ é uma constante muito pequena,

$vruntime$o tempo que aquela tarefa já executou e Δ o tempo decorrido desde a última execução da tarefa. Imagine que alguns processos estão na lista de espera. Para que um escalonador tenha desempenho similar a CFS precisa atualizar o vruntime deles com o menor valor que está na fila de ready e calcular o quantum de cada processo como $quantum = (sched_{latency} x\ priority_i) / \Sigma_j priority_j$. (F)

- No CFS, a escolha de usar uma árvore rubro negra tem a ver não apenas com o fato de o rebalanceamento da árvore ser mais eficiente, mas também porque o tempo de busca de um nó é mais rápido. (V)

Aula 9

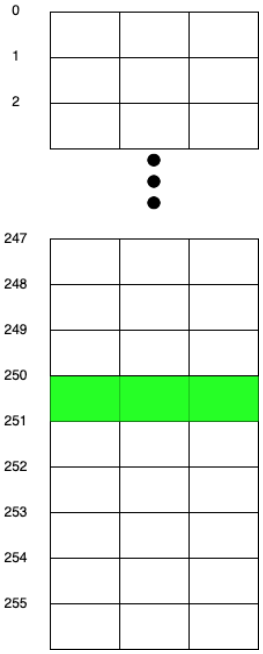
- A tradução do endereçamento virtual para o físico é feita por cada processo.(F)
- Cada processo escolhe seu endereço físico.(F)
- O método de base and bound supõe que os processos não são contíguos na memória.(F)
- O processo pode acessar qualquer endereço virtual?(F)
- A Memory Management Unit faz parte do sistema operacional?(F)
- O espaço de endereçamento pode aumentar ou diminuir dinamicamente com o base and bound?(V)
- Fragmentação interna é o espaço de endereçamento não utilizado pelo processo.(V)
- No base and bound o processo pode modificar qualquer endereço acessível da RAM.(V)
- No base and bound processos conseguem se comunicar utilizando arquivos no disco rígido?(V)

Aula 10

- Com a segmentação é possível ter o isolamento interno de espaços de endereçamento do mesmo processo.(V)
- Cada segmento é um espaço contíguo de memória.(V)
- Com a segmentação é impossível ter fragmentação interna?(F)
- A segmentação aumenta a quantidade de memória que cada PCB ocupa?(V)
- A segmentação permite a comunicação entre processos via compartilhamento de memória.(V)
- Cada processo possui a base e limite dos seus segmentos armazenados na pilha de usuário.(F)
- A fragmentação externa é a principal desvantagem da segmentação.(V)
- Os segmentos podem ter tamanhos variáveis.(V)

Aula 11

- A paginação resolve completamente o problema da fragmentação externa.(V)
- A paginação torna o carregamento do programa mais rápido?(V)
- Suponha que tenhamos 2ˆ8 endereços de memória, tal como a figura abaixo, cada qual com 8 bytes. Suponha que cada página é composta por 32 bytes. Queremos acessar o endereço 250. Sabendo que os 6 primeiros bits são utilizados para o virtual page number e os dois últimos é o offset, então o endereço precisa ser acessado pelo FAH.(V)



- O bit sujo evita o overhead de atualizar a memória sempre.(V)
- O copy on write é um mecanismo que permite a utilização da memória por mais de um processo. Quando um decide sobrescrever nessa região, todos passam a ver as mesmas modificações.(F)
- Uma desvantagem do tamanho de página ser grande é a fragmentação interna.(V)
- A segmentação torna a leitura de dados mais lenta.(V)

Aula 12

- A localidade temporal diz que um processo provavelmente vai acessar a mesma instrução ou dado no futuro.(V)
- A localidade espacial diz que um processo provavelmente vai acessar outra instrução ou dado no endereço próximo ao que foi acessado recentemente.(V)
- O Transfer Lookaside Buffer sempre melhora o desempenho do sistema.(F)
- O uso de páginas muito pequenas prejudicam o TLB.(V)
- O TLB é salvo no PCB.(F)

