Uma imagem contendo placa, foto, laranja, comida

Descrição gerada automaticamente

Os microcomputadores e outros tipos de equipamentos digitais, como celulares, *tablets*, *smartphones* estão completamente integrados em nossas vidas. Apesar de serem diferentes, em relação a suas respectivas finalidades, tamanho e formato, há muita semelhança dentro deles. Todos funcionam por meio do controle de um sistema operacional (SO).

**1.1 Conceito e evolução de Sistemas Operacionais (SOs)**

Vamos discutir a evolução dos sistemas operacionais, desde um pouco antes de eles começarem a existir. Para isso, vamos entender a evolução do computador, até chegar aos atuais microcomputadores. Acreditamos que, desta forma, você terá um panorama mais completo para compreender a real importância de um sistema operacional.

Talvez você não saiba, mas muito tempo já se passou desde a criação do primeiro computador, em 1642, por um cientista francês chamado Blaise Pascal (TANENBAUM; BOS, 2016). O invento era muito diferente do que vemos nas máquinas atuais, pois era basicamente uma máquina de calcular, mesmo assim, já era incrível. Essa é considerada a geração zero dos computadores.

Somente em 1943 foi criada a primeira máquina eletrônica, batizada de Colossus, com o objetivo principal de decifrar mensagens codificadas por um equipamento chamado Enigma (TANENBAUM; BOS, 2016). Alan Turing esteve envolvido na criação desta máquina eletrônica, considerada como o primeiro computador. Infelizmente não houve evolução, visto que foi mantido em segredo por mais de 30 anos, vindo ao conhecimento do público apenas após este período.

A primeira máquina da segunda geração foi o TX-0, que não fez muito sucesso. Não se acreditava no sucesso dos computadores naquela época.

Em 1958, surgem os circuitos impressos, nos quais os componentes eletrônicos são integrados, de onde vem a terminologia de circuito integrado, e com isso temos a terceira geração de computadores. Já era possível construir máquinas menores, mais rápidas e baratas.

A evolução não parou por aí e surgiram os circuitos impressos de integração em escala muito grande, os chamados chips *Very Large Scale Integration* (VLSI). Assim, iníciou a quarta geração de computadores.

Nesta quarta geração, o avanço foi tão grande que os computadores começaram a ser chamados de minicomputadores, eram tão mais baratos que até mesmo uma pessoa física podia comprar um deles. Nascia neste momento o computador pessoal (microcomputador).

### *1.1.1 Conceitos de Sistemas Operacionais e classificação*

Não foram apenas os equipamentos (*hardware*) que evoluíram, com o surgimento do sistema operacional algumas evoluções levaram a diferentes tipos de SOs. Temos o SO que funciona apenas em modo texto

O acesso e uso a esse tipo de SO é feito por comandos, cada um com função específica, digitados no teclado, como: dir, del, cls, mkdir. Outro tipo de SO é aquele com interface gráfica.

Não é apenas pelo tipo de interface que os sistemas operacionais podem ser classificados, mas também segundo a forma como obtemos esses sistemas operacionais. Nesta classificação, podem ser comprados (pagos) e são chamados de sistemas operacionais proprietários, como, por exemplo, o Microsoft Windows.  Ou, então, são de domínio público, gratuito, geralmente referido por *software livre*, como o Linux (NEMETH; SNYDER; HEIN, 2007). Os sistemas operacionais também podem ser classificados quanto a como gerenciam a memória. Existem os que gerenciam apenas memória física e tiveram uso nos primórdios do desenvolvimento dos computadores e também dos SOs, como o CP/M e o MSDos, ambos aposentados hoje em dia.

E há os que gerenciam memória física e virtual, e são todos os SOs modernos, tais como Linux (NEMETH; SNYDER; HEIN, 2007), Windows, Mac OS (TANENBAUM; BOS,  2016). Existem várias versões do Linux, do Windows e do Mac OS, que surgiram com o passar do tempo e com o lançamento de novas versões que sempre traziam diferenças das anteriores. Além disso, você também deve saber sobre a virtualização de memória e as categorias de estruturação do SO, temas que vamos abordar no próximo item.

### *1.1.2 Virtualização e estruturação do SO*

Você sabe o que é virtualização de memória? É capacidade do SO (em conjunto com o equipamento) de gerenciar mais memória do que a máquina tem de fato. Funciona assim: parte do SO e parte das aplicações (programas do usuário), que estão sendo usadas ou executadas, são mantidas na memória física (real) do microcomputador, as outras partes são guardadas no disco rígido da máquina.

Quando uma parte que não está na memória (do SO ou da aplicação) é necessária, o próprio SO escolhe uma região da memória e a desocupa. Para isso, grava o conteúdo dessa região no disco rígido, e então carrega para a memória a parte que passou a ser necessária.

Esse processo de salvar - carregar - salvar é o que define a virtualização e por meio disso a memória do microcomputador só acaba quando esgotam a memória física e a virtual (em disco rígido). É uma maneira do SO, junto com o equipamento, realizarem coisas além do que é possível com a memória física que possuem.

Quanto à necessidade de carregar uma parte do SO ou da aplicação que está no disco, isso acontece, em geral, quando o usuário requisita alguma ação (pelo teclado ou *mouse*) como, por exemplo, carregar um aplicativo para acessar a internet. Posteriormente, quando esse aplicativo estiver funcionando pode pedir mais memória para carregar uma página web ou um filme. Em resposta a essa solicitação, ocorre o processo de virtualização da memória.

Para iniciar uma requisição ao SO ou aplicativo deve-se usar um conjunto de procedimentos, as chamadas de sistema (ou chamadas de SO), que é o mecanismo através do qual os aplicativos conversam com o SO. Um usuário nunca usa diretamente uma chamada de sistema, ele usa um aplicativo que fará o uso.

Esse conceito de virtualização é tão poderoso que é possível (e de fato ocorre) que um SO execute outro SO. Quando fazemos uso desse tipo de estratégia estamos usando a virtualização de SO, muito utilizada em grandes instalações de provedores de serviço de hospedagem de aplicativos, ou mesmo por questões de segurança de dados.

Quanto à estruturação do SO, normalmente temos níveis de responsabilidade dispersos nas seguintes categorias:

* o núcleo (*kernel)* no qual estão os trechos relativos às funções primordiais do SO, sem as quais não haveria nenhuma condição para que o SO funcionasse. Existem SOs que possuem núcleos enormes e outros com núcleos menores. Quanto maior for o núcleo, maior será o consumo de memória que o núcleo irá requerer para manter-se em funcionamento no microcomputador. Também, à priori, quanto maior for o núcleo, maior será o conjunto de funcionalidades que estarão prontas para serem usadas pelos processos sem demandar o envolvimento de memória virtual, ou mesmo da troca de conteúdo de memória. Todavia, quanto maior o núcleo, maior será sua complexidade e maiores serão as possibilidades de problemas. Já os sistemas operacionais com núcleos menores geralmente adotam a estratégia de código transiente complementar, códigos que são anexados (adicionados) ao núcleo apenas quando (e se) necessários, e uma vez que não estejam mais em uso são removidos do núcleo e, portanto, da memória do microcomputador. Nessa abordagem, pode-se ter uma vantagem relacionada ao sistema operacional poder controlar um dispositivo novo, recém-criado, para isso basta o SO receber o acréscimo do código correspondente a esse novo dispositivo;
* os códigos adicionados quando necessários normalmente são denominados de *drivers* (controladores) de dispositivo, ou simplesmente *drivers*. Sua finalidade é permitir que o SO consiga estabelecer comunicação com um novo equipamento que tenha sido conectado ao microcomputador como, por exemplo, um *pendrive*, ou uma câmera de vídeo. A criação desses controladores por vezes é feita pelos fabricantes do dispositivo ou pelos criadores dos SOs. Se você for um fabricante de dispositivos vai querer que todos os SOs possam usar o seu dispositivo, pois é vantajoso para a sua empresa produzir o próprio controlador e disponibilizá-lo junto ao equipamento na hora da venda.
* programas utilitários e executores de comandos são programas com finalidades bem específicas e permitem que o SO realize funções altamente especializadas como, por exemplo, interagir com o usuário. Pense nos utilitários como a caixa de ferramentas que permite que você realize tarefas de manutenção do SO. Remover lixo e manter o SO limpo, ou instalar um utilitário de proteção do SO contra vírus são bons exemplos do potencial desses utilitários;
* códigos de carregamento e inicialização são utilizados quando o microcomputador é ligado e tem por finalidade carregar o SO para a memória e, então, entregar o controle do microcomputador ao SO.

Também é importante conhecer como os SOs são estruturados, saber de quantas e quais partes são compostos. Tipicamente, os SOs são compostos dos seguintes módulos:

* **gerenciador de processos:** cuida da criação, execução e controle de todas as tarefas no microcomputador. Faz parte do gerenciador de processos manter o controle dos estados que um processo pode assumir. Ele é fundamental no SO, pois o tempo de processamento de um microcomputador precisa ser compartilhado entre as diversas aplicações (processos) que estiverem sendo executadas. A distribuição desse tempo disponível é feita de acordo com critérios estabelecidos previamente, que podem levar em conta o tamanho de uma tarefa a ser realizada, ou a prioridade do processo.
* **gerenciador de memória:** permite ao SO ter controle do uso e da distribuição da memória disponível no microcomputador, inclusive com o uso de memória virtual quando necessário. Como todo recurso é finito, e a memória não é exceção, torna-se fundamental o controle. É preciso compreender que não é possível manter todos os aplicativos simultaneamente em execução na memória (e no processador do microcomputador), tornando-se, portanto, necessário algum mecanismo de controle que permita que os processos requisitem e liberem memória para atender as suas necessidades;
* **gerenciador de dispositivos:**atua no estabelecimento das condições necessárias para que os diversos dispositivos do microcomputador possam ser utilizados pelos diversos processos que estejam em execução e que necessitem desses recursos. Há dispositivos que podem ser usados de modo compartilhado, como o teclado ou o dispositivo de acesso à rede ethernet, por exemplo, e há aqueles que têm uso restrito a apenas um processo por vez, como o modem ou mouse. A responsabilidade do gerenciador de dispositivos limita-se a estabelecer as condições para que os processos consigam comunicar-se com o dispositivo instalador. Para isso, torna-se necessário que o gerenciador de dispositivos tenha conhecimento sobre todos os dispositivos existentes.
* **gerenciador de arquivos:** embora esteja fortemente associado ao gerenciador de dispositivos, trata com especificidade do mecanismo necessário à organização, criação, leitura e escrita de arquivos nos dispositivos disponíveis no microcomputador. Quando um processo faz uso exclusivo, por exemplo, de um arquivo em disco, é tarefa do gerenciador de arquivos não permitir que outro aplicativo tenha acesso ao arquivo. O cenário torna-se ainda mais complexo quando se considera que um arquivo pode ser aberto para leitura, escrita ou ambas simultaneamente. Assim, deve-se considerar que para a leitura vários processos podem abrir o arquivo ao mesmo tempo, todavia, para a escrita torna-se um problema se mais de um processo estiver escrevendo ao mesmo tempo no mesmo arquivo, pois pode ocorrer perda de dados ou o arquivo pode ser corrompido. Para complicar ainda mais o cenário, é preciso entender que um disco rígido, por exemplo, pode ser dividido em várias partes (partições) e cada uma destas partições pode ser organizada (e codificada) de maneira diferente.

Alguns SOs podem ter mais módulos do que esses apresentados, ocorre que em algumas construções de SO os arquitetos preferem subdividir responsabilidades, na tentativa de deixar o código-fonte mais fácil de compreensão e manutenção. É comum encontrar um módulo que seja responsável exclusivamente pelo suporte a rede de dados, ou então um dedicado a parte gráfica do SO, ou ainda um responsável exclusivamente pela implementação das políticas de segurança.

Importante destacar que o SO não deve impor políticas, mas disponibilizar recursos (mecanismos) para a imposição de política. Com isso, enfatiza-se que não é uma função do SO permitir ou não que um usuário instale um aplicativo de jogo em seu microcomputador. O SO deve, sim, prover os meios para que o usuário faça as instalações que julgar conveniente, visto que a função do microcomputador e, portanto, do SO, é atender as necessidades dos usuários (quando possíveis), sejam elas quais forem.

### *1.1.3*Design*arquitetural, espaço do núcleo e espaço do usuário*

No núcleo de construção monolítica, todos os elementos (módulos), que compõem o SO, fazem parte de um único bloco, ainda que partes de bloco tenham responsabilidades diferentes. Cada parte tem uma finalidade e estabelece intercomunicação com as demais, sem necessitar de protocolos de comunicação, critérios de identidade e políticas de segurança. A velocidade de comunicação entre os módulos é a mais alta possível por tratar-se de uma estrutura na qual todos os módulos têm acesso a regiões de memória que são compartilhadas entre eles. Contra esse modelo de arquitetura de construção de núcleo de sistema operacional afirma-se que ele leva a uma programação (criação dos códigos-fonte) muito complexa e com forte dependência entre as diversas partes, de forma que a falha em uma delas leva o núcleo a degenerar rapidamente.

Na segunda abordagem, que direciona para um padrão arquitetural em camadas, as camadas mais inferiores provêm serviço para as superiores, de forma que quanto mais baixa for a camada, maior será o envolvimento com o *hardware* do microcomputador. Essa arquitetura permite que mudanças em uma camada sejam facilmente isoladas das camadas acima e abaixo, de forma que cada uma possa evoluir individualmente, sem que o todo possa configurar-se como um código extremamente complexo.

Entre as camadas é estabelecido um protocolo de comunicação que permite que serviços e recursos providos pelas camadas inferiores sejam acessados de maneira estável, ordeira e com fácil aplicação de políticas de segurança e de registro das atividades para fins de auditorias futuras.

Vimos, até aqui, que o núcleo do SO é o que de fato controla o microcomputador, portanto, deve ter uma maneira de proteger o próprio SO para que um aplicativo do usuário, por exemplo, não consuma toda a memória do microcomputador e acabe destruindo o SO que estava na memória. Para essas situações foram criados dois espaços bastante distintos: o espaço do núcleo e o espaço do usuário. No espaço do núcleo, apenas códigos pertencentes ao SO podem ser executados, qualquer uso deste espaço por outros aplicativos resultará em um erro e o programa que tentar isso será encerrado pelo SO.

Caso um aplicativo precise de algum serviço do SO, ou de algum dado que esteja no espaço do SO, deverá acessar esses serviços ou dados utilizando-se das chamadas de sistema. A principal finalidade das chamadas de sistema é permitir a comunicação entre o SO e os aplicativos, mas, ao mesmo tempo, manter separados e seguros os espaços do núcleo e o do usuário.

