Centro Universitário da FEI

CC6270 – Sistemas Operacionais

Visão Inteligente na Cotação do Dólar

Inteligência de negócio aplicada à cadeia de valor empresarial.

Gabriel Spinardi	22.215.065-8
Kaike Rodrigues Zuanetti	22.118.116-7
João Vitor Couto	22.118.022-7

São Bernardo do Campo
Dezembro de 2020

Índice

1. Resumo

Em construção. Só será possível fazer um resumo após todos os resultados e conclusões desse projeto.

2. Palavras-chave

Inteligência Empresarial, *Business Intelligence,* Dólar, software, aplicação, inteligência.

3. Introdução

A inteligência de negócios ou inteligência de negócios pode ser descrita como um processo baseado em tecnologia que analisa dados e fornece informações acionáveis para ajudar executivos, gerentes e outros usuários finais da empresa a tomar decisões de negócios inteligentes. Portanto, os recursos relacionados a essa inteligência incluem várias ferramentas, aplicativos e métodos que permitem às organizações coletar dados de sistemas internos e fontes externas, preparar-se para análises e desenvolver consultas relacionadas. As ferramentas de inteligência de negócios são capazes de acessar e verificar conjuntos de dados cujos resultados são exibidos em relatórios de análise, resumos, painéis gráficos e mapas para fornecer aos usuários informações detalhadas sobre o status dos negócios.

Além de ser uma tecnologia para a busca ampla de soluções objetivas, também pode ser utilizada para auxiliar na tomada de decisões em diversos negócios, desde a operação à estratégia. As decisões operacionais básicas incluem localização e preço do produto. As decisões de estratégia de negócios cobrem as mais amplas prioridades, objetivos e direções.

Em todos os casos, essa inteligência de negócios será mais eficaz se você combinar dados de mercado de operações externas da empresa com dados de fontes internas de negócios da empresa (como dados financeiros ou operacionais). Quando os dados externos e internos são combinados, eles podem fornecer uma imagem mais completa. Na verdade, ele cria "inteligência" que não pode ser exportada de nenhum conjunto de dados.

4. Objetivo do Trabalho

Considerando a abordagem do tema supracitado, este tópico discorrerá sobre base de estudos, requisitos e apresentação de um *software* relacionado à inteligência de negócio para o âmbito empresarial.

4.1. *Software*

Em conformidade com o tema abordado, será criado um *software* capaz de reconhecer possíveis oscilações de valores da moeda norte americana - o Dólar – durante períodos do ano, para que então, faça-se uma estimativa de valorização ou desvalorização desta moeda utilizando inteligência, bem como para auxiliar o cliente final em suas decisões de mercado, seja para investimento de uso pessoal ou empresarial.

4.2. *Hardware*

Para que este projeto seja viável, fora necessário selecionar um sistema de *hardware* capaz de suportar a demanda do *software*, e, assim, será utilizado CPU com 2 núcleos de processamento de 64 bits e 4GB de memória RAM. O responsável pelo desenvolvimento deste projeto avaliou ser o suficiente para suportar o mesmo, após breve mapeamento de possibilidades e orientações[1] presentes em termos de *hardware* atualmente.

4.3. Sistema Operacional

O sistema operacional escolhido é o Ubuntu 20.04.1 LTS, da ©Canonical. Este sistema Linux baseado em Unix nos possibilita, além de uma demanda menor de *hardware* robusto, ótimas soluções de virtualização e consolidação. Isso porque ele consegue ter uma melhor relação entre suas máquinas virtuais e o *hardware* dos servidores, extraindo ao máximo o que os equipamentos têm para oferecer, garantindo um melhor desempenho em relação às demais soluções. Também, é importante citar a robustez de protocolos de segurança envolvendo este sistema, tornando-o altamente propício para esta solução. Sendo assim, considerando colocar em prática este projeto, foi o melhor sistema operacional a ser estudado e por isso fora escolhido.

5. Desenvolvimento

Este projeto é baseado em assuntos diversos que facilitam o bom entendimento do software a ser desenvolvido. No entanto, é necessário entender como essa aplicação irá se comportar dentro do sistema operacional utilizando recursos que se comunicam entre o hardware e o SO, para que se cumpram todas suas funcionalidades sem maiores problemas.

Para tal, abordaremos no desenvolvimento cada tema e setor do software planejado com conceitos a respeito de sua comunicação com o SO e o hardware e, na sequência, a aplicação em termos práticos do tema relacionado.

5.1. Processos

Os processos representam tarefas em execução, mas nem todas as tarefas estão diretamente relacionadas ao aplicativo. Muitos deles são executados em segundo plano e mantêm o sistema em execução - gerenciando a rede, memória, disco, verificação antivírus, etc. Portanto, podemos definir um processo como um software que executa uma determinada ação e pode ser controlado de uma determinada forma, seja um usuário, um aplicativo correspondente ou um sistema operacional. O processo possui muitas características próprias. A estrutura básica consiste em uma imagem do código executável associado ao programa. A memória contém código executável e dados específicos. Ele também descreve os recursos do sistema alocados para o processo, informações sobre atributos de segurança e uma indicação do status atual.