Aula 13

- Para lidar com um grande número de páginas, porém algumas sendo contíguas, os sistemas operacionais buscam agrupar as entradas no TLB numa única. Isso aumenta a probabilidade de hit. (V)
- O PTBR (Pointer Table Base Register) é o responsável por determinar a entrada na tabela de paginação. (V)
- O cache de páginas preenchidas melhora a eficiência do TLB. (V)
- A tabela de páginas é armazenada na RAM, e por isso também é dividida em páginas.(V)
- A tabela de paginação é linear, o que significa que toda a sua memória é contígua. (V)
- A principal vantagem da tabela de paginação ser linear é permitir o acesso direto a qualquer entrada dela.(V)
- A principal desvantagem da tabela de paginação ser linear é que as entradas que não utilizadas para armazenar os endereços físicos ocupam espaço à toa. (V)

- O uso do PDBR (Pointer Directory Base Register) para paginação em multinível sempre vai representar economia de espaço na memória?(F)
- O Diretório de Tabela de Páginas só vai possuir uma entrada válida quando a tabela de páginas para a qual aponta possui pelo menos uma entrada válida.(V)
- A paginação multinível torna mais custoso o TLB miss.(V)
- Na segmentação com paginação, cada segmento possui um único diretório de páginas. Um segmento nesse caso pode ser acessado por mais de um processo e um processo pode acessar mais de um segmento. Essa técnica reduz tanto a fragmentação interna quanto a externa.(V)
- Os processos possuem carregamento mais rápido na RAM quando utilizado a segmentação com paginação. (V)

Aula 14

- Duas políticas de escrita são possíveis, write-back e write-through. Essa última sempre será mais eficiente?(F)
- O uso de cache multinível torna mais custoso o cache miss.(V)
- A paginação sob demanda permite que programas que não cabem na RAM consigam ser executados.(V)
- O Swapping quebra o conceito de justiça, pois o processo que tiver seu conteúdo movido para o disco ficará mais lento em relação aos demais de forma indeterminada.(F)

Aula 15

- Cache miss compulsório ocorre quando o conteúdo da memória é acessado pela primeira vez. (V)
- Uma possível política de substituição em cache é retirar aquela entrada que está mais tempo sem ser acessada. Essa é sempre a melhor forma de atualizar a cache.(F)
- Sempre que tem um processo que segue a suposição de localidade espacial, por exemplo um loop com um conjunto de instruções, a política de substituição FIFO se comporta de forma similar a LRU e tem um péssimo desempenho independente do tamanho do cache. (F)
- A principal desvantagem da política de substituição LRU é o overhead de ter que atualizar a contagem.(V)
- A LRU explora a localidade temporal melhor que as demais políticas?(V)
- Aumentar o tamanho da cache sempre implica no aumento de hit? (F)

Aula 16

- As threads podem compartilhar a pilha de usuário?(F)
- Threads possuem o mesmo espaço de endereçamento?(V)
- Threads podem alterar os dados umas das outras?(F)
- A execução das threads segue uma lógica estritamente sequencial?(F)
- O uso de threads é uma estratégia para evitar o processo inteiro seja colocado no estado de waiting.(V)
- O uso de threads sempre deixam os programas mais rápidos?(F)
- Todo programa concorrente tem paralelismo?(F)
- A segmentação pode ser usada para impedir a interferência de outra thread?(F)
- Threads possuem endereços de memória virtual distintos?(F)

Aula 17

- Todo kernel suporta o uso de threads?(F)

- O escalonamento de threads é sempre feito pelo sistema operacional, independente do kernel. (F)
- No modelo de thread N:1, sempre que uma thread para de executar o sistema operacional assume a cpu para efetuar o próximo escalonamento. (F)
- O uso de threads sempre melhora o desempenho de programas I/O bound, independente do tipo de kernel?(F)
- No caso em que mais de uma thread é criada a função main pode terminar antes que as outras terminem de executar?(V)

Aula 18

- Todo código em assembly é executado sem interrupções?(F)
- As operações em assembly permitem manipular diretamente a memória?(F)
- Condição de corrida é quando o resultado do programa depende da ordem de execução.(V)
- Toda condição de corrida leva a um resultado incorreto?(F)
- Exclusão mútua é quando uma thread impede qualquer outra de executar ao mesmo tempo?(F)
- Exclusão mútua é quando uma thread impede outra com a mesma região crítica de executar ao mesmo tempo. (V)
- Região crítica é o trecho de código que gera condição de corrida. (V)
- Operação atômica é aquela que não pode ser interrompida antes de seu término.(V)