Mas se por algum motivo o usuário precisar que algum aplicativo funcione no espaço do núcleo? Nesta situação, não há como um aplicativo do usuário funcionar no espaço do núcleo, todavia, o usuário pode separar algumas partes do seu aplicativo e reescrevê-las como um *driver* e, então, pedir ao SO para carregar esse *driver*. Depois de carregado, as funções desse *driver* serão executadas no espaço do núcleo e poderão ter acesso direto às funcionalidades do núcleo, mas é uma região muito delicada, qualquer erro e o SO travará (ou reiniciará) e o usuário terá de reiniciar o microcomputador. Dependendo do tamanho do estrago causado, poderá ser necessário reinstalar todo o SO e, nesta situação, o usuário poderá ter grande perda de dados.

## 1.2 Gerenciamento de recursos

Nosso primeiro passo aqui é estabelecer o entendimento do que se refere à palavra recurso no contexto dos sistemas operacionais. Por recurso**,** devemos compreender toda funcionalidade ou capacidade que esteja presente no microcomputador. Alguns recursos são providos através de *hardware* e outros de *software*, portanto, há duas categorias de recursos. Como exemplo de distinção, vamos citar a capacidade de uso de interface gráfica. Tanto no Windows como no Linux, há um motor gráfico com o papel de prover funções primárias de manipulação de interfaces gráficas, tais como: criação de um espaço de interação, ou criação de um objeto de interação. Os espaços de interação devem ser compreendidos como todos os espaços em que o programador pode inserir conteúdo e outros elementos na tela. Quando você inicia um aplicativo em um ambiente gráfico, geralmente ele cria uma visualização gráfica que fica contida dentro de uma janela. Esta janela é um espaço de interação em que são criados novos objetos, normalmente com a finalidade de serem utilizados na interação entre o programa e o usuário, por isso são denominados de objetos de interação.

Note que nesse cenário há claramente a colaboração entre dois recursos: o de *hardware*, representado pela placa gráfica e pelo monitor, e de *software*, que no exemplo é provido pelo motor gráfico.

Em geral, para quase todos os recursos que interagem com o usuário, é usada uma abordagem parecida. Há um recurso de *software* que gerencia as saídas de áudio, e há o *hardware* de áudio que produz o som. Agora que você compreendeu isso, aproveite para ouvir uma música tranquila enquanto lê sobre o gerenciamento de recursos.

### *1.2.1 Conceitos básicos e aplicabilidade de gerenciamento de recursos*

Os recursos que estão disponíveis em um microcomputador são limitados: memória (mesmo virtualizada), espaço em disco rígido, capacidade de processamento. Todo elemento de *hardware* e *software* adicionado ao microcomputador torna-se um recurso que pode ser usado por um único aplicativo, ou compartilhado entre vários outros.

Tome como exemplo uma placa de rede, que permite ao microcomputador conseguir comunicar-se com outros equipamentos mesmo que estejam do outro lado do mundo, o acesso a este recurso é compartilhado entre todos os aplicativos que estejam instalados no microcomputador.

A tarefa de gerenciamento desses recursos cabe ao SO e ele tem de lidar com as questões relacionadas ao controle de estado de um recurso (pronto, disponível, desligado etc.), ao controle de qual aplicativo (e usuário) tem acesso ao recurso, manter um histórico de quem faz uso dos recursos, alocar e desalocar os recursos e manter controle dessa variação de *status*.

Também é tarefa do sistema operacional controlar o quanto um usuário ou aplicativo pode ter de tempo e uso de um determinado recurso. Esse balanço entre quem usa e por quanto tempo é a chave de um gerenciamento de sucesso em qualquer SO. O tempo que um processo utiliza é o tempo em execução em alguma das unidades de processamento central (CPU) que o microcomputador possua. Atualmente, os microcomputadores possuem várias CPUs e quanto maior o número de CPU, maior a capacidade de processamento.

### *1.3.1 Definição de processo*

Podemos entender como é o processo de um aplicativo sendo executado, com parte do seu código de execução (ou todo, se for pequeno) em memória real (ou virtual), além, também, de existir um contexto associado. Um contexto de um processo refere-se a todos os recursos que ele esteja usando, como memória, arquivos abertos no disco rígido, conexão com a impressora. Para que um processo seja corretamente executado, ele precisa do seu código de execução e de seu contexto. Em alguns sistemas operacionais, o termo usado para um processo pode ser tarefa ou *job*.

Durante o período em que um determinado processo está ativo ele pode ser encontrado em vários estágios:

a) novo – quando o processo está em fase de criação;

b) pronto -  quando o processo está preparado e aguardando ser direcionado para a execução em uma CPU;

c) esperando -  quando o processo está esperando algum recurso ou evento, para que seja executado;

d) executando – é o estado no qual o processo está em execução em alguma CPU;

e) terminado – quando o processo não está mais em execução e o seu contexto já foi liberado.

### *1.3.2 Diagrama de execução e estados do processo*

Na Figura a seguir, podemos ver como os estados e a interação entre os estados que um processo pode ter. Observe que entre o início (novo) e o término (terminado) um processo percorrerá várias vezes os estados possíveis.

A transição do estado de **novo** para **pronto** ocorre logo após a criação do processo, em seguida será escalonado (selecionado para a execução) para a primeira execução, e neste período fará a solicitação aos recursos que venha a necessitar. Perceba que o estado **pronto**se refere à situação na qual o processo tem reunida todas as condições para ser executado. A partir da solicitação de recursos, que ocorre na primeira execução ou ao longo do tempo em que o processo estiver ativo, ele poder ter a transição para o estado de **esperando**, e ficará até que os recursos solicitados sejam fornecidos ao processo. Quando um processo está no estado pronto, ele aguardará apenas na fila para ser escalonado, e durante sua execução pode passar aos estados de esperando (no caso de solicitar novos recursos) ou pronto (no caso de terminar o seu *quantum* de execução e precisar aguardar por um novo período) e de terminado*.* O estado de **terminado** ocorre quando o processo chega ao fim, não restando mais nenhum processamento a ser feito. A mudança de estado de um processo ocorrerá sempre em função da prioridade, respeitando-se o tipo de política de escolha para a execução que estiver em uso pelo gerenciador de processos, que este processo possui (menor prioridade recebe menos tempo de execução), bem como da disponibilidade dos recursos que o processo vier a solicitar.

Cada processo é representado por um bloco de controle de processos, o qual consiste em um conjunto de metadados sobre os processos, normalmente referenciado por *Process Control Block*(PCB). Um PCB contém informação sobre:

* o número do processo que o identifica;
* o estado atual do processo, dentre  os vários estados que um processo pode assumir;
* um contador do programa que aponta o endereço de memória que contém a próxima instrução do código de execução do aplicativo que deverá ser executado quanto o processo mudar para o estado de executando;
* os registradores da CPU, que são variáveis mantidas pelo SO e fundamentais para a execução do processo;
* dados que indicam a prioridade do processo dentro da política de escalonamento que estiver em uso no SO;
* a gerência de memória para o processo;
* o *status* de E/S que inclui dados dos recursos (dispositivos) de E/S utilizados pelo processo.
* Na Figura a seguir, podemos ver uma ilustração de como o PCB é gerenciado no momento em que o SO precisa trocar de contexto para salvar e carregar um novo processo para a execução em alguma CPU.

Diagrama

Descrição gerada automaticamente

Perceba que, quando vai ocorrer uma troca de processo na CPU (período inativo), o contexto (PCB) do processo que estava em execução é salvo em outra região da memória. Em seguida, o contexto do próximo processo a ser executado é carregado, e somente após essa carga é que o processo poderá ter sua execução iniciada.

De maneira geral, segundo Tanenbaum e Bos (2016) e Deitel (2005):

* o escalonamento FCFS tem por direcionamento atender o processo que chegar primeiro, resultante direta da tradução literal de *First come, first served*, para o qual temos que o primeiro a chegar, será o primeiro a ser servido sendo um algoritmo não preemptivo. O termo preemptivo refere-se à política de escalonamento na qual o processo é interrompido sempre que o SO julgar necessário. Dentre os motivos que justificam o SO interromper o processo está o término da fatia de tempo (*quantum)* que o processo possui;
* o SJF não preemptivo, no qual se escolhe o processo que represente o menor trabalho, introduz aqui um problema relativo a: como saber qual é o menor trabalho?SJF vem do inglês *Shortest Job Fisrt*, que significa o menor trabalho primeiro. Por ser não preemptivo este processo, uma vez iniciado na CPU, é executado até o seu término;
* o SJF preemptivo também escolhe o processo que represente o menor trabalho e interrompe o processo após o término do *quantum* de tempo;
* o escalonador por Prioridade Preemptivo faz o escalonamento baseado na simples prioridade dos processos e encerrando-os quando esses processos esgotarem os seus respectivos *quantum* de tempo;
* o algoritmo escalonador Round Robin é idêntico ao FCFS, porém é preemptivo, assim, trata-se de um sinônimo de escalonamento por revezamento, e é uma abordagem bastante comum.

## 1.4 Interagindo com o gerenciador de processos

O gerenciador de processos possui certos níveis de configuração, mas pode definir arbitrariamente a prioridade de um determinado processo. Apesar de termos a impressão de que um microcomputador executa vários processos ao mesmo tempo, isto não é fato. Ele executa apenas um processo por *quantum* de tempo, que nada mais é do que a menor fatia de tempo que um processo recebe para ser executado em uma CPU. A priori, todos os processos recebem o mesmo *quantum*de tempo, mas o usuário pode definir novas divisões deste tempo.

Caso o usuário decida que um processo é mais prioritário, será dada uma fatia maior de tempo para esse processo, portanto, o seu *quantum*de tempo será maior do que dos demais processos, fazendo com que seja executado mais rápido. Para que ocorra a interação dos aplicativos com o sistema operacional sempre será utilizado algum recurso de comunicação entre aplicativos. Vamos conhecer a linguagem de programação a seguir.

### *1.4.1 Usando linguagem de programação*

A comunicação entre aplicativos ocorre através da *application programming interface* (API), que significa interface de programação de aplicativos, constituída de um conjunto de chamadas de sistema e ou chamadas de aplicativos.  É com essas API que qualquer programa se comunica com o sistema operacional e por vezes com outros programas que estejam em execução no microcomputador. Em alguns sistemas operacionais, a comunicação entre aplicativos recebe o nome de *interprocess comunication* (IPC).

Na Figura a seguir podemos ver o conjunto de API para acesso a memória no Windows. As APIs são utilizadas por meio de linguagens de programação, seja a linguagem Java, C, C++, dentre muitas outras.

Assim, perceba que o SO, muito além de gerenciar o microcomputador, também fornece funcionalidades que por vezes não estão presentes no *hardware*. Essas funcionalidades são relativas aos serviços que o SO fornece e que podem ou não estarem ligados ao *hardware*. Essas funcionalidades por vezes são referenciadas como **funcionalidades de *software***, a exemplo podemos citar o uso por vários usuários simultâneos (multiusuário) via rede; os serviços de autenticação de usuário; os serviços de proteção de dados (uso de senha; encriptação; criação de pastas; criação de arquivos etc). Há um conjunto amplo dessas funcionalidades de *software* providas pelo SO, que não são relacionadas ao *hardware*, mas que de alguma forma melhoram o seu acesso, ou até incluem características diferentes de uso.

Dessa forma, podemos afirmar que um bom SO melhora o equipamento, enquanto um SO ruim irá piorar o desempenho da máquina.

Veja que, na Figura com o exemplo de API, as chamadas de sistema são acessadas usando as APIs, por meio da linguagem de programação de seus comandos próprios. Em C, por exemplo, utiliza-se *malloc* para solicitar memória. Internamente, na linguagem, usa-se uma das APIs do SO para solicitar memória. Por exemplo, no Windows tais APIs possuem nomes que são diferentes dos nomes usados no Linux. Uma imagem contendo Linha do tempo

Descrição gerada automaticamente Assim, o encadeamento correto é: programa do usuário - comandos da linguagem de programação - APIs - chamada de sistema. Não se preocupe em decorar esse grande encadeamento, ele ficará cada vez mais claro ao longo da disciplina.

Texto

Descrição gerada automaticamente

No exemplo, *open*, *creat*, *read*, *write* e *close* são comandos da linguagem C. Internamente na linguagem C, quando usados no SO Linux, por exemplo, serão redirecionados para a API *syscall*(). Esta API fará, enfim, a chamada de sistema correspondente, ficando parecida como “syscall( 0x5 )”  estivesse solicitando a chamada de sistema de abertura de arquivo, cujo número de chamada no sistema de macros do Linux fosse 0x5 e apontará para o trecho de código que pertence ao *kernel* do SO. No Linux, a relação das chamadas de sistema pode ser vista no arquivo localizado em: /usr/include/sys/syscall.h.

### *1.4.2 Simulação de processos*

Você sabe em que consiste uma simulação de processos? É um aplicativo que permitisse a simulação de execução de processos e experimentasse (e compreendesse) de perto a dinâmica de um gerenciador de processos. Para realizar essa tarefa é necessário que se estabeleçam as seguintes condições:

a) um sistema que permita a criação de tarefas, associando necessidades de recursos para essas tarefas;

b) um mecanismo de fornecimento de recursos para atender as tarefas;

c) um escalonador que gerencie os estados das tarefas, alternando aqueles que estarão em execução e ou nos demais estados.

Claro que tudo isso como uma simulação.

Texto

Descrição gerada automaticamente com confiança média

## 2.1 A linguagem de programação e o SO

Obviamente, os sistemas operacionais não se comunicam com os processos, tampouco com o usuário, utilizando-se da linguagem humana (DEITEL; DEITEL, 2011). De fato, a capacidade de fala humana é tão complexa que ainda hoje existem vários estudos e pesquisadores com a finalidade de desenvolver habilidades de compreensão e fala da linguagem humana para microcomputadores. Tais estudos estão no campo do processamento da linguagem natural e são muito interessantes.

### *2.1.1 Linguagem de programação*

Linguagem de programação para computadores (ou simplesmente linguagem de programação) é um conjunto de palavras-chave (instruções) que são utilizadas para compor sequências de comandos, que uma vez executados na ordem definida, produzem um resultado planejado. Existem tipos de linguagens de programação de baixo nível e de alto nível. Em Deitel e Deitel (2011), por exemplo, podemos ver uma ampla discussão em torno da linguagem de programação C, que pode ser utilizada tanto em baixo nível, como em alto nível.

Mas vamos definir o que são essas linguagens. Linguagens de baixo nível são aquelas que lidam diretamente com questões associadas ao *hardware*. Tanenbaum e Bos (2016) nos dizem que:

Assim fica claro que a linguagem de baixo nível está associada a questões tecnicistas e programas feitos nessas linguagens em geral são para um *hardware* específico. Se você usa uma linguagem de baixo nível para escrever um programa que permitirá que o usuário digite um texto no teclado do fabricante X, esse programa não funcionará se o teclado for da marca Y.