Da criação à conclusão, o processo passa por diferentes estados. No momento da criação, seu status é considerado "novo". Na verdade, torna-se “em execução”, quando depende da ocorrência de um evento, torna-se “espera”, quando não é mais necessário, o processo se “completa”. O sistema operacional coleta todas essas informações por meio de uma estrutura específica chamada PCB (sigla de *Process Control Blocks*, o que em tradução livre seria Blocos de Controle de Processos).

5.2. *Threads*

Uma *thread* é um pequeno programa que atua como um subsistema, é uma forma de um processo se dividir em duas ou mais tarefas. É um termo em inglês para "linha" ou "*thread* de execução". Essas várias tarefas podem ser executadas ao mesmo tempo para que sejam executados mais rapidamente do que os programas em um único bloco de programa, ou podem

realmente ser executados juntos, mas são tão rápidos que parecem funcionar ao mesmo tempo. As várias *threads* que existem no programa podem trocar dados e informações entre si e compartilhar os mesmos recursos do sistema, incluindo o mesmo espaço de memória. Portanto, os usuários podem usar funções do sistema enquanto outros *threads* estão trabalhando e realizando outros cálculos e operações. Parece que o usuário virtual está ocultando trabalho no mesmo computador que você ao mesmo tempo.

Por causa das rápidas mudanças de um *thread* para outro, parece que eles estão executando em paralelo em um *hardware* equipado com apenas uma CPU. Esses sistemas são chamados de *thread* único. Para *hardware* com múltiplas CPUs, os *threads* são realmente criados ao mesmo tempo, o que é chamado de *multithreading*.

A *thread* pode responder por conta própria sem ter que repetir todo o processo, economizando recursos como memória, processamento e fazendo uso de dispositivos de I / O, variáveis e outros métodos. Também pode-se desistir da CPU você mesmo, porque não vê a necessidade de continuar o processamento proposto pela própria CPU ou pelo usuário. Isso é feito usando o método de *thread-yield*. Além das funções mencionadas, um *thread* também pode realizar outras funções, por exemplo, aguardar a sincronização de outro *thread*.

5.2.1. Aplicação

```
while key == True:
    y =  datetime.date(datetime.today).isocalendar()[1]
    compY= y-1
    if compY<=0
        compY=51
    media = Thread(target=Media,args=[cotac,y])
    mediaC = Thread(target=Media,args=[cotac,compY])
    media.start()
    mediaC.start()
    resul=mediaC-media
    if (resul <=0)
        resul = "Queda"
    else
    resul = "Subida"
    Escrever(resul)
        time.sleep(1.5)
```

5.3. Escalonamento de Processos

O escalonamento de processos é a ação de alternar os processos ativos de acordo com regras reconhecidas para que todos os processos tenham a oportunidade de usar a CPU (Unidade Central de Processamento). O escalonador faz parte do SO e é responsável por determinar a relação entre os processos concluídos a serem executados. Existem várias maneiras de implementar o agendamento. Esses métodos devem seguir, como justiça (cada processo obtém uma parcela razoável de seu tempo de CPU), eficiência (para garantir 100% de ocupação do tempo de CPU), minimizar o tempo de resposta aos comandos interativos do usuário e maximizar o número de serviços processados por hora. Esses métodos são:

- **Escalonamento "*Round-Robin*":** Cada processo recebe um intervalo de tempo (*quantum*), e se o processo ainda está rodando no final de seu *quantum* (ou se o processo bloqueia ou termina antes de terminar), a CPU é obtida a partir do processo, e o escalonador escolhe um novo processo a ser executado. O escalonador mantém uma lista de processos executáveis (prontos), e quando o *quantum* termina e o processo não é finalizado, ele será colocado no final da lista. O planejador sempre seleciona o primeiro processo desta lista para executar.
- **Escalonamento com Prioridade:** A prioridade é fornecer tratamentos diferentes para processos diferentes. No momento em que o processo for criado, ele terá prioridade. Quando o planejador deve escolher o processo a ser executado, ele escolherá a prioridade mais alta. Cada vez que o processo for executado, o escalonador irá diminuir sua prioridade, e quando sua prioridade for menor que a de outro processo concluído, ele será interrompido e executará outro processo.
- **Filas Múltiplas:** Devemos determinar a prioridade. Em comparação com os processos de nível inferior, os processos de nível superior são selecionados para execução com mais frequência (eles obterão uma quantidade maior de quantização para processamento). O processo de nível mais alto atual recebe 1 *quantum*, o nível inferior recebe 2, o outro 4 e assim por diante, expresso como uma potência de 2. Quando um processo usa todos os *quantum* que recebe, ele desce de um nível (menos seleções, mas é executado por mais tempo).
- **Menor Serviço Primeiro:** útil para sistemas de processamento em lote porque os usuários geralmente sabem como estimar o tempo de execução de um programa (porque

o programa está constantemente em execução). O planejador seleciona o trabalho com o tempo de execução mais curto entre todos os trabalhos disponíveis. Por exemplo: Existem quatro trabalhos (A, B, C, D). A execução de "A" leva 8 minutos e as demais execuções levam 4 minutos.