Aula 19

- Quando uma thread espera pelo lock, ela é automaticamente bloqueada.(V)
- O lock é armazenado na pilha de usuário específica de uma thread. (F)
- Lock só é necessário quando há paralelismo?(F)
- Considere o código abaixo:

```
#include <pthreads.h>
#include <stdio.h>
int lock = 0;
int result = 0;
void *add()
{
      while( lock == 1 );
      lock = 1;
      result = result + 1;
      lock = 0;
}

int main()
{
      pthread_t a, b;
      pthread_create(&a,NULL, add, NULL);
      pthread_create(&b,NULL, add, NULL);
      printf("%d",result);
      return 0;
}
```

Um resultado possível da função print é 0. (V)

Aula 20

- Desabilitar interrupções para evitar concorrência só é possível em arquiteturas de 1 CPU.(V)
- Considere o código abaixo:

```
#include <pthreads.h>
#include <stdio.h>
int lock = 0;
int result = 0;
void *add()
{
        while( lock == 1 );
        lock = 1;
        result = result + 1;
        lock = 0;
}

int main()
{
        pthread_t a, b;
        pthread_create(&a,NULL, add, NULL);
        pthread_create(&b,NULL, add, NULL);
        printf("%d",result);
        return 0;
}
```

  Os resultados possíveis da função print são 0, 1 e 2. (V)
- A principal desvantagem do spin lock é o gasto desnecessário de CPU.(V)
- Suponha que o escalonador tenha duas tarefas, A e B, para executar. Suponha que A e B compartilhem uma variável, tornando-a uma região crítica. Suponha ainda que prioridade de A > prioridade de B depois de algum tempo. Assim, se B pega o lock antes de A ganhar maior prioridade, então nenhuma das duas tarefas executará. (V)
- A thread que chama yield fica bloqueada. (F)
- Um dos principais problemas do yield é que pode haver muitas trocas de contexto enquanto não há nenhuma outra thread esperando.(F)
- A técnica de two phase lock permite que o desperdício de ciclos na CPU seja reduzido como também evita de bloquear a thread. (V)
- Suponha que numa CPU tenhamos duas instruções atômicas distintas, uma de test e outra de set. Considere também o lock implementado abaixo:

```
void init(int *mutex)
{
  *mutex = 0;
}

void lock(int *mutex)
{
  while(test(*mutex) == 1);
    set(*mutex);
}

void unlock(int *mutex)
{
  *mutex = 0;
}
```

  Haveria condição de corrida usando essa implementação?(V)

- Considere a implementação do lock abaixo.

```
void init(int *mutex)
{
  *mutex = 0;
}

void lock(int *mutex)
{
  while(TestAndSet(*mutex) == 1);
}

void unlock(int *mutex)
{
  *mutex = 0;
  yield();
}
```

  Ainda teríamos o problema da espera ocupada?(V)

Aula 21

- As variáveis de estado garantem que não haverá perda do sinal quando uma thread finalizar. (V)
- Imagine que existem 3 threads, A, B e C. Suponha que B, a thread secundária, esteja executando no momento, enquanto as demais esperam pelo sinal. Suponha que as threads não verifiquem a variável de estado. Agora B finaliza enviando sinal antes de A chamar o wait, porém depois de C efetuar a mesma chamada. Nesse caso há alguma possibilidade de A executar de fato?