Desse modo, usando linguagem de baixo nível produzimos os *drivers* e o próprio sistema operacional. Alguns utilitários são feitos com linguagens de baixo nível, como as linguagens *Assembly* e a C.

Já as linguagens de alto nível são mais apropriadas para a construção de aplicativos diretamente usados pelo usuário, ou para aquelas que não lidam com questões de *hardware*. Um programa para internet, como um navegador *web*, é feito em linguagem de alto nível. Mas mesmo os aplicativos (e as linguagens de alto nível) do usuário utilizam o *hardware*. Então, como é possível que uma linguagem de alto nível faça acesso ao *hardware*, visto que esse tipo de linguagem não é próprio para tal feito? A resposta para essa pergunta chama-se abstração.

A abstração é, portanto, uma maneira que nos permite esconder detalhes técnicos que são requeridos, mas os quais não deveríamos nos preocupar, e deviam ser feitos automaticamente, para assim nos concentramos no que realmente importa.

O sistema operacional produz uma abstração enorme que esconde o *hardware* e o disponibiliza aos programadores de computador por meio de um conjunto de chamadas de sistema, o qual é utilizado pelas linguagens na produção de APIs, que são as interfaces programáveis usadas pelos aplicativos. Na Tabela a seguir, podemos ver uma parte da relação de APIs em um comparativo entre o Windows e sistemas Unix.

Quando um programador deseja criar um aplicativo que, por exemplo, crie uma nova pasta no disco rígido, ele teria de produzir algo parecido como a seguinte sequência de comandos da linguagem C++:

Forma

Descrição gerada automaticamente com confiança média

Todavia, se a linguagem utilizada fosse Java, ficaria algo como:

Forma

Descrição gerada automaticamente com confiança média

O acesso às outras interfaces de programação previstas na API do SO é chamado de maneiras semelhantes aos dois trechos de linguagem de programação apresentados. E em alguns casos o uso é até mais fácil do que o modo demonstrado nos exemplos.

### *2.1.2* Threads

Um aplicativo normalmente é desenvolvido como uma sequência de comandos que são executados para realizarem uma determinada tarefa. Estamos falando de algo como o que ilustra a Figura a seguir.

Diagrama

Descrição gerada automaticamente

Pela Figura anterior, observe que na Parte 1 temos um programa composto de 5 instruções (A, B, C, D, E) que são executadas sequencialmente, isto é, primeiro A, depois B, e assim sucessivamente. Ocorre que, eventualmente, pode-se usar um recurso de programação chamado de *thread*, que constitui no primeiro passo em direção à programação concorrente e utilizada quando desejamos que um programa tenha duas ou mais linhas de execução simultâneas.

Uma linha de execução corresponde a uma sequência de comandos que estão executados, sendo que a linha principal é aquela que deu início ao programa e é a tarefa principal para o SO. As *threads* são ramais da linha principal, utilizadas, normalmente, para realizar alguma atividade que seja complementar ao processamento que está sendo executado na linha principal.

*Threads* podem ser usadas para “apressar” o processamento, quando o *hardware* no qual o programa está sendo executado possui mais de uma CPU. Nesse caso é possível indicar à CPU principal que as *threads* devam ser executadas em diferentes CPUs do microcomputador, estabelecendo-se o conceito de paralelismo, no qual a execução de um processo é fragmentada (de maneira planejada e controlada) em diversos segmentos que são executados simultaneamente nas diferentes CPUs do microcomputador.

Interface gráfica do usuário

Descrição gerada automaticamente

Observe na Figura anterior que no momento da captura da imagem do monitor de recursos do Windows, havia 166 processos em execução que geraram 2171 *threads*.

### 2.1.3 Modelos de Criação de Threads

A criação de *threads* de um processo no sistema operacional não é uma tarefa difícil, de fato em várias linguagens a tarefa é bem simples. O trecho de código-fonte, obtido na documentação do Java, listado a seguir, ilustra esse processo:

Forma

Descrição gerada automaticamente com confiança média

Uma observação relevante se refere ao fato de que o processo é o ponto no qual os recursos são solicitados, portanto a problemática de solicitação de recursos cabe ao processo, enquanto a *thread* seria destinada exclusivamente ao trabalho que requer exclusivamente o tempo da CPU.

Uma observação relevante se refere ao fato de que o processo é o ponto no qual os recursos são solicitados, portanto a problemática de solicitação de recursos cabe ao processo, enquanto a *thread* seria destinada exclusivamente ao trabalho que requer exclusivamente o tempo da CPU. Se uma abordagem de *thread* for realizada em um microcomputador com uma única CPU, pode não haver ganho significativo no desempenho do processo, todavia, em máquinas com múltiplas CPUs esse ganho será significativo.

Tanenbaum e Bos (2016), enfatizando as vantagens do uso de *threads*, fazem uma analogia da situação para uma aplicação de edição de texto. Se o processador for construído em duas partes, uma interativa e que mantém contato com o usuário, e a segunda, que faz o trabalho de formatação do texto, poderá haver grande benefício para o usuário. Caso o usuário faça uma alteração em uma página, em uma parte inicial de um livro, a *thread* que trabalha na formatação trabalharia para reformatar o livro. Assim, quando o usuário se mover pelas páginas do livro (as páginas que ficam depois da alteração), já vai encontrá-las corrigidas, tudo graças ao trabalho feito em paralelo pela *thread* interativa.

Tanenbaum e Bos (2016), enfatizando as vantagens do uso de *threads*, fazem uma analogia da situação para uma aplicação de edição de texto. Se o processador for construído em duas partes, uma interativa e que mantém contato com o usuário, e a segunda, que faz o trabalho de formatação do texto, poderá haver grande benefício para o usuário. Caso o usuário faça uma alteração em uma página, em uma parte inicial de um livro, a *thread* que trabalha na formatação trabalharia para reformatar o livro. Assim, quando o usuário se mover pelas páginas do livro (as páginas que ficam depois da alteração), já vai encontrá-las corrigidas, tudo graças ao trabalho feito em paralelo pela *thread* interativa.

### *2.2.1 Escalonamento (FIFO e prioridades)*

Até este ponto, você já percebeu o quanto o SO é relevante para o bom funcionamento e uso dos microcomputadores, não é mesmo? De fato, sem o SO, usar um microcomputador seria uma tarefa totalmente técnica. Lembrando-se sempre de que outros tipos de equipamentos, além dos microcomputadores, também têm o funcionamento facilitado pelo uso de algum tipo de SO. Dentre as diversas tarefas que o SO realiza, a responsabilidade de selecionar qual processo (dentre os muitos que possam existir) deve ser executado em primeiro, segundo etc. é certamente a mais relevante para a percepção do bom funcionamento por parte dos usuários do microcomputador.

A decisão sobre qual processo deve ser executado recebe o nome de escalonamento, existindo, portanto, os algoritmos de escalonamento. Mas qual o motivo para existirem diferentes algoritmos de escalonamento? Em primeiro lugar, é preciso esclarecer que alguns dos algoritmos de escalonamento têm função acadêmica, no contexto de SO, pois podem não ser boas escolhas para o tipo de tarefa que os microcomputadores possuem atualmente.

Antes, contudo, vamos rever um conceito importante chamado de preempção. Um dos significados da palavra é o de preferência, todavia, para os algoritmos de escalonamento, preemptivos são aqueles nos quais o processo que está em execução pode ser suspenso a qualquer momento, já os não preemptivos, uma vez iniciados, serão executados até o fim. Perceba que pela existência da preempção, há motivos pelos quais o processo que está em execução possa ser suspenso. E quais seriam esses motivos?

O primeiro deles é quando ocorre uma interrupção. Uma interrupção no contexto de SO refere-se a uma sinalização de *hardware* ou de *software* (no caso o *software* será uma das partes do SO) que indicam à CPU que um evento de prioridade ocorreu em alguma parte do *hardware* do microcomputador, ou em alguma parte prioritária do código do SO. O termo técnico para uma interrupção é requisição de interrupção, do inglês *interrupt request*(IRQ)*.*Quando uma interrupção é sinalizada, o processo que estava em execução é suspenso, a CPU irá então executar o trecho de código que irá atender a IRQ. Este trecho de código está no espaço do núcleo e foi posto na memória pelo SO e pertence ao SO. Quando o trecho de código que atende a IRQ for concluído, a execução retornará ao processo que estava sendo executado no momento em que ocorreu a IRQ.

O segundo momento em que o processo pode ter a execução suspensa está relacionado ao tipo de algoritmo de escalonamento. Dentre eles destacamos o FCFS, SJF Não Preemptivo, SJF Preemptivo, Prioridade Preemptivo e Round Robin. O FCFS objetiva atender o processo que chegar primeiro, dentro da política “primeiro a chegar, primeiro a ser servido”, do inglês first come, first served. Já o SJF vem do inglês short jobs first, que em tradução livre significa “o menor trabalho primeiro”, sendo que esse algoritmo pode ser preemptivo ou não. O algoritmo por prioridade preemptivo baseia-se na prioridade do processo, conceito de que pode variar de implementação para outra em diferentes SOs. Uma sugestão de leitura para aprofundar ou relembrar detalhes sobre esses algoritmos é a encontrada em Deitel (2005) e Tanenbaum e Bos (2016).

Neste capítulo, todavia, vamos abordar o algoritmo por prioridade, com tempo definido de execução e preemptivo e também o algoritmo first in, first out (FIFO), que é uma abordagem simplificada de escalonamento. Porém, antes vamos destacar um pouco sobre algoritmos em lote.

Um algoritmo em lote tipicamente é usado quando existe um conjunto conhecido e finito de processos que devem ser executados e para os quais não há necessidade de interação, durante o processamento, entre o SO e o usuário. O conjunto de processos tem a execução iniciada e somente após o seu término um novo conjunto poderá ser admitido. Todavia, dentro do lote, qual deve ser o primeiro processo a ser executado?

Bom, a execução em lote, a escolha do primeiro, segundo etc. processo a ser executado pode ser determinada pela ordem de inscrição no lote. Aqui a política será do tipo “quem entrar primeiro na fila, será atendido primeiro e sairá primeiro”, o que a caracteriza com FIFO. Outra ordem possível de execução pode ser pelo critério de “menor trabalho”, estimado pelo usuário e informado durante a criação do lote de processos. Neste tipo de algoritmo os processos de menor tamanho (menor trabalho) serão executados primeiros.

Entretanto, ocorre que nos sistemas operacionais modernos para microcomputadores não há aplicação prática desses tipos de algoritmos, visto que uma característica principal que encontramos nos SOs atuais é a interação com o usuário. De fato, a interação evoluiu tanto que hoje os processos interagem com o usuário via teclado, mouse, tela sensível ao toque, e até mesmo com o uso da voz e sons. Por fim, note que, embora, o conceito de escalonamento por lote possa não ser uma boa política para um SO moderno, não quer dizer que em outras situações não possa ser usado. Existem muitos servidores na internet, como os servidores web, que usam esse tipo de processamento em lote para gerenciar pools (conjunto) de portas de conexão de seus clientes.

Como você pode perceber, uma política FIFO talvez não seja uma solução ideal quando temos processos e ambientes interativos entre o SO e o usuário. Surge então o conceito de execução dos processos de maneira compartilhada. Neste compartilhamento o tempo disponível para o processamento será dividido igualmente entre todos os processos. Essa parcela de tempo que cada processo recebe é denominada de quantum. Em uma abordagem simples, o primeiro processo a chegar seria a prioridade um. Caso surja outro processo, o primeiro processo continuará a ser a prioridade um, e será executado pelo seu quantum de tempo e, ao término desse tempo, será a vez da prioridade dois. Dessa forma, em algum momento, todos os processos serão prioridade e estarão em execução pelo seu quantum de tempo. Como cada processo é interrompido ao término do seu quantum de tempo, esse algoritmo por prioridade preemptivo.

2.2.2 Troca de contexto

Compreendemos que nos sistemas operacionais modernos ocorre constantemente um processo de substituição do processo que está em execução na CPU. Esse processo está associado diretamente à política de escalonamento que estiver em uso no SO, e junto com a decisão de substituir um processo por outro na CPU várias ações precisam ser tomadas pelo SO.

O primeiro passo que deve ocorrer quando o SO decide realizar a troca de processo na CPU é providenciar a proteção de todos os dados que estejam em memória e também a situação em relação a eventuais recursos que possam ter sido solicitados pelo processo, além de informações sobre a próxima instrução que deverá ser executada quando o processo retornar. Os dados, os recursos e as informações sobre a próxima instrução compõem o contexto de um processo.

O processo de proteção do contexto pode envolver simples remanejamento de dados de um ponto para outro na memória RAM ou deslocamento de dados da memória RAM para o disco, esta decisão é tomada em consideração ao estado de alocação da memória RAM no instante em que for ocorrer uma troca de contexto.

Para uma manipulação mais facilitada, as informações sobre o contexto do processo são reunidas em uma estrutura de bloco de controle. O bloco de controle de processo normalmente é denominado por PCB, do inglês process control block e estão registradas muitas informações sobre o processo, como, por exemplo:

* O ponteiro contador do programa, que tem por finalidade exclusiva indicar o endereço da próxima instrução que deverá ser executada para o processo. É por esse contador que a CPU saberá qual instrução deve ser executada, mesmo com várias trocas de contexto. Quando um processo é encerrado, ou entra em espera, um novo processo é escalado para a execução, e dentre as muitas coisas que precisam ocorrer, uma delas é carregar para a CPU a instrução que está contida neste contador;
* Além do contador de programa, outros controles são necessários para que a CPU execute corretamente a diversidade de processos que podem estar carregados. Esses controles são denominados por registradores, compreendidos como variáveis que pertencem a CPU. Cada processo em execução terá valores diferentes de registradores, portanto quando ocorre a troca de contexto, esses registradores precisam ser corrigidos, processo a processo. O contador de programas é um desses registradores;
* Registrador de base de memória, que também é um registrador, e demais informações associadas, dentre elas a tabela de páginas de memória, ou tabelas de segmento. Sem essas informações a CPU não saberia em que parte da memória os dados do processo estariam;
* Informações de entrada e saída, relacionando os dispositivos que estão em uso pelo processo, inclusive a relação de arquivos que estão abertos, dentre outras informações necessárias.