- **Escalonamento Dirigido a Política:** Se existem n usuários ativos, então, cada um receberá aproximadamente 1/n do tempo de UCP.
- **Escalonamento em Dois Níveis:** Até agora, consideramos que todos os processos executáveis estão na memória. Em vez disso, precisamos manter parte do processo em disco (porque a memória principal dificilmente contém todos os dados necessários). O problema é que o tempo para ativar o processo no disco é muito maior do que o tempo para ativar o processo na memória. A melhor solução é usar um escalonador de dois níveis. Por esse motivo, um subconjunto de procedimentos executáveis é mantido na memória e outro subconjunto é mantido no disco. O escalonador (nível baixo) é usado apenas para realizar a alternância entre processos na memória (por qualquer um dos métodos acima), enquanto outro escalonador (nível alto) é usado para alterar periodicamente o conjunto de processos na memória (para aqueles processos no disco).

5.3.1. Aplicação

```
def Media(x,y):
    time.sleep(0.1)
    print("A media é : ")
    print(x[y].mean())

def Mediana(x,y):
    time.sleep(0.3)
    print("A mediana é : ")
    print(x[y].median())

def Variancia(x,y):
    time.sleep(0.6)
    print("A variancia é : ")
    print(x[y].var())

def DesPadrao(x,y):
    time.sleep(0.9)
    print("A desvio padrão é : ")
    print(x[y].std())

def CoefCorre(x,y,z):
    return x[y].corr(x[z])

covid = pd.read_csv("a.csv")
key = True
```

5.4. *Scripts*

Uma linguagem de script (s*cripting*) é uma linguagem de programação que oferece suporte a scripts. Um script é um programa escrito para um sistema de tempo de execução especial. O programa pode executar tarefas automaticamente e o operador pode executá-lo uma vez. Linguagens de *script* são frequentemente interpretadas (em vez de compiladas). As primitivas geralmente são tarefas básicas ou chamadas APIs (interfaces de programação de aplicativos), e a linguagem permite que sejam combinadas em programas complexos. Os ambientes que podem ser executados automaticamente por *scripts* incluem aplicativos de *software*, páginas da *web* em um navegador, *shells* de sistema operacional (SO), sistemas incorporados e muitos jogos.

Uma linguagem de *script* pode ser considerada uma linguagem de domínio específico de um ambiente específico; no caso de *script*, é um programa de aplicativo, também chamado de linguagem estendida. As linguagens de script às vezes são chamadas de linguagens de

programação de alto nível porque operam em um alto nível de abstração, ou são chamadas de linguagens de controle, especialmente para a linguagem de controle *mainframe*.

5.4.1. Aplicação

```
#!/bin/bash
python cota.py

notify-send -i face-laugh -t 3000 "$(cat ~/cota.txt)"

exit
```

5.5. Comandos Unix

A plataforma UNIX possui diversos comandos que auxiliam no funcionamento do software.

5.5.1. Aplicação

ls: listar o conteúdo do diretório corrente (ou de um diretório dado).
rm: remover arquivos.
mv: movimentar arquivos.
cp: copiar arquivos.
cat: apresentar o conteúdo de arquivos.
more: visualizar o conteúdo de arquivos (paginado).
ln: criar links (atalhos).

5.6. Gerenciamento de Memória

A necessidade de manter vários programas ativos na memória do sistema cria outro requisito: controlar como esses programas usam essa memória. Portanto, o gerenciamento de memória é o resultado de duas práticas diferentes aplicadas em sistemas de computação:

• Como a memória é vista, isto é, como pode ser utilizada pelos processos existentes neste sistema.

• Como os processos são tratados pelo SO quanto às suas necessidades de uso de memória.

Em sistemas de computador, o armazenamento de dados ocorre em vários níveis. Isso significa que o armazenamento será realizado em diferentes tipos de dispositivos devido aos quatro fatores básicos a seguir:

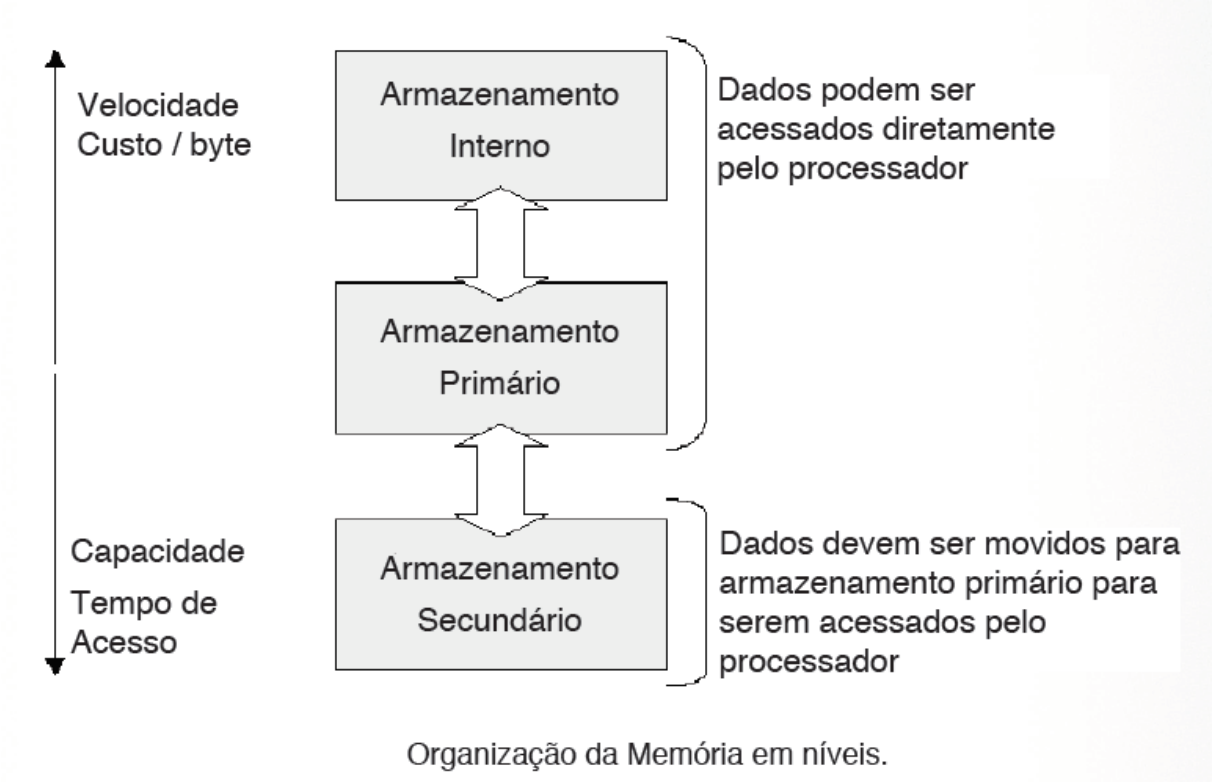
a) **Tempo de acesso.**

b) **Velocidade de operação.**

c) **Custo por unidade de armazenamento.**

d) **Capacidade de armazenamento.**

Portanto, o projetista do sistema operacional determina quanto cada tipo de memória é necessário para tornar o sistema eficiente e economicamente viável. Acontece que quanto maior a velocidade, mais cara a memória e menor a capacidade de armazenamento de dados. A figura a seguir mostra essa hierarquia de organização de memória:



Organização da Memória em níveis.

A memória interna é um local de memória disponível dentro do processador para permitir ou acelerar sua operação. Ele consiste em registros do processador e seu cache interno.

A memória principal é um local da memória interna acessível diretamente pelo processador. Normalmente, eles são CI (Circuito Integrado), como SRAM, DRAM, EPROM, PROM etc.

A memória auxiliar é uma localização de memória externa que não pode ser acessada diretamente pelo processador e deve ser movida para a memória principal antes do uso. Geralmente, eles são dispositivos de armazenamento em massa, como discos rígidos.

Perceba que o armazenamento interno possui a maior velocidade de acesso, ou seja, o menor tempo de acesso. Embora sejam os mais caros, representam os dispositivos de melhor desempenho. Por outro lado, os dispositivos de armazenamento secundário são os que possuem a maior capacidade e a melhor relação custo/*byte*, mas a velocidade é muito mais lenta.

Geralmente, os sistemas de gerenciamento de memória podem ser divididos em duas categorias: sistemas que movem processos (programas) do disco para a memória principal (e vice-versa) e sistemas que não realizam esta operação (trabalhando apenas na memória).

5.6.1. Alocações Particionadas Estática e Dinâmica

Em um sistema multiprograma, a memória principal é dividida em blocos chamados de partições. Inicialmente, embora essas partições sejam de tamanho fixo, não são necessariamente do mesmo tamanho entre si, permitindo diferentes configurações. Este esquema é chamado de alocação de partição estática e tem os seguintes problemas principais:

- Os programas geralmente não podem preencher completamente a partição onde são carregados, desperdiçando espaço.
- Se um programa for maior do que qualquer partição disponível, ele aguardará para acomodá-lo, mesmo se duas ou mais partições adjacentes se somarem para formar o tamanho do programa. Esse tipo de problema (fazendo com que a fragmentação da memória seja bloqueada por outros programas) é chamado de fragmentação.

5.6.2. *Swapping*

Em um sistema em lote, é simples e eficiente organizar a memória em partições fixas. Desde que trabalhos suficientes possam ser mantidos na memória para que a CPU esteja sempre ocupada, não há razão para usar outra organização mais complexa.