Aula 22

- Considere o código abaixo, retirado do livro, no qual temos produtor e consumidor alterando o conteúdo de uma variável inteira:

```
int loops;
cond_t cond;
mutex_t mutex;
void*producer(void*arg)
{
      int i;
      for (i = 0; i < loops; i++)
      {
            Pthread_mutex_lock(&mutex); //
            if (count == 1)
                  Pthread_cond_wait(&cond, &mutex);
            put(i);
            Pthread_cond_signal(&cond);
            Pthread_mutex_unlock(&mutex);
      }
}

void*consumer(void*arg)
{
      int i;
      for (i = 0; i < loops; i++)
      {
```

```
                Pthread_mutex_lock(&mutex);
                if (count == 0)
                      Pthread_cond_wait(&cond, &mutex);
                int tmp = get();
                Pthread_cond_signal(&cond); //
                Pthread_mutex_unlock(&mutex);
                printf("%d\n", tmp);
          }
     }
```

Suponha que tenhamos duas threads consumidoras, A e B, e uma produtora. É possível a leitura da variável mesmo com count=0?(V)

- Ainda de acordo com o código acima, suponha que tenhamos duas threads produtoras e uma consumidora. É possível a escrita da variável mesmo com count=1?(V)
- Trocar apenas if por while resolve todos os conflitos no código acima?(F)

Aula 23

- Considere o código abaixo:

```
semaphore a, b, c;
semaphore_init(a,1);
semaphore_init(b,0);
semaphore_init(c,0);

void *p1()
{
   sem_wait(a);
   exec_something();
   sem_post(b);
}

void *p2()
{
   sem_wait(b);
   exec_something();
   sem_post(c);
}

void *p3()
{
   sem_wait(c);
   exec_something();
   sem_post(a);
}
```

Imagine 3 threads, cada uma executando um dos trechos acima. Com base nisso, é correto afirmar que a ordem de execução será p1 -> p2 -> p3 sempre? (V)

- Suponha agora que apenas o "exec_something()" seja trocado de lugar em todos trechos de código, sendo a última chamada de função em cada thread. Com base nisso, é correto afirmar que a ordem de execução será p1 -> p2 -> p3 sempre? (F)

- Uma variável compartilhada x, inicializada com 0, é operada em 4 processos concorrentes W, X, Y e Z conforme se segue. Suponha ainda um semáforo S. Cada processo X e W chama wait(S), lê da memória, adiciona 1, escreve de volta na memória, chama post(S) e finaliza. Por outro lado, os processos Y e Z efetuam as mesmas operações, exceto que ao invés de adicionar 1, subtrai 2. Dado que o semáforo foi inicializado com 2, o valor máximo que x pode atingir é -2? (F)

Aula 24

- Considere o código abaixo, onde "a", "b" e "c" são semáforos inicializados com 1:

```
void *p1()
{
   sem_wait(a);
   sem_wait(b);
   exec_something();
   sem_post(a);
   sem_post(b);
}
```

```
void *p2()
{
   sem_wait(b);
   sem_wait(c);
   exec_something();
   sem_post(b);
   sem_post(c);
}
```

```
void *p3()
{
   sem_wait(c);
   sem_wait(a);
   exec_something();
   sem_post(c);
   sem_post(a);
}
```

Pode ocorrer deadlock?(V)

- Ainda em relação ao código anterior, é possível que tenha uma condição de corrida?(F)
- Para solucionar o problema, basta mudar a ordem em que os semáforos recebem wait em uma única thread?(V)
- Algumas funções precisam adquirir mais de um lock. Suponha a seguinte função:

```
void *proc(lock_t a,lock_t b)
{
        acquire(a);
        acquire(b);
        exec_something();
        release(a);
        release(b);
}
```

Suponha que tenham agora dois locks, lck1 e lck2, tal como duas threads que executam o mesmo trecho acima. No entanto, suponha que a thread1 chamou proc(&lck1,&lck2), enquanto a thread2 chamou proc(&lck2,&lck1). Nesse caso, haverá deadlock?(V)

- Ainda de acordo com o código acima, bastaria ordenar o processo de adquirir os locks de acordo com o endereço de memória das variáveis solucionaria o problema?(V)

Aula 25

- Uma das vantagens do SSD em relação ao disco é o tempo menor de acesso. (V)
- Uma das desvantagens do SSD em relação ao disco é o custo por byte e a capacidade de armazenamento reduzida.(V)
- Alguns componentes do disco magnético são os pratos (platters), eixo (spindle) e cabeça (head). A cabeça é uma para cada superfície dos pratos, e ela está apoiada no braço que faz com que ela se aproxime e se afaste do eixo.(V)
- O menor tamanho de endereçamento no disco é dado pelo tamanho do setor. As operações em disco são atômicas e o conjunto de setores formando um círculo é denominado de trilha. Uma superfície de disco é composta por várias dessas trilhas. (V)
- O endereçamento no disco é feito por meio de logical block address que especifica um determinado setor no disco.(V)