O ponto central a ser compreendido aqui é que toda troca de processo a ser executada é precedida de uma atualização do PCB do processo em execução, atualizando-se os valores dos registradores da CPU, do ponteiro contador de programa, e das informações de entrada e saída. Em seguida, ocorre um salvamento de PCB do processo que estava em execução. Neste ponto ocorre o carregamento de um novo PCB e ajustes nos registradores da CPU para coincidirem com os dados do PCB do processo que irá ser executado. Após a realização dessa troca de contexto, o processo terá sua execução iniciada e/ou continuada na CPU, repetindo o ciclo de salvamento e carregamento enquanto o microcomputador estiver em uso.

2.3 Escalonamento de processos

Você deve ter percebido que a característica mais marcante no ambiente de funcionamento do SO é a constante troca de processos. De fato, é o principal fator de relevância de um SO, que pode ser bonito e atrativo visualmente, oferecer uma infinidade de funcionalidades, mas se for ruim na gestão dos processos, estará fadado ao fracasso.

A chave para o sucesso de um SO está intrinsecamente ligada à forma como gerencia os processos e, ainda mais especificamente, à forma como ele escolhe qual processo deve ser executado. Essa escolha pode envolver várias questões, tais como a idade do processo (há quanto tempo está sendo executado), ou a quantidade de tempo que ele está esperando, ou ainda o tamanho do trabalho a ser realizado pelo processo.

Cada uma dessas questões traz consigo diversas possibilidades de ação e é justamente a escolha tomada que torna o SO bom ou ruim. Em Tanenbaun e Bos (2016) e em Machado e Maia (2013), há uma extensa discussão sobre os algoritmos de gerenciamento de tarefas. Já em Deitel (2005), ocorre uma discussão com viés mais prático do que teórico. Na sequência, vamos analisar uma abordagem de escalonador de tarefas e ver os prós e contras da possível abordagem.

2.3.1 SJF-Preemptivo

Tipicamente o algoritmo SJF-preemptivo (lembre-se de que SJF significa “a tarefa menor primeiro”) (MELO NETO, 2014) associa duas abordagens em uma só: a primeira refere-se à escolha da menor tarefa; a segunda, ao uso de divisão de tempo com término arbitrário (preemptivo) ao término da divisão de tempo da tarefa (término do quantum de tempo). Sobre o SJF, Silberschatz, Galvin e Gagne (2015, p. 63) nos dizem que:

o algoritmo SJF é um caso especial do algoritmo geral de scheduling por prioridades. Uma prioridade é associada a cada processo, e a CPU é alocada ao processo com a prioridade mais alta. Processos com prioridades iguais são organizados no schedule em ordem FCFS. O algoritmo SJF é simplesmente um algoritmo por prioridades em que a prioridade (p) é o inverso do próximo pico de CPU (previsto). Quanto maior o pico de CPU, menor a prioridade, e vice-versa (SILBERCHATZ; GALVIN; GAGNE, 2015, p. 63).

Sobre a escolha da tarefa de menor trabalho, há alguns problemas a serem superados. O primeiro dele diz respeito a como determinar o menor trabalho. Este de fato é o mais significativo problema envolvendo o algoritmo SJF, pois não há uma maneira definitiva de determinar essa informação. O tamanho de uma tarefa não pode ser determinado pelo tamanho do aplicativo, nem pela quantidade de memória que ele utiliza. Tampouco pela quantidade dos demais recursos que ele utiliza.

O segundo problema refere-se à possibilidade de que algum processo nunca venha a ser executado por ele ser “grande” e o SO sempre preferir escolher tarefas de menor tamanho. Em um cenário em que surjam vários processos de tamanho pequeno, os processos que forem maiores nunca serão executados.

2.4 Comunicação e sincronização de processos

A comunicação de um processo pode ocorrer entre o processo e o SO, ou então entre diferentes processos. Quando ocorre entre processos é denominada de comunicação entre processos. Há ocasiões em que um processo produz dados para serem usados (consumidos) por outros processos. Como exemplo, um navegador web, que não se comunica diretamente com o hardware, mas faz uso dos recursos de rede, através da comunicação entre processos. Essa comunicação também pode se dar por meio do uso de API e chamadas de sistema.

No caso típico de produtor-consumidor, damos o nome de pipeline, que caracteriza a comunicação entre processos, na qual a saída de um é a entrada de outro. Além da comunicação, tem de ocorrer uma sincronização entre os processos. O consumidor não deve tentar ler os dados antes que o produtor os disponibilize, portanto é preciso estabelecer uma estratégia em que o produtor possa informar ao consumidor que há dados disponíveis.

A comunicação e a sincronização também estão presentes quando dois processos, digamos A e B, desejam ter acesso ao mesmo recurso, embora quando A esteja usando o recurso, este deva ficar indisponível para B, então B poderá usar quando A não estiver usando, e assim sucessivamente. Aqui a sincronização deve auxiliar na solução do uso compartilhado do recurso.

Vamos a um exemplo. Um processo X, que produz dados que devem ser impressos, e Y é o processo que deve imprimir os dados. Y deve sincronizar seu trabalho com o processo X para que a impressão comece apenas depois de X ter disponibilizado dados para serem impressos.

Para o estabelecimento da comunicação entre processos (ou com o SO) existem as seguintes abordagens: a) utilizar API ou chamada de sistema; b) usar memória compartilhada; c) usar encadeamento de saída/entrada (pipeline).

Quando se utiliza a API, ou chamada de sistema, para o estabelecimento de comunicação se estabelece um cenário bastante previsível de comunicação, no qual o uso da API define os mecanismos envolvidos nessa comunicação, pois as API determinam quais dados devem ser fornecidos ao solicitar um serviço, bem como quais dados serão retornados como resultados da execução do código correspondente a API.

Por outro lado, caso o recurso de comunicação entre os processos seja uma abordagem baseada no compartilhamento de memória, então será necessário o estabelecimento de estratégias de apoio que todos deverão cordialmente seguir.

No compartilhamento de memória estabelece-se um cenário no qual dois ou mais aplicativos terão acesso à mesma região da memória, e será necessário estabelecer de que forma irão compartilhar essa memória. Este cenário em particular introduz uma problemática que é o controle de concorrência e região crítica, que veremos logo adiante.

Já na abordagem de pipeline simplesmente um processo, digamos X, inicia um processo Y, este produz dados e os disponibiliza através, por exemplo, da saída padrão (normalmente é o vídeo). Assim, enquanto Y estiver produzindo dados, X poderá ser executado normalmente. Não há necessidade de se estabelecer outros mecanismos de controle da comunicação.

2.4.1 Programação concorrente

Sempre que dois ou mais processos necessitam usar os mesmos recursos ocorrem condições de corrida. Suponha que o processo A deseja escrever dados em uma região da memória que está compartilhado com o processo B e com o processo C. Suponha que em um determinado momento, o valor da região da memória seja 20 (vinte), todos os processos A, B e C leem esse valor e começam o processamento. Supondo-se que o processamento que deva ocorrer seja somar 1 ao valor que estava na memória, o correto seria que ao término do trabalho encontrássemos o s novo valor na memória: 23 (vinte e três).

Mas supondo-se que todos os processos estejam trabalhando em prioridades diferentes, com o término do processo B ele vai escrever o novo valor, que será 21 (vinte e um). Neste instante, o processo B é suspenso e quem passa a ser executado é o processo A (que também leu o valor 20, lembra-se?). O processo A conclui seu processamento e escreve o valor que encontrou (20 + 1) que é igual a 21 (vinte e um). O mesmo ocorre com o processo C que também leu o valor 20, calculou o valor 21 e o escreveu.

O resultado desse processamento e da condição de corrida será que ao término do processamento de A, B e C o valor de memória será 21 e não 23, que deveria ser o valor correto. Este tipo de problema de concorrência pode ter graves consequências.

Para superar este tipo de problema uma estratégia bastante utilizada é o estabelecimento de regiões críticas e o uso de semáforos para controlar quem tem acesso à região, definindo-se assim apenas um por vez.

2.4.2 Problemas de compartilhamento de recursos

Da situação analisada nas condições de corrida, percebe-se que é necessário organizar solidamente o acesso a recursos compartilhados, pois os problemas de compartilhamento de recursos podem causar graves prejuízos de funcionamento aos processos com resultantes de perdas de dados para o usuário. A condição de corrida em relação aos recursos compartilhados torna a região crítica, também chamada de seção crítica.

A implementação de semáforos na solução de problemas de compartilhamento de recursos é um problema que sempre preocupou os desenvolvedores de SO e os programadores que fazem uso de técnicas que envolvem o compartilhamento. A solução que envolve a implementação de semáforos tem sido amplamente utilizada e a partir disso são estabelecidas duas vertentes de soluções. Caso você tenha ficado curioso em saber mais, leia a seção “Semáforos” do capítulo “Sincronização de processos” em “Fundamentos de Sistemas Operacionais” (SILBERSHATZ; GALVIN; GAGNE, 2015).

Algumas estratégias de controle envolvem o estabelecimento de semáforos para o controle de acesso à região crítica. E o que vem a ser semáforos? Trata-se de uma variável de controle. O processo que conseguir “travar” a variável terá acesso à memória. Essa trava poderia ocorrer da seguinte forma: a variável terá inicialmente o valor zero, se um processo ao ler a variável, encontrar o valor zero então ele muda o valor para um, e a variável fica travada, pois os demais processos encontrarão o valor um, indicando que a variável está travada.

Mas essa solução que parece criativa e simples está errada, pois ela tem o mesmo problema que tenta resolver, portanto solução não resolve nada.

Tentativas de solucionar o problema do compartilhamento de recursos encontraram soluções de dois tipos: a espera ocupada e a espera não ocupada (dormir). O primeiro ocupa tempo de processamento da CPU, sendo, portanto, um processo no estado de “executando”, enquanto que o segundo não ocupa tempo de processamento da CPU, estando na maioria dos SOs modernos, no estado “esperando”. Este estado em particular é usado vinculando-se o processo que se deseja que “durma” a alguma IRQ de software. Quando a IRQ ocorrer o processo será acordado.

Na espera ocupada, o processo que irá esperar pelo recurso ocupará tempo de processamento da CPU, o que é uma estratégia ruim. Já na outra abordagem, o processo que espera pelo recurso será colocado em espera e somente será acordado quando o recurso estiver disponível para ele.

Bom, você percebeu que o trabalho do gerenciador de tarefas é enorme e que se não for bem desenvolvido o SO inteiro será de qualidade inferior, ou até mesmo inútil. Sendo assim, de nada adiantará quaisquer outras funcionalidades e características fantásticas se o SO não conseguir lidar bem com o gerenciamento das tarefas.

Foto de uma placa

Descrição gerada automaticamente com confiança baixa

**3.1 A comunicação de mensagens e a sincronização entre os processos**

Percebe-se, facilmente, que os processos, enquanto executados, geram uma intensa comunicação interna, portanto, produzem uma enorme quantidade de mensagens que são trocadas entre eles, na forma das APIs, chamadas de sistema e até mesmo recursos compartilhados. Tal é a importância dessa intensa comunicação que qualquer erro leva a trágicas consequências, como o encerramento prematuro de processos e até falhas gerais do SO.

A comunicação ocorre e é necessária para que os processos utilizem os recursos que estão disponíveis no microcomputador por meio do sistema operacional. Mas você precisa compreender que as transferências de dados entre a unidade de processamento central, a memória RAM e o disco rígido também são tipos de comunicação.

Você pode se perguntar: como assim? Por exemplo, quando você edita uma imagem em seu microcomputador usando um aplicativo gráfico, para que a imagem seja exibida no display, foram necessárias diversas comunicações, que, inclusive, envolveram a transferência da imagem original para a memória RAM e também para a memória do display.

Portanto, a comunicação é caracterizada por mensagens curtas, médias e grandes. Ainda em nosso exemplo, considere que durante e após a edição da imagem, você precisa salvar o que foi feito. Esse processo de registrar o trabalho no disco rígido envolve um conjunto extenso de comunicações.

Gráfico

Descrição gerada automaticamente

A exemplo do entrelaçamento das comunicações entre os processos, observe a Figura anterior que apresenta a comunicação que ocorre (generalizada) para o uso do Binder no sistema operacional Android (TANENBAUM; BOS, 2016). O Binder é utilizado para que aplicações do espaço do usuário possam invocar e utilizar recursos que estejam disponíveis em equipamentos alcançados por intermédio da rede de dados. Trata-se, portanto, do uso de recursos compartilhados a distância. E esse compartilhamento requer o entrelaçamento das comunicações entre processos feito de forma correta. Além disso, devemos considerar os problemas relacionados à memória RAM, tais como: proteção, alocação e desalocação de trechos, localização e fragmentação das partes livres da memória.

Ou seja, tudo precisa ocorrer em sintonia entre os processos e os diversos hardwares que compõem o microcomputador, o que é trabalho do sistema operacional.

A comunicação entre os processos é estabelecida de uma maneira fortemente estruturada, normalmente chamada de protocolo. O protocolo pode ser muito simples, do tipo “quando um estiver falando, o outro apenas ouve”, mas há situações em que requer vários dados repassados do emissor para o receptor. Esses dados repassados são chamados de conteúdo da comunicação ou parâmetros.

Tanenbaum e Bos (2016, p. 28) trazem que “processos relacionados que estão cooperando para finalizar alguma tarefa muitas vezes precisam comunicar-se entre si e sincronizar as atividades”. Isso estabelece uma das razões pelas quais os processos se comunicam. Outras razões incluem: solicitar algum tipo de serviço; a sincronização de alguma tarefa; o envio ou recebimento de dados; a notificação de resultado de processamento; e o erro ou sucesso nas tarefas. Portanto, como há diferentes motivos pelos quais os processos se comunicam, há também diferentes maneiras de estabelecer e realizar a comunicação. Mesmo assim, há previsibilidade quanto aos possíveis tipos de comunicação e dessa forma torna-se possível criar protocolos e/ou tipos padrões de mensagens que serão trocadas.

Suponha que exista um processo X que forneça os serviços de data e hora. Para atender uma demanda entre X e os outros processos que venham a se comunicar com ele, cria-se o seguinte padrão para a comunicação: para solicitar a data deve ser usada a mensagem *obterData*() e para obter a hora, a mensagem *obterHora*(). Como você percebeu, essas mensagens são parecidas com as APIs e o modo de usá-las estabelece o protocolo de comunicação. Suponha ainda que seja possível enviar uma mensagem para alterar a data e/ou a hora: *definirData*(valor) e *definirHora*(valor). O valor que surge dentro dos parênteses é um indicativo de que que junto à mensagem deve ser enviada nova data ou hora para a qual X deve alterar os valores anteriores. Assim, para alterar a data, por exemplo, para 22 de maio de 2018, usaríamos: *definirData*(“22/05/2018”).

Essa comunicação é denominada comunicação entre processos (IPC), do inglês *interprocess communication.* Tanenbaum e Bos (2016) estabelecem três questões que precisam ser tratadas em relação ao funcionamento dos processos e que envolvem IPC na solução.