Em um sistema de *time-sharing*, a situação é diferente: geralmente há mais usuários mantendo todos os processos (programas) do que a memória, por isso é necessário manter os processos extras no disco. Para executar esses procedimentos, eles devem ser trazidos para a memória principal. O movimento de um processo da memória principal para o disco, e vice-versa, é chamado de *swapping*.

Na alocação de partição estática e dinâmica, o programa permanece na memória principal mesmo enquanto espera por um evento (como uma operação de leitura ou gravação

em um dispositivo periférico) até que a execução termine. Em outras palavras, o programa só reserva a memória principal depois de terminar a execução.

A tecnologia de troca pode ser usada em sistemas multiprograma com tamanhos de partição variáveis. Desta forma, sob certas condições, um programa pode ser movido da memória principal para o disco (*swap out*), e o mesmo programa também pode ser devolvido do disco para a memória principal (*swap in*), como se nada tivesse acontecido.

Para se ter uma compreensão mais profunda do assunto, é importante mencionar que a troca é realizada por meio de uma rotina especial de SO chamada relocator ou switch. A existência do relocador no sistema depende do tipo de gerenciamento de memória fornecido pelo sistema operacional. A seguir está uma descrição simplificada do trabalho realizado pelo pessoal realocado.

De acordo com as instruções do sistema operacional (usado para gerenciar memória e processos), o relocador pode ser solicitado para excluir o conteúdo da área de memória e armazená-lo no disco. O que geralmente acontece é que o relocador copia essa área da memória em um arquivo especial denominado *swap file*. Ao copiar uma área da memória para o disco, a área é marcada como livre, disponibilizando-a para outros processos. Além disso, o conteúdo copiado para a memória também é gravado para que o conteúdo possa ser restaurado.

5.6.3. Memória Virtual

conceito de relocação de memória possibilita o desenvolvimento de um método mais otimizado de uso de memória, denominado memória virtual. O conceito de memória virtual é baseado no endereçamento desconectando os programas do endereço físico da memória principal. Portanto, os programas e suas estruturas de dados não são mais limitados pelo tamanho da memória física disponível.

O termo memória virtual geralmente está associado à capacidade do sistema de lidar com mais memória do que a memória fisicamente disponível. Este conceito apareceu no computador Atlas 1960, que foi fabricado pela Universidade de Manchester (Inglaterra), embora tenha sido mais amplamente utilizado recentemente.

Na verdade, a memória virtual do sistema é um arquivo de troca ou *swap file* gravado no disco rígido. Portanto, a memória total de um sistema com memória virtual é a soma da memória física de tamanho fixo e da memória virtual. O tamanho da memória virtual (chamado de arquivo de paginação) é basicamente definido pelo mínimo dos seguintes:

• Capacidade de endereçamento do processador.

• Capacidade de administração de endereços do SO.

• Capacidade de armazenamento dos dispositivos de armazenamento secundário (unidades de disco).

5.6.4. Espaço de Endereçamento Virtual

Os programas no ambiente de memória virtual não se referem a endereços de memória física (endereços reais), mas apenas a endereços virtuais. Quando a instrução é executada, como o processador acessa apenas o local da memória principal, o endereço virtual é convertido em um endereço físico.

O mecanismo de conversão de endereços virtuais em endereços físicos é chamado de mapeamento. No sistema atual, esse mecanismo é completado pelo *hardware* com a ajuda do SO, e o endereço localizado no espaço de endereço virtual é convertido em um endereço de memória física, pois o programa executado em seu contexto precisa estar localizado no espaço de endereço real para poder ser referenciado ou realizado.

Portanto, o programa não precisa necessariamente ser contínuo na memória real para ser executado. Este mecanismo é responsável por manter a tabela de mapeamento exclusiva de cada processo, associando o endereço virtual do processo à sua localização na memória física.

5.6.5. Aplicação

O gerenciamento de memória utilizado neste projeto foi feito através do Garbage Collector do Python.

5.7. Paginação

Paginação é uma técnica de gerenciamento de memória que usa o conceito de memória virtual visto anteriormente, ou seja, a quantidade de endereçamento é maior que o tamanho real da memória do sistema. Dessa forma, o endereçamento global ou espaço de endereço virtual é dividido em pequenos blocos iguais chamados de páginas virtuais.

Cada página possui um número que a identifica, e a memória física é dividida em blocos iguais com o mesmo tamanho da página virtual, chamados de quadro de página. Os frames da página são identificados por números e correspondem a áreas específicas da memória física. Todos os quadros de página são identificados pelo número zero.