- O tempo de acesso é determinado por três fatores: o tempo de busca, o tempo de rotação e o tempo de transferência. Primeiramente o disco se move até o setor desejado. Então depois disso começa imediatamente a escrita/leitura dos dados. (F)
- O track buffer (buffer de trilha) atua como um cache das operações em disco. Para evitar sucessivas movimentações no disco o track buffer armazena até uma trilha completa. (V)
- O track buffer atua evitando múltiplas movimentações da cabeça até a trilha/setor correto. Para isso ele se vale do princípio da localidade.(V)
- Os discos só movem em um único sentido.(V)

Aula 26

- Ao lidar com armazenamento persistente, o desenvolvedor precisa especificar o exato endereço de memória onde deseja salvar o conteúdo.(F)
- Nome e número de arquivo desempenham uma relação biunívoca sempre. (F)
- Caminho absoluto é aquele que especifica desde o diretório raiz até o arquivo.(V)
- Suponha que dentro do diretório A tenha outro diretório B, com o arquivo foo.txt. Suponha agora que o diretório B seja copiado, para o diretório C, que está acima do diretório A. O programador que acessar foo.txt a partir do diretório A terá necessariamente que especificar o caminho absoluto, independente do sistema operacional.(F)
- Nome e número de diretório desempenham uma relação biunívoca sempre. (V)
- Para que um arquivo seja deletado é necessário que todos os seus links lógicos sejam deletados.(F)
- Os atalhos também são gerenciados por sistemas de arquivo. (F)
- Considere a permissão 400. Nela apenas o dono pode escrever. (F)
- Volume representa uma instância de disco rígido e só pode existir apenas um dele por sistema operacional.(F)

Aula 27

- O acesso ao conteúdo dos arquivos sempre se dá de forma sequencial.(F)
- Todo sistema de arquivos possui 4 elementos básicos: Diretório, Estrutura de Indexação, Mapas de Espaço Livre e Heurísticas de Localidade. (V)
- Todo diretório também é um arquivo. (V)
- A Estrutura de indexação é armazenada na memória RAM (F)
- Cada processo tem a sua própria estrutura de indexação para garantir o princípio de isolamento. (F)
- As estruturas de indexação mapeiam o nome do arquivo diretamente para o número do bloco inicial do arquivo. (F)
- Todos os programas podem, mediante uma chamada de sistema, alterar a estrutura de indexação e os mapeamentos. (F)
- Toda leitura de arquivo começa pelo bloco inicial, mas pode saltar de acordo com um offset nos demais blocos.(F)
- Mapas de espaço livre mapeiam apenas os endereços de memória não ocupados. (F)
- Bitmaps possuem definidos explicitamente os endereços de memória aos quais correspondem. (F)
- Heurísticas de localidade são utilizadas para evitar o acionamento de partes mecânicas do disco. (V)
- Todo programa precisa saber o endereço físico da estrutura de indexação. (F)

- Para criar um arquivo basta alocar espaço no sistema de arquivos. (F)