A primeira diz respeito à simples necessidade de troca de mensagens (visando troca de dados ou serviços). Quando processos desejam apenas se comunicarem, basta o uso do mecanismo correto para o envio do tipo certo de mensagem.

A segunda questão refere-se ao estabelecimento da sequência correta de ações quando dependências estão presentes. Isso é para evitar que processos se atrapalhem, o que geralmente acontece quando dois ou mais estão interessados no mesmo recurso, o que chamamos de relação produtor-consumidor. Como exemplo, imagine o processo A que lê e carrega dados do disco rígido; o processo B que decodifica dados relativos a músicas e vídeos; e o processo C que apresenta músicas e vídeos para o usuário. Essa relação precisa funcionar da seguinte maneira: A(B(C())), isto é, o processo A utiliza os dados produzidos por B, que utiliza os dados produzidos por C. Isso estabelece uma sequência correta de ações visando o alcance de um resultado.

Já a terceira questão refere-se a processos não se atrapalharem mutuamente. Isso significa que o interesse em um mesmo recurso por vários processos, como o estabelecimento da necessidade de compartilhamento de algo. Processos de uma mesma família podem desejar acessar um mesmo arquivo de dados, ou uma mesma região da memória. Quando uma situação como essa ocorre, encontramos uma condição de corrida, o que torna o trecho do processo envolvido uma região crítica.

Para a solução de acesso a um recurso compartilhado é necessário estabelecer uma política que defina o momento certo para um processo acessar o recurso e, ao mesmo tempo, definir que os demais não podem ter o acesso.

Isso requer algum tipo de solução que garanta a exclusividade de acesso à região crítica do código que trata do recurso compartilhado. O que é importante, pois, do contrário, pode ocorrer inconsistência nos dados, já que há uma ordem específica em que os processos deveriam acessar e modificar os dados do recurso compartilhado. Essa solução em particular é conhecida por exclusão mútua. Tanenbaum e Bos (2016) propõem o estabelecimento de quatro condições para que se encontre uma boa solução para a exclusão mútua:

1. Dois processos jamais podem estar simultaneamente dentro de duas regiões críticas;

2. Nenhuma suposição pode ser feita a respeito de velocidades ou do número de CPUs;

3. Nenhum processo executando fora de sua região crítica pode bloquear qualquer recurso;

4. Nenhum processo deve ser obrigado a esperar eternamente para entrar em sua região crítica (TANENBAUM; BOS, 2016, p. 83).

Diagrama

Descrição gerada automaticamente

### ***3.1.1 Algoritmo de Peterson/semáforos***

Então são dois grandes problemas que precisam ser resolvidos: 1 - garantir que dois processos não cheguem à região crítica simultaneamente; 2 - garantir que todos os processos que compartilham a região crítica tenham a sua vez de acesso.

Uma proposta para solucionar os dois problemas foi dada por Peterson (TANENBAUM; BOS, 2016), que é do tipo “espera ocupada”, cujo algoritmo na linguagem de programação C pode ser visto no Quadro a seguir.

Texto

Descrição gerada automaticamente

Na solução apresentada no Quadro, o grande segredo está na condição while (turn == process && interested[other]== TRUE. Podemos ler o trecho da seguinte forma: permaneça esperando enquanto for a nossa vez e o outro processo ainda está interessado. Observe que o uso de leave\_region indica que o valor de interested para o processo na região crítica é informar que não tem mais interesse: interested[process]=FALSE. Quando isso ocorrer, o processo que estava no “permaneça enquanto (...)” perceberá que quem o estava usando perdeu o interesse, logo podemos usar.

Contra a solução proposta por Peterson está o fato de ela necessitar do conhecimento prévio da quantidade de processos que irão compartilhar o recurso. Caso a solução seja empregada estratégia que permita o aumento do número de processos que compartilham o recurso, a solução não funcionará. Pesa também contra essa solução, o fato de consistir em uma espera ocupada, obviamente implicando no consumo da CPU e significativo desperdício da capacidade de processamento.

inda sobre a solução de Peterson, observe que existem algumas variáveis (turn, interested) usadas para regular o acesso à região crítica. Essas variáveis atuam como semáforos e têm a função de sinalizar para que os demais processos percebam se podem ou não acessar à região crítica. Portanto, temos uma técnica combinada de semáforos e inserção obrigatória de código de controle da região crítica.

A inserção obrigatória de código de controle de região crítica refere-se à necessidade de todos os processos incorporarem as chamadas enter\_region e leave\_region. Sem essa obediência a estratégia falharia.

A problemática que envolve o controle das regiões críticas pode ser aumentada quando consideramos um cenário em que não apenas diferentes processos pertençam a mesma família, mas, inclusive, a famílias diferentes. Considere que um usuário inicia um aplicativo de navegador para acessar diversas páginas de conteúdo na web. Suponha agora que o usuário também inicie um aplicativo para acessar um serviço de banco e, ao mesmo tempo, resolva ouvir música em algum aplicativo diretamente da internet.

Todos esses aplicativos estão fazendo uso da comunicação pela rede de dados, ou seja, todos competem pelo uso da rede, recurso controlado pelo SO. Como isso acontece? Na verdade, em uma visão generalista, são problemas resolvidos com o mesmo tipo de estratégia: permitir que apenas um processo use o recurso por vez.

No caso do envio e recebimento de dados pela rede, a grande saída é delegar toda a atividade a duas únicas partes do sistema operacional. Todas as aplicações e seus inúmeros processos sequer tentam fazer acesso direto ao recurso compartilhado (rede), eles enviam e recebem dados por meio dessas partes específicas do sistema operacional, fornecido por uma chamada de sistema.

Centralizando o acesso ao recurso e garantindo que apenas um único caminho esteja disponível para o uso do recurso, a ocorrência de condições de corrida torna-se mínima. Essa estratégia também resolve o problema de garantir que nenhum processo fique esperando para sempre sem ser atendido.

Por isso, basta que as solicitações sejam colocadas em uma fila e atendidas com algum algoritmo simples, que pode ser do tipo FIFO, uma política simplificada de lidar com um conjunto de requisições de recursos. Não se preocupe que essa política será explicada um pouco adiante do capítulo. Enquanto isso, vamos seguir nosso estudo com o tema gerenciamento de memória real.

**3.2 Gerenciamento de memória real**

Memória real refere-se à memória de acesso randômico (RAM), do inglês random access memory. É aquela na qual os processos são efetivamente armazenados em tempo de execução. Se um microcomputador possui 8 gigabytes de memória, significa que esse é o limite máximo de memória que pode ser usado na máquina, sem o uso de virtualização. Eventualmente, o proprietário pode comprar mais memória RAM e adicioná-la ao hardware, respeitando os limites aceitos pela placa principal instalada no microcomputador.A memória RAM normalmente tem o design de uma pequena placa de circuito impresso com uma região de conectores que fica em contato na placa principal, nos memory slots.

A memória real é acessada pela CPU e também pela direct memory access (DMA) e em ambas as modalidades o SO pode exercer controle. A tecnologia DMA foi criada para que a transferência ocorresse diretamente por seus canais de comunicação e não exigisse a participação da CPU.

Mas essa tecnologia nem sempre esteve presente. Os primeiros microcomputadores não dispunham da DMA e todo o tráfego de dados gerado pelos processos, e que envolviam a RAM e o disco rígido, contavam com o trabalho da CPU. Nessa abordagem antiga, a CPU também tinha a função de processar a transferência de dados e quando um processo precisava movimentar quantidades significativas de dados, ocorria uma sobrecarga no sistema.

Ainda hoje, mesmo em microcomputadores modernos, alguns tipos de transferência de dados que envolvem apenas a memória RAM são manipulados exclusivamente pela CPU, impactando no desempenho do sistema como um todo.

**3.2.1 Conceitos básicos e funcionalidades**

Os processos não fazem, de fato, acesso direto ao endereçamento real da memória. É produzido um mapeamento com blocos, denominado espaço de endereçamento, cuja finalidade é facilitar o gerenciamento da memória.

Por meio do espaço de endereçamento é criada uma abstração que faz com que o processo tenha “sua” própria memória. Assim, o endereçamento de um processo sempre será diferente do utilizado por outro, exceto quando há memória compartilhada entre eles.

O gerenciador de memória fornece uma parcela da memória real para o processo quando solicitado. Caso a quantidade de memória pedida não esteja disponível, o processo é informado, e então é fornecida a quantidade máxima possível. Nesse caso, cabe ao processo decidir se irá realizar o processamento esperado ou se ele deve ser interrompido devido à falta de recursos. A partir desse pressuposto, torna-se necessário que os programadores também se esforcem para que seus programas sejam adaptativos, de forma que essa adaptação diz respeito aos seus programas terem capacidade para realizar o processamento necessário de modo fragmentado, sem que essa fragmentação comprometa o todo. Não estamos aqui nos referindo à uma divisão do trabalho feita em processos menores, o que caracterizaria a programação paralela, mas sim de um processo que precisa, por exemplo, realizar uma transformação em uma imagem. Esse processo deve ser projetado de maneira que se possa trabalhar na imagem inteira por vez (caso tenha conseguido muita memória RAM), ou, então, que se consiga aplicar a transformação da imagem por partes, até que o trabalho tenha sido todo processado e transformado.

Há situações em que não é possível fragmentar o processamento, restando ao processo abortar a missão, notificando o usuário que ocorreu um erro devido à falta de memória.

**3.2.2 Monoprogramação sem trocas de processos ou Paginação**

No início do desenvolvimento dos microcomputadores e das primeiras versões de SO, o hardware não possuía características de proteção da memória. A proteção, no caso, refere-se a impedir que um determinado processo enderece e manipule memória fora do seu espaço de endereçamento.

Um microcomputador sem hardware de proteção torna a multiprogramação impossível. Qualquer programa pode apresentar mau funcionamento acidental (ou intencional) e comprometer toda a memória do computador, destruindo o SO e todos os dados em RAM.

Hoje os microcomputadores modernos contam com a presença de hardware de proteção, mas ainda existe um conjunto enorme de equipamentos, conhecidos como equipamentos embarcados, que não possuem proteção de memória, sendo então monoprogramados.

E como isso é feito? Bom, na monoprogramação não ocorre a necessidade de trocas de processos e, por consequência direta, não ocorre remanejamento da memória, não existindo também a necessidade de paginação. O trabalho de gerenciamento da memória é praticamente desnecessário, visto que toda a memória fica à disposição do processo que esteja em execução.

**3.3 Gerenciamento de memória**

Nos cenários da multiprogramação, a alocação da memória é dinâmica. A dinamicidade vem da imprevisibilidade da quantidade de processos que podem ser executados e também da quantidade de memória requisitada por esses processos.

Um processo pode solicitar determinada quantidade de memória e, momentos depois, aumentar ou diminuir esse pedido. A variabilidade na alocação da memória pelos processos gera um enorme trabalho para o gerenciador e traz problemas relacionados à criação de espaços livres entre os espaços alocados na memória.

Suponha que exista um conjunto de aplicativos do usuário, representados pelos processos (A, B, C e D), que nas linhas de tempo ilustradas na Figura a seguir (a, b, c, d, e, f, g), podem ou não estar em execução nas respectivas linhas. Perceba a dinâmica entre a ocupação e liberação da memória com o término e reinício de alguns desses aplicativos ao longo do tempo.

Diagrama

Descrição gerada automaticamente

Note que na Figura, os espaços hachurados indicam trechos que não estão alocados. Em certas situações, um processo pode ter um pedido de memória negado, mesmo que haja livre, bastando apenas que não exista um bloco de memória livre que seja contíguo ao bloco de memória do processo.

Para minimizar esse tipo de problema, a memória é organizada pelo SO em pequenas frações. Se essas frações forem muito pequenas, haverá um aumento da complexidade do gerenciamento. Portanto, é preciso procurar um equilíbrio entre a quantidade e o tamanho das frações.

Quando a memória é fracionada, para ter a organização e o gerenciamento facilitados, é necessário algum mecanismo para o controle, como o mapa de bits.

3.3.1 Mapa de bits

O mapa de bits é uma maneira de endereçar (com objetivo de controlar) pequenas partes da memória. O controle funciona assim: elege-se uma quantidade de memória que será representada por um bit. Divide-se a quantidade de memória total pela quantidade que foi eleita. O resultado será a quantidade de bits necessários no mapa.

Texto

Descrição gerada automaticamente

Na Figura anterior, a memória está representada na parte (a), o mapa de bits em (b), e os processos com uso da memória em (c). Olhando da esquerda para a direita, perceba que os números “uns” (1) referem-se às porções alocadas da memória e os zeros (0) às porções livres.

Observando o lado superior esquerdo da parte (c), podemos perceber o P05, no qual “P” indica que a região está alocada a um processo, “0” que o trecho alocado tem início em zero e o 5 que o trecho tem o tamanho de 5 bytes, e assim sucessivamente.

**3.3.2 Alocação de segmentos livres**

Na abordagem baseada em mapa de bits, o maior problema consiste na alocação de segmentos livres. Quando um processo é carregado para a memória é necessário encontrar um trecho grande o suficiente para que possa contê-lo. Esse trabalho requer que o gerenciador de memória percorra todo o mapa de bits para encontrar a região livre que satisfaça o tamanho necessário. É uma operação lenta, pois, inevitavelmente, trata-se de uma busca sequencial.

O tempo necessário para isso é diretamente proporcional à quantidade de processos que estejam em memória. Isso ocorre, pois quanto mais processos houver, maior será a necessidade de comparações nos trechos livres encontrados. E lembrando-se de que ao final, pode-se chegar à conclusão de que não há memória contigua livre, mesmo em situações em que, ao somar os vários trechos de memória livre, o resultado seja muito maior do que se procurava inicialmente.

Considere que ocorra uma falha no gerenciamento e na alocação de segmentos livres, o resultado seria uma catástrofe no sistema operacional, pois um determinado processo poderia sobrescrever os dados de outro processo, destruindo-o ou modificando-o de forma que ele realize processamentos diferentes daqueles para os quais tenha sido construído.

Essa preocupação é real e sistemas operacionais com falhas na alocação e proteção de memória estão expostos a vulnerabilidades que abrem espaço para hackers (programadores experientes que usam das falhas para acesso a funcionalidades ou dados no sistema operacional) instalaram trechos de código mal-intencionado e prejudicar os interesses do usuário. Muito bem, agora que apresentamos um panorama sobre a memória real, vamos para outro tipo: a virtual.

**3.4 Gerenciamento de memória virtual**

O gerenciamento de memória virtual tem por finalidade permitir que os microcomputadores pareçam ter muito mais memória do que realmente a instalada no hardware. Junto com essa funcionalidade acaba sendo necessário implementar uma série de recursos para o controle dessa abstração.