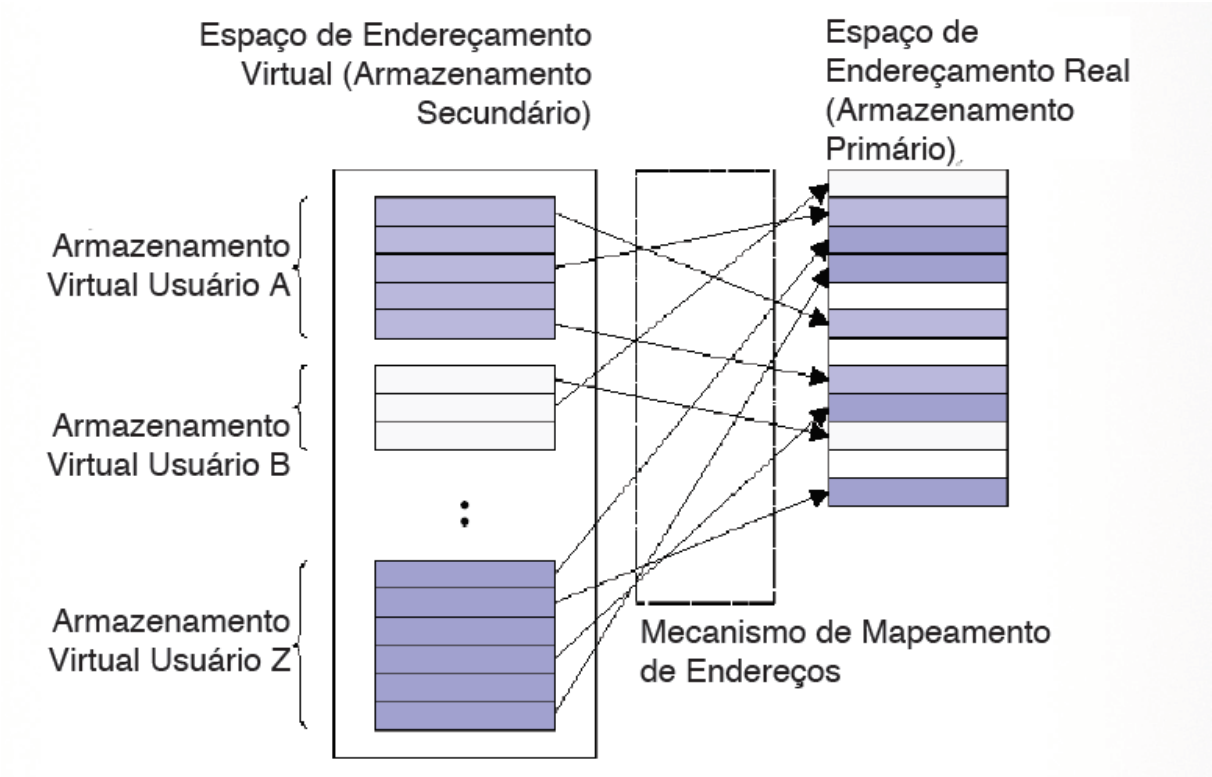
O endereço gerado pelo programa em execução é chamado de endereço virtual e forma o espaço de endereço virtual mencionado anteriormente. Quando o programa executa uma das seguintes instruções da linguagem de programação: MOV REG, 2060. Neste idioma, esta

instrução indica que o conteúdo do endereço de memória 2060 deve ser copiado para o registrador REG.

Portanto, o endereço 2060 gerado pelo programa é um endereço virtual. Em computadores sem um mecanismo de memória virtual, o endereço virtual é colocado diretamente no barramento de memória, o que fará com que o mesmo endereço seja usado para acessar a palavra da memória física (leitura ou gravação). Para memória virtual, o endereço virtual chegará à MMU (Unidade de Gerenciamento de Memória), CI ou uma coleção de CI que mapeia o endereço virtual para o endereço físico.

Para fazer o esquema de divisão proposto para paginação em um SO, o mapeamento deve ser executado para identificar quais páginas virtuais estão na memória física (no quadro da página) e quais estão na memória virtual do sistema (arquivo de troca). Esse mapeamento é feito determinando a tabela de páginas do relacionamento entre a página virtual, o espaço de endereço virtual e o quadro de página do espaço de endereço físico (real).

A figura a seguir demonstra o uso do espaço de endereço virtual e o mapeamento de endereços virtuais para endereços físicos no espaço de endereço de memória real.



No tempo de execução, a página virtual é movida do armazenamento secundário para o armazenamento principal e colocada no quadro da página. Quando o programa está em execução, o sistema operacional associa a página virtual ao programa em execução sem considerar o posicionamento contínuo de partes do mesmo programa.

Ao executar, o MMU consultará a tabela de páginas para converter o endereço virtual em um endereço físico.

5.8. Sistema de Arquivos

Na verdade, um sistema de arquivos é um conjunto de estruturas lógicas, ou seja, arquivos feitos diretamente por meio de um software, que permitem ao sistema operacional acessar e controlar os dados gravados no disco.

Cada sistema operacional lida com sistemas de arquivos diferentes e cada sistema de arquivos tem suas próprias características, como restrições, qualidade, velocidade, gerenciamento de espaço e outras características. O sistema de arquivos define como os *bytes* que constituem o arquivo são armazenados no disco e como o sistema operacional acessa os dados.

Além disso, o desenvolvimento de dispositivos de armazenamento também contribuiu para o surgimento de novos sistemas.