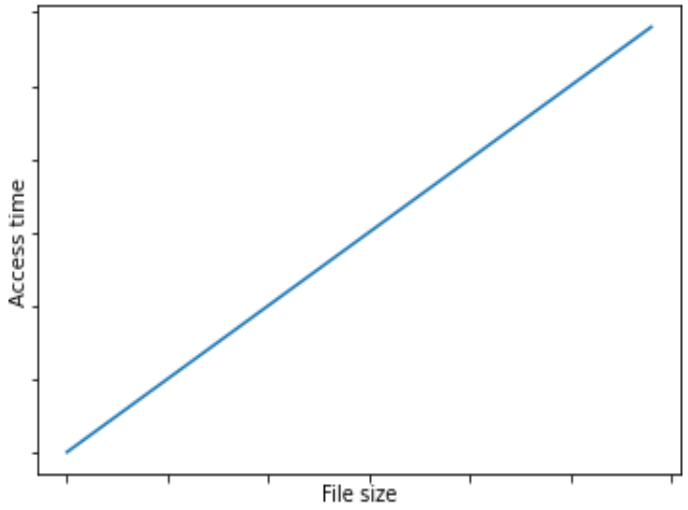
Aula 28

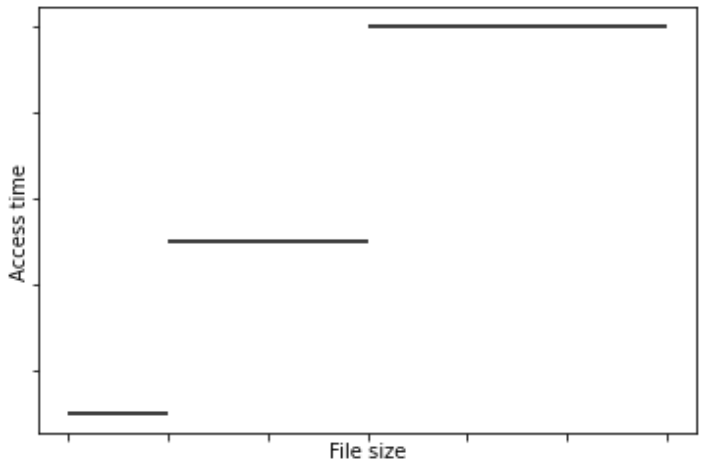
- A estrutura de indexação FAT se organiza como uma lista encadeada de blocos. (V)
- No FAT, todos os blocos possuem mesmo tamanho de memória. (V)
- Mais de um arquivo pode estar armazenado no mesmo bloco. (F)
- O disco também está sujeito a fragmentação externa e interna.(V)
- Cada arquivo tem apenas um bloco associado. (F)
- Cada entrada na Tabela FAT possui apenas um bloco correspondente.(V)
- A lista encadeada da Tabela FAT impede o acesso a uma parte do arquivo utilizando apenas o offset. (F)
- A cada entrada da tabela FAT armazena um ponteiro para a entrada estritamente abaixo dela. (F)
- O uso de blocos muito pequenos faz com que o tempo médio de acesso a um arquivo aumente muito. (V)
- A heurística Next Fit percorre a tabela de entradas até encontrar o primeiro espaço disponível. (V)
- O uso de blocos de tamanho muito pequenos torna mais lenta a criação e alocação de novos arquivos. (V)
- Todo arquivo pode ter uma quantidade arbitrária de tamanho, desde que limite o máximo do disco. (V)

Aula 29

- Uma desvantagem de utilizar um vetor de ponteiros com único nível é que o tamanho dos arquivos ficam limitados a quantidade de ponteiros existentes. (V)
- Para lidar com arquivos grandes, o vetor de ponteiros de única camada precisaria ser grande o suficiente de acordo com a necessidade. Como é inviável ter vários tamanhos de vetor de ponteiros, o uso de um vetor relativamente grande para todos os arquivos geraria fragmentação interna. (V)
- O uso de uma estrutura fixa de camadas faria com que o tempo de acesso para todos os arquivos, independentemente do tamanho, crescesse bastante.(V)
- A complexidade de acessar um arquivo, na estrutura de FFS, é proporcional ao tamanho de cada arquivo. (V)
- Os primeiros blocos de memória de um arquivo são acessados mais rapidamente que os demais.(V)
- Para ler um trecho intermediário sempre é preciso acessar os blocos anteriores no FFS.(F)
- O inode é a estrutura responsável por mapear cada arquivo. (V)
- Qualquer programa pode acessar todos os inodes do sistema. (F)
- Normalmente a escrita em disco não é feita instantaneamente. Isso permite que o sistema de arquivos receba novos pedidos de alocação e gerencie melhor o uso de blocos. (V)
- Quando um arquivo precisa mais espaço do que um bloco, uma vez que o primeiro já foi alocado todos os outros blocos disponíveis têm igual probabilidade de serem alocados para o arquivo em questão. (F)
- A vantagem de organizar os grupos por cilindro é evitar que o braço se mova mudando de trilha. (V)
- Suponha que você possa escolher onde armazenar os inodes. A configuração que melhor trás desempenho é colocá-los apenas no disco mais externo. (F)