Lembre-se sempre de que os processos precisam estar na memória RAM quando forem executados. Caso um processo (ou parte dele) esteja em disco rígido (na memória virtual), será necessário mover o trecho do disco rígido para a memória RAM. Obviamente isso requer tempo: de processamento da CPU, dos canais de DMA e do disco rígido.

Um microcomputador com quantidade adequada de memória RAM pode ter seu desempenho melhorado com o uso de memória virtual, mas esse resultado pode não ser tão positivo se a quantidade de memória real for pequena e os aplicativos que o usuário iniciar requisitarem muita memória.

Suponha existir um microcomputador com 2 gigabytes de memória livre durante a execução por um determinado SO. São carregados 3 processos, e cada um requer 1 gigabyte de memória. Utilizando-se os 2 gigabytes livres, seria possível manter dois processos em memória, porém como são três, um terá de ser continuamente carregado do disco rígido para a RAM e depois carregado da RAM para o disco rígido.

Se consideramos que o tempo de transferência de dados para um disco rígido é muito maior do que o tempo de transferência para a memória RAM, é fácil perceber que este microcomputador estará constantemente efetuando leitura e escrita no disco rígido, o que irá afetar o desempenho geral da máquina.

Mas como minimizar essa situação? Como transformar um processo de 1 gigabyte em um menor? Aliás, isso é possível?

Bem, segundo Silberschatz e Galvin (2015), um processo de 1 gigabyte não pode ser transformado em um menor, mas podemos dividi-lo em partes menores. Estatisticamente, pode-se afirmar que um processo (de 1 gigabyte) não é executado de uma só vez na CPU, apenas uma parte dele, em cada uma das vezes que o processo estiver no estado de executando.

Assim, surge uma estratégia de fracionamento da memória que será utilizada pelos processos em partes menores e diretamente mapeada na memória real.

Para poder implementar a memória virtual, a memória real do microcomputador é organizada em uma abstração denominada de páginas, compostas por endereços contíguos. O endereçamento das páginas não é correspondente aos endereços reais da memória do microcomputador.

**3.4.1 Conceitos de Paginação**

Quando um processo é criado e iniciado, ele pode ser grande demais para ser mantido inteiro na memória. Isso pode ocorrer por não haver memória livre suficiente, ou devido ao total existente no microcomputador ser menor do que o necessário para o processo. Além disso, quando o processo é executado pela primeira vez (ou em qualquer uma das vezes que estiver em execução), ele faz suas requisições de memória e esta pode ser maior do que a memória real disponível no momento. Caso seja o cenário, o SO irá prover mais memória para o processo utilizar (ou até mesmo para acomodar o próprio processo), lançando mão da memória virtual.

As páginas de um processo são simplesmente blocos de divisões da memória relativos ao processo e podem estar na memória real ou armazenadas no disco rígido.

Quando o processo faz referência (indica que vai usar) a um bloco (página) que não está na memória principal (seja memória para todo o processo ou parte dele), é necessário trazer esse bloco do disco rígido para a memória RAM. E para isso acontecer, algum bloco deve sair. Nessa situação, surge uma pergunta crucial ao gerenciamento e implementação da paginação: qual bloco deve sair?

**3.4.2 Implementação de Paginação**

Umas das estratégias para gerir a troca de páginas na memória é o algoritmo first in, first out (FIFO). Mesmo sendo uma estratégia simples de escolha, é muito utilizada na gestão de filas de pedidos de recursos. Tanenbaun e Bos nos mostram (2016) que:

o FIFO remove a página menos recentemente carregada, independente de quando essa página foi referenciada pela última vez. Há um contador associado a cada quadro de página e, de início, todos os contadores estão definidos como 0. Após cada falta de página ter sido tratada, o contador de cada página que se encontra na memória no momento é aumentado em (1) um (TANENBAUM; BOS, 2016, p. 352).

Na estratégia FIFO, a solução é a mais simples possível: quem chegou primeiro, agora tem de sair.

É como em um brinquedo de parque de diversões, a roda gigante, por exemplo. A roda vai girando e todos vão se divertindo. Perceba que as pessoas que estão no brinquedo não chegaram ao mesmo tempo (é o mesmo que ocorre com as páginas de memória virtual que estão na memória real). Quando chega uma nova pessoa para subir na roda gigante, alguém tem de sair. Sairá (como todos nós sabemos) quem estava lá por mais tempo. O primeiro que havia chegado na roda gigante, será o primeiro a sair. E isso é o que define o funcionamento do algoritmo FIFO.

Se no sistema operacional estiver em uso uma política de escalonamento por quantum de tempo, esse sobre e desce da roda gigante, isto é, entra e sai da memória RAM, ocorrerá com frequência. Pois, quando o processo que está em execução já finalizou seu quantum de tempo, ele sairá da memória e da CPU, e um novo processo será carregado. O novo processo certamente terá suas próprias páginas de memória, o que é completamente normal.

O processo de troca de página pelo algoritmo FIFO ocorre sempre que uma das dezesseis páginas da memória virtual for referenciada e não estiver em nenhum dos oito quadros nos quais a memória principal foi dividida.

Se considerarmos também o cenário em que ocorre a programação paralela, haverá um potencial alto de troca de páginas na memória (BARCELLOS, 2002).

Na ocorrência de um volume grande de troca de páginas, pode haver perda de desempenho de modo geral por limitar as funções de entrada e saída da memória à velocidade do disco rígido, que é significativamente menor do que o da memória RAM. Portanto, em programação paralela, é importante que o processo seja executado em um microcomputador corretamente dimensionado para a tarefa.

Tanenbaum e Bos (2016) nos dizem que o conjunto de páginas do processo é denominado conjunto de trabalho e pode ter páginas mais e menos utilizadas.

Texto

Descrição gerada automaticamente

**4.1 Gerenciamento de memória cache**

Vimos que no passado, o microcomputador era um equipamento precário, com poucos recursos, mas que evoluiu bastante, baseado em tecnologias que permitiram que os componentes eletrônicos ficassem cada dia menores e mais baratos.

Naquela época, como o usuário poderia pedir para que o SO carregasse algum aplicativo? De fato, essa questão envolvia um bocado de trabalho, pois o acesso às unidades de armazenamento externas era caro e limitado (DEITEL; DEITEL, 2011).

Basicamente havia três caminhos: fornecer o aplicativo a partir do teclado, ou seja, digitando todos os comandos necessários, todas as vezes que o aplicativo necessitasse ser executado; usar um gravador de fita K7, no qual tivesse gravado previamente uma cópia do programa; e usar um disco flexível, inserindo-o em alguma unidade leitora.

Funcionava deste jeito: você digitava o programa e o testava, se tudo corresse bem, você deveria retirar o disco de inicialização do SO (caso tivesse sido usado na inicialização do microcomputador), depois inserir um novo disco com espaço para salvar o seu programa. Após esse procedimento, você teria seu programa protegido em um disco flexível. Era algo parecido como se estivesse usando um gravador de fita K7, mas as chances de algo dar errado eram muito grandes. Vale lembrar que uma fita K7 permitia a gravação de uma pequena quantidade de kilobytes, enquanto um disco flexível de 3.1/2” pode chegar a 1.2 megabytes.

Dentre as evoluções surgidas, estão os equipamentos ligados à memória, seja de acesso randômico (RAM) que é a memória principal de um microcomputador, até as secundárias, tais como os discos rígidos e os discos rígidos portáteis.

Von Neumann é reconhecido no cenário da Computação como uma pessoa de grande importância, em especial com suas contribuições na área da arquitetura de computadores. De fato, os microcomputadores atuais, todos, sem exceção, possuem influências de seus trabalhos. Para conhecer mais sobre ele, a recomendação é a leitura do artigo (KOWALTOWSKI, 1996), disponível em: < http://www.scielo.br/scielo.php?script=sci\_arttext&pid=S0103-40141996000100022>.

Tabela

Descrição gerada automaticamente

No topo, vemos os registradores e cache nível 1 e nível 2. Esses três itens, que são três tipos de memória, estão integrados, fazem parte da CPU e de dois grandes grupos: a memória volátil, memória que mantém o estado quando alimentados por energia elétrica - registradores, cache e RAM; e a memória persistente ou não-volátil: memória que mantém o estado mesmo sem energia elétrica – HD, pendrive, disquete, dispositivos de armazenamento.

Após as memórias cache encontramos a RAM, ali caracterizada como a memória física (real) e a memória virtual. Na continuação do Quadro, temos os equipamentos de armazenamento de dados e, por fim, os equipamentos de entrada de dados.

Os registradores e as memórias cache são ligados e diretamente relacionados à CPU (DEITEL; DEITEL, 2011), além de apresentarem um tempo de funcionamento extremamente pequeno, portanto são dispositivos que permitem uma transferência de dados em grande velocidade (DEITEL; DEITEL, 2011). Os registradores normalmente são acessados cerca de 1 ns (nanosegundo), já a memória cache tem velocidade de acesso típico próximo de 2 ns. Por fim, a memória RAM apresenta um tempo de acesso na casa de 10 ns. Perceba pela diferença de tempos que as memórias caches são extremamente rápidas e é necessário que sejam assim para que possam ser utilizadas pela CPU sem que seus respectivos tempos afetem o desempenho.

Observe que existe uma grande quantidade de tipos de memórias, subdivididas entre as voláteis e as não-voláteis. E entre esses tipos de memórias também há diferenças, em relação à velocidade. A escolha de qual será instalada no microcomputador é importante, pois afetará o desempenho da máquina.

Para fins didáticos, 1 ns é igual a 0,00000001 segundos, e o ms é igual a 0,001 segundos. Veja a seguinte comparação: você caminhando normalmente tem uma velocidade cerca de 4 km por hora, e um avião viaja a cerca de 850 km por hora. Fazendo as contas, percebemos que o avião é cerca de 200 vezes mais rápido do que você. Mas se a sua velocidade fosse de 1 milisegundo e a do avião 1 nanosegundo, o avião seria 100.000 (cem mil) vezes mais rápido. Logo, a memória cache é 100.000 vezes mais rápida do que a memória RAM.

Tendo em vista que a memória cache é mais rápida que a RAM, ela assume uma função relacionada ao armazenamento de itens (código ou dados) que estão sendo muito utilizados, de maneira a acelerar a execução dos processos. Quando um dado ou parte do código de um processo não se encontra na memória cache, ocorre um novo acesso à RAM.

Tabela

Descrição gerada automaticamente

Descrição: A figura mostra uma Tabela de duas colunas e 13 linhas na forma de uma retângulo grande. A primeira coluna da tabela possui a informação Tipo, que indica o tipo de memória ou dispositivo de entrada ou saída de um computador. A segunda coluna possui a informação Nome, que o nome do dispositivo de memória ou de entrada e saída. E assim, essas duas informações formam a primeira linha da tabela.

A segunda linha da tabela possui a informação Memória Volátil na primeira coluna e a informação Registradores da CPU na segunda coluna, indicando que os registradores que existem dentro do processador são um tipo de memória volátil.

A terceira linha da tabela possui a informação Memória Volátil na primeira coluna e a informação Cache L1 na segunda coluna, indicando que a memória cache L1 é um tipo de memória volátil.

A quarta linha da tabela possui a informação Memória Volátil na primeira coluna e a informação Cache L2 na segunda coluna, indicando que a memória cache L2 é um tipo de memória volátil.

A quinta linha da tabela possui a informação Memória Volátil na primeira coluna e a informação RAM na segunda coluna, indicando que a memória principal RAM é um tipo de memória volátil.

A sexta linha da tabela possui a informação Memória Volátil na primeira coluna e a informação Memória Virtual na segunda coluna, indicando que a memória virtual é um tipo de memória volátil.

A sétima linha da tabela possui a informação Memória Não-Volátil na primeira coluna e a informação ROM / BIOS na segunda coluna, indicando que a memória a BIOS do seu computador é um tipo de memória ROM não volátil.

A oitava linha da tabela possui a informação Memória Não-Volátil na primeira coluna e a informação Disco Rígido na segunda coluna, indicando que a disco rígido HD do seu computador é um tipo de memória não volátil.

A nona linha da tabela possui a informação Memória Não-Volátil na primeira coluna e a informação CD / DVD na segunda coluna, indicando que mídias de CD / DVD do seu computador são um tipo de memória não volátil.

A décima linha da tabela possui a informação Memória Não-Volátil na primeira coluna e a informação Dispositivos portáteis USB na segunda coluna, indicando que os dispositivos portáveis como Pendrive, hd externo, etc do seu computador são um tipo de memória não volátil.

A décima primeira linha da tabela possui a informação Dispositivo de Entrada na primeira coluna e a informação Teclado na segunda coluna, indicando que o teclado do seu computador é um dispositivo de entrada de dados.

A décima segunda linha da tabela possui a informação Dispositivo de Entrada na primeira coluna e a informação Apontador (Mouse) na segunda coluna, indicando que o mouse do seu computador é um dispositivo de entrada de dados.

A décima primeira linha da tabela possui a informação Dispositivo de Entrada na primeira coluna e a informação Scanner, câmera, microfone, etc. na segunda coluna, indicando que dispositivos como scanner, câmeras e microfones do seu computador são dispositivos de entrada de dados.

Figura 1 - Hierarquia dos tipos de memória e dispositivos de E/S, no início do quadro, os registradores e as memórias cache que pertencem à CPU.Fonte: Elaborado pelo autor, 2018.

Descrição: A figura mostra um homem segurando uma prancheta, que é um objeto retângulo usado para prender folhas de papel e permitir que você faça anotações nesse papeis sem a necessidade de apoiar em uma mesa. A figura também mostra diversos objetos espalhados sobre um mesa, como um agenda na cor preta com folhas brancas, localizada à esquerda da mesa, um régua bege e três lápis de escrever na parte da frente da mesa, um compasso ao lado da régua, porém localizado mais no meio da mesa, uma calculadora no canto frontal direito da mesa, um disquete na direta da mesa mais ao centro e um esquadro na direta no fundo da mesa. Todos esses objetos estão sobre folhas de papel com plantas de desenho de uma construção como um prédio, uma casa etc. O objetivo dessa imagem é mostrar o disquete, que são objetos quadrados na cor preta com uma parte metálica em cima. Esse dispositivo era usado para armazenar dados do seu computador.

Figura 2 - Itens que estão cada vez mais caminhando para a obsolescência, ou já nem são mais utilizados, como o disco flexível de 3.1/2”, à direita.Fonte: Moon Light PhotoStudio, Shutterstock, 2018.