No Windows, o número de sistemas de arquivos é ainda mais limitado. Na era do Windows 95, a Microsoft usava o sistema de arquivos FAT16. Devido às suas limitações, ele foi substituído pelo FAT32 e pelo NTFS muitos anos depois. Esta função ainda está em uso hoje e foi estabelecida devido à sua flexibilidade.

No vasto mundo do Linux, você pode encontrar uma variedade de distribuições e a gama de sistemas de arquivos é muito maior. Os mais usados são EXT3 e EXT4 e ReiserFS. Existem XFS e JFS menos conhecidos.

5.8.1. Métodos de Acesso

1. Sequencial:

- Apenas no final do arquivo novos registros podem ser gravados.
- Exemplo: Fita magnética.

2. Acesso Direto:

- Maior eficiência do que a sequencial;
- Permite ler / escrever registros diretamente em sua localização através do número do registro, que é relativo ao início do arquivo.
- Não há restrição à ordem de leitura ou escrita dos registros, sendo sempre necessário especificar o número do registro.

Só é possível ao usar um arquivo de definição de registro de tamanho fixo.

3. Acesso Direto + Acesso Sequencial:

- Você pode acessar diretamente qualquer registro no arquivo e, em seguida, acessar outros registros sequencialmente a partir daí.

4. Acesso Indexado ou Acesso por Chave:

- Este é o método mais complicado;
- Baseia-se em acesso direto;
- O arquivo deve ter uma área de índice na qual haja ponteiros para diferentes registros.
- Quando a aplicação deseja acessar o registro, deve especificar uma chave, e o sistema buscará o ponteiro correspondente na área de índice através desta chave para acessar diretamente o arquivo.

5.8.2. Virtualização e *Cloud Computing*

A virtualização está relacionada ao compartilhamento de recursos de TI criando uma infraestrutura virtual (compartilhando vários recursos dedicados) a partir de um ambiente físico. Ele permite que você execute vários sistemas operacionais e aplicativos simultaneamente na mesma máquina virtual. Ao mesmo tempo, a computação em nuvem se propõe a fornecer soluções de TI por meio de um modelo de serviço de rede compartilhado ou dedicado. No entanto, embora sejam diferentes entre si, podem ser complementares. Os provedores de nuvem usam a virtualização para maximizar sua infraestrutura de serviço e melhorar o gerenciamento do *data center*, reduzindo ainda mais os custos e os requisitos de manutenção. Os provedores de nuvem ainda fornecem soluções de virtualização para seus clientes. A virtualização tem três características que a tornam ideal para a computação em nuvem, que são:

- **Particionamento:** Possibilidade de suportar várias aplicações e sistemas operacionais em um único sistema, e alocar os recursos disponíveis de acordo com a necessidade de cada sistema.
- **Isolamento:** Cada máquina virtual é isolada de seu sistema *host* físico e de outras máquinas virtuais. Dessa forma, se uma instância virtual falhar, as outras máquinas virtuais não terão problemas. Além disso, nenhum dado é compartilhado entre um contêiner virtual e outro contêiner virtual.

- **Encapsulamento:** Uma máquina virtual pode ser representada (ou mesmo armazenada como) um único arquivo, para que você possa identificá-la facilmente com base nos serviços que ela fornece. Essencialmente, o processo encapsulado pode ser um serviço comercial. A máquina virtual encapsulada pode ser fornecida ao aplicativo como uma entidade completa. Portanto, o pacote de software pode proteger cada aplicativo sem interferir em outros aplicativos.

Assim, quanto mais ambientes virtualizados, melhores serão os resultados no processo de implantação da nuvem. Os principais benefícios serão o gerenciamento aprimorado do ambiente de TI, a segurança da informação e economias significativas de custos.

5.8.3. Segurança

A proteção de acesso a arquivos visa realizar o compartilhamento seguro de arquivos entre usuários quando necessário. Geralmente, se deve conceder direitos de acesso, como leitura, gravação, execução e exclusão.

Existem diferentes mecanismos de níveis de proteção. Alguns deles são:

Senha de Acesso:

- O sistema concede acesso a determinados arquivos / diretórios por meio de senhas;
- Cada arquivo possui apenas uma senha, e a autoridade de acesso pode ter diferentes níveis de acesso;
- A desvantagem de compartilhar, porque igual o proprietário, todos os outros usuários precisam saber a senha.

Grupos de Usuários:

- Existe em vários sistemas operacionais;
- Associar cada usuário a um grupo de usuários que compartilham arquivos e diretórios;
- Existe três níveis de proteção: *owner* (dono), *group* (grupo) *all* (todos);
- O tipo de acesso (leitura, gravação, execução e exclusão) deve estar associado a três níveis de proteção.

Lista de Controle de Acesso (*Access Control List* - ACL):

- Consiste em uma lista associada a cada arquivo, especificando usuários e tipos de acesso permitidos;
- A lista de verificação do sistema operacional se a operação exigida pelo usuário está autorizada;

- Considerando que um arquivo pode ser compartilhado por vários usuários, a estrutura pode ser muito grande.
- A pesquisa sequencial na lista pode causar *overhead*.