- A vantagem de colocar os inodes na mesma trilha que seus grupos associados é diminuir a movimentação do braço.(V)
- O sistema de arquivos busca, sempre que possível, alocar um arquivo dentro do mesmo grupo que seu inode, independentemente do seu tamanho. (F)
- Um dos problemas do FFS é em relação a arquivos muito pequenos, pois proporcionalmente a maior parte do espaço será ocupada com inode/bloco de dados não utilizado.(V)
- Veja as figuras abaixo:



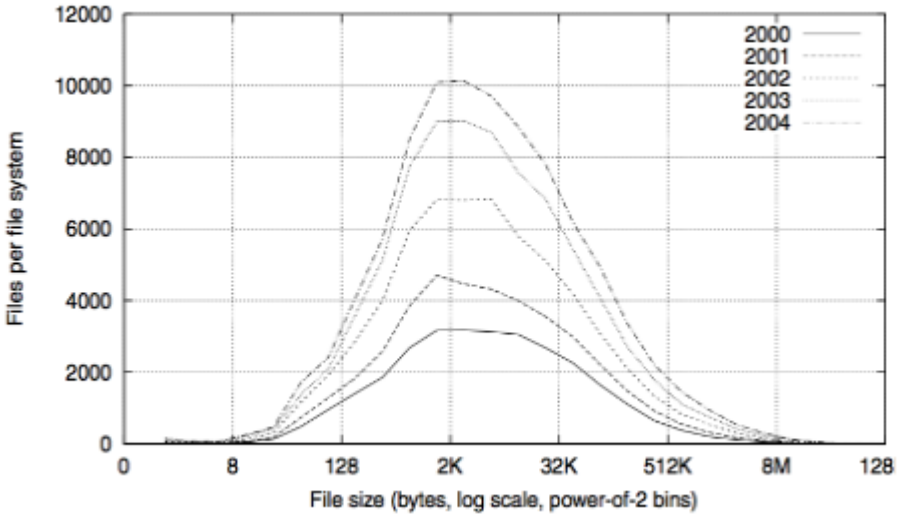


Podemos dizer que a segunda representa efetivamente o FFS enquanto a primeira o FAT?(V)

Aula 30

- No NTFS atributos residentes são todos os metadados e dados que cabem no registro.(V)
- Cada registro no Master File Table tem tamanho fixo. (F)
- Cada registro no Master File Table tem um tamanho máximo permitido. (V)
- Os registros no Master File Table podem referenciar outros quando o tamanho do arquivo ultrapassa o limite estabelecido. (V)

- A lista de atributos só é utilizada quando não é possível alocar todos os ponteiros para extensões no registro. (V)
- As extensões só são utilizadas quando os atributos não cabem no registro. Para isso o registro inclui uma sessão de dados não residentes com os ponteiros necessários para as extensões. (V)
- Os registros no Master File Table devem ser imutáveis para garantir a persistência e coerência dos dados. (F)
- Uma das principais vantagens do NTFS sobre o FFS é que o tamanho das árvores são dinâmicos.(V)
- Veja a figura abaixo:



Dada a tendência de ter maior quantidade de arquivos pequenos, podemos dizer que o NTFS seria a solução mais adequada? (V)

- Quanto mais fragmentado o disco, menor o tamanho das extensões, maior a quantidade de extensões necessárias e consequentemente maior a profundidade das árvores.(V)
- No NTFS o mapa de bits também é utilizado para heurísticas. Isso porque parte dele é armazenado na RAM, logo arquivos que foram criados em tempos similares tendem a ficar próximos no disco também, dado que muito provavelmente a mesma partição do bitmap foi utilizada para alocar espaço.(V)
- A principal importância de se reservar um espaço exclusivo para a MFT é evitar a sua fragmentação. Isso porque quanto mais fragmentado for o arquivo, maior a movimentação necessária no disco para acessar os dados e consequentemente pior o throughput. Além disso, como o MFT é um dos arquivos mais acessados, é importante dar prioridade de alocação de memória a ele.(V)