Dessa forma, quando um processo é escalonado para a execução, uma parte significativa é carregada para a memória RAM. Todavia, vimos que a memória RAM tem um atraso de 10 ns para ser acessada. Isso seria como se um avião tivesse que parar nos semáforos. Assim, para minimizar a perda de tempo e o poder de processamento da CPU, quando um processo é lido, parte (ou todo) do fragmento lido é copiado (de uma vez) da memória RAM para a cache.

**4.1.1 Conceitos e funcionalidade**

Quando a CPU for executar a próxima instrução do processo, ela busca essa instrução da memória cache (que, como vimos, é muito mais rápida do que a RAM). Se a próxima instrução não estiver na cache L1, verifica-se se está na cache L2, se não estiver, então ela é recuperada na RAM, e não apenas ela, mas um outro fragmento do processo, que é armazenado na L1 inicialmente. Mas quando há algo na cache que não está sendo utilizado (código ou dados), ele sai da L1 e fica na L2. Se também não for usado por muito tempo na L2, então é descartado.

Portanto, a função das memórias caches é manter por perto e à disposição da CPU os códigos e dados que estiverem sendo muito utilizados. Assim, evita-se um custoso acesso à memória RAM e, pior ainda, à alguma memória secundária.

Com relação à velocidade da memória secundária, enquanto as memórias caches têm acesso na casa dos 2 ns e o acesso à memória RAM em torno de 10 ns, o acesso ao disco rígido ocorre em torno de 10 ms (milissegundos) (DEITEL; DEITEL, 2011).

O próximo Quadro apresenta uma classificação dos tipos de memória de um típico microcomputador.

Tabela

Descrição gerada automaticamente

Descrição: A figura mostra uma Tabela de três colunas e 5 linhas na forma de uma retângulo grande. A primeira coluna da tabela possui a informação Tempo de acesso, que indica o tempo gasto para acessar dados em um tipo de memória. A segunda coluna possui a informação Tipo de uso, que indica o tipo de memória. E a terceira coluna possui a informação Capacidade Típica, que indica a quantidade de dados mais comuns desse tipo de memória. E assim, essas informações formam a primeira linha da tabela.

A segunda linha da tabela possui a informação 1 nanosegundo na primeira coluna, a informação Registrador na segunda coluna e a informação maior que 1 kilobyte na terceira coluna, indicando que o tempo de acesso da memória do tipo registradores é de 1 nanosegundo e a capacidade típica é maior do que 1 kilobytes.

A terceira linha da tabela possui a informação 2 nanosegundos na primeira coluna, a informação Cache na segunda coluna e a informação maior que aproximadamente (símbolo de til) 4 megabyte na terceira coluna, indicando que o tempo de acesso da memória do tipo cache é de 2 nanosegundos e a capacidade típica é aproximadamente 4 megabyte, atualmente podendo chegar a 12 megabytes.

A quarta linha da tabela possui a informação 10 nanosegundos na primeira coluna, a informação RAM na segunda coluna e a informação entre um e oito gigabytes na terceira coluna, indicando que o tempo de acesso da memória do tipo RAM é de 10 nanosegundos e a capacidade típica está entre um a oito gigabyte, atualmente podendo chegar a mais de 64 gigabytes.

A quinta linha da tabela possui a informação 10 milissegundos na primeira coluna, a informação Disco Rígido na segunda coluna e a informação entre um e 4 terabytes na terceira coluna, indicando que o tempo de acesso do disco rígido é de 10 milissegundos e a capacidade típica está entre um a quatro terabytes.

Figura 3 - As diferenças de tempos entre os dispositivos de armazenamento de dados do tipo memória.Fonte: Adaptado de TANENBAUM; BOS, 2016.

Qual o impacto de um microcomputador não ter uma memória cache? Seria a diminuição drástica da velocidade de processamento do conjunto, algo em torno de 10 vezes mais lento (DEITEL; DEITEL, 2011), tendo em vista que a diferença entre a velocidade da RAM e das memórias cache é cerca de 10 vezes.

Nos primeiros computadores, a programação, da forma que conhecemos e entendemos, não existia. O equipamento era produzido para um fim específico, portanto era altamente especializado em uma única tarefa. Hoje, nossos microcomputadores realizam uma infinidade de tarefas sendo programáveis. A preocupação em produzir um equipamento capaz de ser programado remonta ao ano de 1945, no contexto do projeto EDVAC. Para conhecer essa história, acesse o relatório confidencial do projeto (ECKERT; MAUCHLY, 2006) disponível em: < http://archive.computerhistory.org/resources/text/Knuth\_Don\_X4100/PDF\_index/k-8-pdf/k-8-u2736-Report-EDVAC.pdf>.

Então, por que não usamos apenas memória cache ao invés de memória RAM? Essa é uma pergunta comum e a resposta é simples: o preço das memórias caches em relação à RAM acompanha a diferença de velocidade.

Considere agora o impacto que há quando o sistema operacional opera com características de paralelismo. Como sabemos, um dos grandes trabalhos realizados pelo SO é relativo à troca de contexto entre os processos. Os processos são trocados em função das decisões do algoritmo de escalonamento que estiver sendo usado pelo sistema operacional. Em sistemas com preempção por interrupções e compartilhamento de tempo, o nível com que o sistema operacional troca de processo em execução é muito alto.

Considere agora um volume alto de trocas de contexto que requerem que os dados e o código de programa dos processos sejam constantemente carregados para execução, salvos na troca de contexto e novamente carregados para continuar a execução.

Sem o uso e apoio das memórias cache, o desempenho seria muito afetado e certamente não teríamos os equipamentos rápidos como os que nós costumamos ver hoje em dia.

Ainda assim, até o momento, não conseguimos fugir da necessidade de utilizar memórias secundárias em disco rígido, que são absurdamente lentas se compararmos com a velocidade da RAM e das memórias cache.

Como você deve saber, o computador não surgiu da forma como o vemos hoje, houve um enorme processo de evolução que culminou nos modernos equipamentos que estamos acostumados a utilizar. Grande parte dessa evolução tem relação com o processo de fabricação, que também foi modernizado. Você já viu como é feito uma unidade de processamento central? Vale a pena conferir o Processo de fabricação do Intel Core i5 (INTEL, 2014) em: <https://www.youtube.com/watch?v=2W6kCXwSZnk&feature=youtu.be>.

Se houvesse um novo tipo de dispositivo de armazenamento secundário que pudesse ser usado em substituição aos discos rígidos e que apresentasse uma velocidade semelhante à das memórias RAM e cache, estaríamos em um outro patamar de desempenho da computação.

**4.1.2 Segmentação**

O conceito de paginação com o qual trabalhamos até este ponto consiste em uma abstração da memória virtual, dividida em frações e fornecidas aos processos. Todavia, essa estratégia pode representar dificuldades quando para um processo foi alocado uma página de memória virtual e durante o processamento passa a ser necessária mais memória. Para resolver essa e outras situações, uma nova abstração foi introduzida na qual a memória é segmentada e cada segmento pode possuir espaços de endereçamentos completamente independentes. Dessa forma, um segmento pode ter tamanho arbitrário, isto é, de zero até algum valor máximo permitido.

Usando-se essa abstração, as porções de memórias alocadas aos processos podem crescer ou diminuir de tamanho em função das solicitações dos processos.

Segmentos podem ser criados simplesmente para alocar estruturas de dados, tais como pilhas, variáveis, listas etc. de forma que na conversão dos códigos-fontes dos aplicativos, essas estruturas de dados sejam criadas de forma que possam ser suportadas em diferentes segmentos de memória.

A seguir, vamos conhecer o que é e como funcionam o gerenciamento de entrada e saída.

**4.2 Gerenciamento de entrada e saída**

Para que os processos sejam executados, quase sempre é preciso que eles tenham acesso a recursos, necessários para a escrita ou leitura de dados. A leitura de dados está relacionada ao conceito de entrada de dados, enquanto a escrita fica associada à saída de dados. São verdadeiros desafios ao sistema operacional e a forma de gestão compete ao gerenciamento de entrada e saída.

Com o aperfeiçoamento dos microcomputadores, houve também uma evolução paralela de certos tipos de dispositivos, como os de entrada e saída de dados para registro em massa. Por registro em massa estamos nos referindo a grandes quantidades de dados. A Tabela a seguir apresenta alguns desses dispositivos usados desde o surgimento do microcomputador e do sistema operacional.

A capacidade de armazenamento mudou muito em relação aos quesitos velocidade, confiabilidade e, principalmente, quantidade de dados que podem ser guardados em tais dispositivos. De fato, muito do que se faz atualmente seria impossível se não houvesse equipamentos que suportassem a enorme quantidade de dados que é necessária, por exemplo, para a simples existência de um filme guardado no microcomputador (DEITEL; DEITEL, 2011).

**4.2.1 Interrupção, acesso direto à memória**

Os diversos dispositivos que permitem a realização da entrada e saída de dados são controlados pelo sistema operacional, mas esse controle é parcial, pois alguns operam de maneira completamente separada do microcomputador. No caso desse tipo de equipamento, é necessário algum mecanismo que permita uma forte relação entre o equipamento de entrada e saída, o sistema operacional e o hardware.

Esse trio deve ser sincronizado, é preciso que o equipamento externo possa comunicar a CPU que ele precisa transferir dados, que ele está pronto para ser usado, ou que acabou de ocorrer alguma situação que merece atenção do sistema operacional.

Uma solução é permitir e disponibilizar mecanismos sólidos para que, quando um dispositivo precise de atenção do sistema operacional, ele possa lançar mão desse recurso.

Nos microcomputadores modernos, este mecanismo é conhecido por interrupção, sendo, de fato, uma requisição de interrupção ou interruption request (IRQ) (TANENBAUM; BOS, 2016). Quando uma IRQ ocorre é sinal de que algum fato relevante aconteceu em alguma parte do hardware e precisa de atenção. Ignorar uma IRQ, isto é, não dar o tratamento adequado para uma solicitação de interrupção, pode levar o equipamento a uma perda de dados ou até o travamento total.

Os discos rígidos também utilizam as interrupções sempre que concluem uma operação de escrita ou leitura e para informar ao sistema operacional que a tarefa foi finalizada. Assim, quando um processo precisa de dados que estão em um disco rígido, ele solicita ao sistema operacional que coloca o processo no estado de esperando. Depois disso, o sistema operacional providencia a solicitação de leitura de dados ao disco rígido. Enquanto o disco rígido providencia os dados solicitados pelo sistema operacional, este estará livre para escalonar outro processo. Quando, enfim, os dados estiverem disponíveis, o disco rígido gerará uma IRQ que notificará o sistema operacional, que então, obterá os dados e os disponibilizará para o processo que havia solicitado, ocasião na qual terá seu estado alterado de esperando para pronto e será escalonado pelo sistema operacional de acordo com o algoritmo que estiver em uso.

Isso que acabamos de apresentar é parte da comunicação e notificação entre o equipamento de entrada e saída e o sistema operacional. A outra parte refere-se propriamente à transferência dos dados.

Se uma CPU precisasse ficar totalmente envolvida na transferência de dados, digamos entre um disco rígido e a memória, haveria uma enorme sobrecarga de processamento e o tempo de transferência de dados iria “roubar” o tempo de processamento de fato. Na verdade, os primeiros microcomputadores funcionavam dessa maneira e durante as leituras e escritas de grandes volumes de dados, a máquina, literalmente, não fazia outra coisa e o usuário tinha a impressão de travamento.

Descrição: O quadro é formado de 6 colunas e 6 linhas. A primeira coluna da tabela possui a informação Nome do dispositivo. A segunda coluna possui a informação Período de uso aproximado. E a terceira coluna possui a informação Capacidade predominante. A quarta coluna possui a informação Custo. A quinta coluna possui a informação tempo de acesso. A sexta coluna possui a informação Confiabilidade, que significa a confiabilidade no armazenamento dos dados. E assim, essas informações formam a primeira linha da tabela.

A segunda linha da tabela possui na primeira coluna a informação “Fita K7”, na segunda coluna a informação “Década de 1980 e parte da década de 1990”, na terceira coluna a informação “Poucos kilobytes”, na quarta coluna “Preços mínimos”, na quinta coluna a informação “Dependia da velocidade de gravação. Lento”, na sexta coluna a informação “Pouco confiável”.

A terceira linha da tabela possui na primeira coluna a informação “Disco 5.1/4”, na segunda coluna a informação “Década de 1990”, na terceira coluna a informação “160 ou 360 kilobytes”, na quarta coluna “Preços mínimos”, na quinta coluna a informação “Cerca de 30 kilobytes por segundo”, na sexta coluna a informação “O disco era facilmente danificado”.

A quarta linha da tabela possui na primeira coluna a informação “Disco 3.1/2”, na segunda coluna a informação “Década de 1990 e posterior”, na terceira coluna a informação “1.44 megabytes”, na quarta coluna “Preços mínimos”, na quinta coluna a informação “Cerca de 60 kilobytes por segundo”, na sexta coluna a informação “Vários anos se manuseado adequadamente”.

A quinta linha da tabela possui na primeira coluna a informação “Disco Compacto”, na segunda coluna a informação “Década de 1990 e atualmente”, na terceira coluna a informação “Variável, em torno de 700 Megabytes”, na quarta coluna “Preços mínimos”, na quinta coluna a informação “Taxas de transferências variáveis para escrita e leitura. Medianamente rápido”, na sexta coluna a informação “Teoricamente vários anos, mas a mídia era muito frágil e tinha que ser manuseada com muito cuidado”.

A sexta linha da tabela possui na primeira coluna a informação “Disco rígido”, na segunda coluna a informação “Década de 1990 e atualmente”, na terceira coluna a informação “1 GB – 1990 200GB – 1990 1000GB - 2007”, na quarta coluna “Variáveis dependendo da capacidade, podendo chegar a milhares de reais”, na quinta coluna a informação “Taxas variando de 1,7 até cerca de 30 megabytes por segundo. Rápido”, na sexta coluna a informação “Durável, e por permanecer dentro do microcomputador bem protegido”.

Tabela

Descrição gerada automaticamente

Quadro 1 - Evolução dos principais dispositivos de armazenamento de dados usados nos microcomputadores, destacando-se as respectivas capacidades de armazenamento de dados.Fonte: Elaborado pelo autor, 2018.

Quando tratamos do escalonador de tarefas e da próxima instrução a ser executada pela CPU, estamos nos referindo, de fato, às instruções que são “compreendidas” pelo processador. No processador, as instruções são representadas por códigos numéricos e há um número bem pequeno delas. Para entender o assunto, a recomendação é o livro “Programming Ground Up” (BARTLETT, 2003) sobre a linguagem assembly, disponível em: <http://nongnu.askapache.com/pgubook/ProgrammingGroundUp-0-9.pdf>.

Nos equipamentos modernos foi introduzido o conceito de acesso direto à memória, ou DMA, que permite que tais dispositivos utilizem um canal exclusivo para a transferência de dados, desocupando parcialmente a CPU. O resultado é que as transferências de dados quase não ocupam tempo de processamento, portanto o microcomputador, como um todo, apresentará um desempenho idêntico, mesmo durante a execução de processos que façam intenso uso de dados.

**4.2.2 Deadlock**

Você já tem conhecimento das condições de corrida que ocorrem quando processos diferentes disputam o mesmo tipo de recurso e das estratégias para resolver os problemas de acesso às regiões críticas dos processos, relacionados às condições de corrida. Todavia, essa não é a única situação em que a competição por recursos, por parte dos processos, pode resultar em problemas.

Vamos supor que X necessita dos recursos A e B, dos quais ele já possui privilégio de uso de um dos recursos e, neste caso, digamos do A. Como o processo X precisa de A e B, e ele já possui A, resta aguardar até que o recurso B seja liberado e alocado pelo sistema operacional a X. Assim, X poderá realizar o processamento pretendido. Mas considere a situação particular na qual o processo Y já possui o acesso ao recurso B e está dependendo do recurso A que está bloqueado.

O resumo da situação é: X depende que Y libere A, enquanto Y somente liberará B quando X liberar A. Isso leva os processos a uma situação bloqueante sem solução. Ambos estão condenados a nunca serem concluídos.

Diagrama

Descrição gerada automaticamente

Descrição: A figura oito retângulos nas cores cinza, que representa ações dos processos e alaranjado, que representam os processos e o recursos (regiões críticas) utilizados pelos processos.

Figura 4 - Uma generalização da situação de deadlock em que há mútua dependência paralisante, na qual dois processos detêm e competem por recursos mútuos.Fonte: Elaborado pelo autor, 2018.

Os projetistas dos sistemas operacionais têm algumas opções para tentar minimizar a ocorrência desse tipo de situação. A primeira é não fazer nada. Se isso acontecer, os processos terão que lidar com o fato, assim eles devem ter um tempo de espera, caso não tenham sucesso em bloquear todos os recursos, e liberar aqueles que já foram bloqueados e se encerraram, ou então recomeçarem novamente. Esta técnica é chamada de Técnica de Avestruz.

Outra opção consiste em forçar a preempção no uso dos recursos, dessa forma, arbitrariamente, o sistema operacional pode suspender o privilégio de acesso a um recurso por um determinado processo e entregar o controle para outro. Posteriormente, o controle voltará ao processo que inicialmente o estava utilizando. Esta técnica é conhecida como Técnica de Preempção. Existem outras maneiras que envolvem a eliminação de um dos processos bloqueantes, ou que os processos informem antecipadamente quais recursos (e em que ordem) serão necessários para a realização de seus processamentos.

O sistema operacional Linux, por exemplo, não trata os problemas (isto é, não toma nenhuma providência e conhecimento) que ocorrem no espaço do usuário. Ele considera isso um problema a ser resolvido pelos processos. Já no espaço do núcleo, a ocorrência de deadlocks se dá eliminando as condições nas quais o problema possa ocorrer e se programando corretamente para que tais situações sejam anuladas.

Muito bem, continuando com nosso estudo, vamos para o próximo tópico: gerenciamento de arquivo.

**4.3 Gerenciamento de arquivo**

O gerenciamento de arquivo é uma responsabilidade do sistema operacional que está intimamente relacionada aos processos de entrada e saída (NEMETH; SNYDER; HEIN, 2007).

Um arquivo, como já sabemos, é uma abstração para uma região da memória secundária (externa em discos rígidos, por exemplo), nas quais os dados do usuário são guardados de maneira a não serem perdidos quando o equipamento for desligado. Essa discussão também é totalmente aplicada aos discos flexíveis, pendrivers e outros tipos de dispositivos para a guarda de dados.

Ocorre que os arquivos não podem simplesmente serem escritos no dispositivo, é preciso que haja um planejamento, um cuidado e uma estruturação de tal forma que seja possível a recuperação de qualquer arquivo em qualquer tempo pelo usuário e/ou pelos processos do usuário.

Para que essa organização seja possível, foram criados certos protocolos que governam a forma como os dados, os arquivos e as pastas de arquivos são criados, acessados e mantidos nos discos rígidos. Este protocolo recebeu o nome de sistema de arquivo.

Descrição: A figura mostra uma Tabela de três colunas e 13 linhas na forma de uma retângulo grande. A primeira coluna da tabela possui a informação Sistema operacional que o suporta. A segunda coluna possui a informação Sistema de arquivo. E a terceira coluna possui a informação Principal uso.

Tabela

Descrição gerada automaticamente

O que você precisa compreender é que todos sistemas de arquivos têm em comum as seguintes características: suportam a criação de pastas (diretórios); de arquivos; a exclusão de arquivos e pastas, bem como suas renomeações, completando as funcionalidades básica do sistema de arquivo, relacionadas às operações de leitura, escrita e execução de arquivo.

**4.3.1 Tipos, sistemas e acesso**

Alguns sistemas de arquivos suportam a recuperação de arquivos e/ou pastas apagadas (o que nem sempre é possível), outros não suportam (mesmo que parcialmente) esse tipo de operação e há aqueles que suportam a encriptação dos dados, de forma que apenas quem possuir chaves e senhas possa ter acesso aos dados.

Os sistemas de arquivos mais modernos suportam também a vinculação entre as pastas e os arquivos e seus proprietários, como o controle de tipo de acesso que pode ser realizado sobre tais arquivos, como de leitura, escrita etc. (NEMETH; SNYDER; HEIN, 2007).

Como você deve ter percebido, a memória tem um papel essencial nos microcomputadores atuais. Em destaque, temos os discos rígidos que são amplamente utilizados e existem em diversas capacidades e fabricantes. Uma delas é a Seagate, companhia internacional com vários produtos nessa área. E você sabe como funciona um disco rígido? A empresa publicou um vídeo explicativo sobre o tema, disponível em: <https://youtu.be/lpYfep68xnA>. (SEAGATE TECNOLOGY, 2016).

Vimos há diferentes tipos de sistemas de arquivos, alguns são compatíveis com determinados SOs e outros não. Isso em alguns momentos torna-se um problema para o usuário, que pode não ter acesso aos seus dados devido a eles terem sido escritos utilizando outro tipo de sistema de arquivo.

Há casos em que o dispositivo usado no acesso aos dados permite apenas a escrita, ou então apenas a leitura. Também existem particularidades que estão associadas à mídia usada em tais equipamentos. Um desses casos ocorre quando em um dispositivo de leitor de CD seja usada uma mídia que permite apenas a leitura. Uma tentativa de escrita resultaria em um erro, pois não seria uma operação permitida para aquele dispositivo.

**4.3.2 Estrutura de diretório**

Uma diretório (ou pasta) é um conceito que permite agrupar um conjunto de arquivos (e outras pastas) sob um mesmo nome. Essa abstração é utilizada para permitir uma melhor organização nos dados.

Assim, o usuário pode criar estruturas que sejam convenientes para agrupar arquivos de uma mesma natureza, tais como documentos, fotos, músicas etc.

Na implementação de uma estrutura de diretório são necessárias algumas informações para a localização no disco. Em alguns casos, a depender do tipo de sistema de arquivo que esteja em uso, pode ser utilizada uma referência ao disco inteiro (para sistemas de alocação direta), ou um número que indique uma determinada posição do diretório no disco (número do primeiro bloco) para estruturas encadeadas, por exemplo.

Esses dados precisam ser armazenados em um ponto no disco de tal forma que exista uma convenção sobre como localizá-los (MACHADO; MAIA, 2013). Geralmente, utiliza-se uma parte do disco, previamente conveniada, a qual tem uma função de tabela de alocação. De fato, o conceito dessa tabela é amplamente conhecido como File Allocation Table (FAT), sigla que, inclusive, tem forte relação com os nomes dos sistemas de arquivos da Microsoft, mas como conceito está relacionada a todos os sistemas de arquivos. Essa tabela de alocação precisará ser preenchida com várias informações, tais como o tamanho do diretório (ou do arquivo), seu nome, dados do proprietário, permissões de acesso etc. Já que estamos tratando de criação e acesso a diretórios, arquivos e pastas, uma pergunta é válida: somente o usuário tem acesso a esses recursos? É um questionamento relacionado aos mecanismos de segurança, nosso tópico a seguir.

**4.4 Mecanismos de segurança**

As permissões de acesso e nomes de usuários dos diretórios e arquivos são exemplos de mecanismos de segurança que são implementados pelo sistema operacional. Utilizando desses dados simples, dono e permissões, já é possível prevenir uma série de problemas, como um usuário copiar ou ler dados de outros ou ainda modificar tais dados sem o consentimento do dono. Mas há casos em que você deseja que outros usuários possam ler ou até mesmo alterar seus arquivos, para isso as permissões de acesso serão de grande ajuda. Todavia os problemas de segurança são mais complexos e, portanto, devem existir mais mecanismos de proteção.

Uma grande parte da segurança de dados que existe hoje é vinculada ao sistema operacional. Se ele falha, várias outras ações de segurança deixarão de ser efetivas. No entanto, é muito difícil para o sistema operacional proteger seus dados sem a ajuda do hardware. Uma das ações básicas para proteger os dados do usuário é garantir que um aplicativo (processo) não tenha acesso irrestrito a toda memória real do microcomputador. Isso é necessário para impedir que um processo modifique dados de outro processo. No passado, as CPUs não tinham um chamado “modo protegido”, agora, exceto alguns microprocessadores e microcontroladores, esse modo protegido já existe.

A memória do microcomputador, bem como os dispositivos que permitem que ele se comunique com outros equipamentos, também deve ser alvo de preocupações relativas à segurança. Muitas são as técnicas utilizadas para explorar brechas no gerenciamento de segurança, ou até mesmo funcionalidades essenciais de alguns tipos de aplicativos. A exemplo, podemos citar um aplicativo para a publicação de páginas na internet, denominado servidor web. Um servidor web necessita, essencialmente, comunicar-se através da internet e, para tanto, precisa de acesso à rede de dados do microcomputador, que requisitará esse tipo de recurso ao sistema operacional.

Ao permitir que o servidor web faça uso de comunicação com a internet, também passa a ser possível que usuários externos (desconhecidos ao sistema operacional) possam solicitar recursos. Esses recursos, em tese, deveriam se referir às páginas html (dentre outras) que seriam disponibilizadas pelo servidor web, todavia, caso existam falhas no design desse aplicativo servidor, ele pode ser utilizado como recurso para que sejam obtidos documentos que não deveriam ser acessados por terceiros, como, por exemplo, a relação de usuários registrados no sistema operacional.

A segurança de dados nasce no sistema operacional e se estende até o comportamento do usuário. Um caso que recebeu muita atenção da mídia foi o do Wikileaks, cujo fundador, Julian Assange, está no centro da divulgação de dados, por exemplo, do partido Republicano dos Estados Unidos. Atualmente, Assange está refugiado em uma embaixada sem tratado de extradição. O caso é notório pela forma como os dados iniciais (cerca de 92 mil documentos sobre operações militares de diversos países) foram obtidos. Há hipótese de que foram a partir de um sistema militar de correio eletrônico usado pelo exército. Para descobrir quem esteve envolvido no processo de cópia não autorizada dos documentos foram utilizados os registros de históricos (logs) do sistema operacional, que permitiu visualizar os usuários que fizeram uso de determinados equipamentos no período em que ocorreu o vazamento. Isso reforça que é sempre importante manter o controle de acesso, com o uso exclusivo de credenciais, nunca cedendo a terceiros, como também de uma política de guarda e verificação dos históricos do sistema operacional.

Percebe-se que o sistema de arquivo é um ponto de vulnerabilidade em qualquer sistema operacional, pois mesmo que existam várias ações para tornar o SO seguro, se as credenciais dos usuários forem roubadas, de nada valerá qualquer ação tomada anteriormente. Uma maneira de tornar um sistema operacional mais seguro é melhorar o seu ambiente de funcionamento.

**4.4.1 Ambiente, controle de acesso aos recursos**

Suponha que você coloque o seu sistema operacional para funcionar como um grande processo dentro de outro sistema operacional. Isso seria possível? Tornaria seu sistema um pouco mais seguro?

A resposta para as duas questões é: sim. Este conceito, na verdade, é conhecido como virtualização e pode ser utilizado para dar mais segurança a um sistema operacional, ou ainda para que um microcomputador potente possa parecer ser mais de um.

Pela virtualização é possível que em um mesmo microcomputador você tenha executado vários sistemas operacionais ao mesmo tempo. Isso é uma vantagem quando, por exemplo, você precisa ter duas máquinas com sistemas operacionais diferentes para realizar tarefas que dependem de aplicativos que funcionam somente em determinados sistemas.

Mas, retornando ao foco da segurança, através da virtualização, a máquina real, hospedeira, torna-se um pouco mais afastada dos usuários, podendo minimizar os riscos de perda de dados e criando máquinas virtuais para demandar necessidades específicas. Suponha que para o servidor web, nos criássemos uma máquina virtual e lá incluíssemos os documentos html que desejamos publicar. Caso essa máquina fosse alvo de alguma violação de segurança, no máximo, poderiam obter os documentos html (que já estão expostos) e a máquina real ainda estaria protegida.

Com a virtualização, pode-se controlar o acesso a certos recursos quando forem ofertados através de redes de dados. Mesmo que não se utilize o conceito de virtualização, é possível também tornar o ambiente do sistema operacional mais seguro. Para isso, os dados que se deseja proteger podem ser criptografados utilizando-se esquemas bem complexos. Nesses processos, os dados criptografados somente poderão ser lidos se o usuário possuir uma senha especial de acesso.

**4.4.2 Noção básica de criptografia**

A criptografia consiste em transformar os dados em outros, de forma que essa transformação seja controlada e reversível. Os dados transformados não farão sentido e nem poderão ser associados a algum processo específico. A reversão para os originais somente pode ocorrer quando o usuário possuir o algoritmo e uma chave de transformação. Daqui em diante, vamos usar o termo criptografia para nos referirmos à essa transformação.

A criptografia pode ser, em geral, de dois tipos: simétrica e assimétrica. A simétrica é aquela na qual existe apenas uma única chave de decriptação, sem a qual os dados estarão perdidos. Já a criptografia por chave assimétrica requer duas senhas, uma que é pública e todos podem conhecer, e outra que é secreta.

Diagrama

Descrição gerada automaticamente

Descrição: A figura mostra a sequência seguida para a criptografia de um texto. A figura é dividida em cinco partes.

Começando da esquerda para a direita. A primeira parte é o símbolo P com o rótulo Entrada de texto puro e uma seta horizontal apontando da esquerda para a direita.