5.8.4. Aplicação

```
def Escrever(x):
    arquivo = open("cota.txt","w")
    arquivo.write(str(x))
csv = "Brasil.csv"
cotac = pd.read_csv(csv)
```

5.9. Entradas e Saídas

Uma das principais funções do sistema operacional é gerenciar os dispositivos de entrada e saída (E / S) conectados ao computador. A tarefa do sistema operacional é enviar um sinal para informar ao usuário o que ele deseja que o dispositivo execute. Lide com interrupções e erros gerados pelo dispositivo.

O dispositivo de *hardware* precisa ser controlado para fornecer entrada e saída para o processador. O controle do *hardware* é realizado por meio de *hardware* e *software* apropriados.

A parte de *hardware* é chamada de controlador de *hardware*, que segue o padrão determinado pelo barramento (IDE, SCSI, USB etc.). Portanto, existem controladores de *hardware* conectados a cada tipo de barramento: controlador de *hardware* IDE, controlador de *hardware* SCSI etc.

Para usar um dispositivo de *hardware*, ele deve estar conectado à interface física do controlador de *hardware*. Por exemplo, um disco rígido IDE deve ser conectado a uma das quatro interfaces disponíveis no controlador IDE. Geralmente, o sistema operacional pode ter um *software* de *driver* de dispositivo. O *driver* de dispositivo do controlador de *hardware* geralmente é universal e está embutido no próprio sistema operacional. Além disso, os *drivers* para dispositivos de *hardware* geralmente são específicos porque controlam funções específicas fornecidas pelo fabricante.

Um dispositivo periférico é qualquer dispositivo de *hardware* conectado a um computador para permitir que ele interaja com o mundo exterior.

- ***Software* de E/S:**

O objetivo principal do *software* gerenciador de I / O é padronizar o acesso e o controle dos dispositivos, tanto quanto possível, permitindo assim que novos dispositivos sejam inseridos no sistema do computador sem a necessidade de outro *software* auxiliar. Devido à diversidade, complexidade e particularidade dos periféricos descobertos, esta se tornou uma tarefa muito complexa. Por conveniência, o *software* de E / S geralmente é dividido em várias camadas. Por meio de uma série de operações comuns a todos os dispositivos, cada camada tem funções bem definidas necessárias para executar e interfaces bem definidas de camadas adjacentes. Uma maneira de conseguir essa estrutura é dividir o *software* em quatro camadas, das quais temos a camada superior onde o usuário pode ver o I / O. A segunda camada, independente do dispositivo, visualiza o *software* de I / O. Da mesma forma, a terceira camada é usada como interface padrão para o *driver* e a última interface (camada inferior) é composta pelo próprio *driver*.

- **Utilização de *buffer*:**

Quando os dados não podem ser armazenados no destino, o *buffer* pode ser usado - exatamente como um pacote que a rede recebe e precisa ser verificado. Outro exemplo são os dispositivos com restrições de tempo real, nos quais os dados devem ser colocados no *buffer* de saída com antecedência para separar a taxa na qual o *buffer* é preenchido. Esta taxa é calculada com base na taxa de esvaziamento. Desta forma, as sobreposições de *buffer* são evitadas.

5.9.1. Impasses Envolvidos

Certos dispositivos devem ser dedicados ao usuário até que o usuário conclua sua tarefa e não podem ser interrompidos para atender aos requisitos de outros processos. Quando dois processos alocam recursos um ao outro de forma que nenhum deles possa executar tarefas, mas eles não os disponibilizam antes de executar as tarefas, esses processos ficam em um estado de conflito e permanecem lá até que fatores externos os deixem se livrar dessa situação. O princípio básico do *deadlock* foi formalmente descrito: se cada processo em um grupo de processos está esperando por um evento que só pode ser causado por outro processo no grupo, então um grupo de processos está em um *deadlock*. Quando todos os processos estão esperando, nenhum deles causará quaisquer eventos que possam despertar outros membros do conjunto, e todos os processos esperarão para sempre.

6. Bibliografia

KNOW SOLUTIONS. **O que é *Business Intelligence* (BI)?** Disponível em https://www.knowsolution.com.br/o-que-e-business-intelligence-bi/. Acesso em 01/12/2020.

WIKIPÉDIA, A ENCICLOPÉDIA LIVRE. **Inteligência empresarial.** Disponível em https://pt.wikipedia.org/wiki/Intelig%C3%AAncia_empresarial. Acesso em 01/12/2020.

ICMP, CONSULTORIA EM TI. **7 vantagens de usar Linux que todo profissional de TI deveria conhecer.** Disponível em https://www.icmpconsultoria.com.br/post/2017/03/02/7-vantagens-de-usar-linux. Acesso em 01/12/2020.

[1] ©TOTVS, **Descrição de Infra Estrutura:** Requisitos de *Hardware*. Disponível em https://tdn.totvs.com/download/attachments/498434294/Requisitos_Hardware_v2.pdf?version=1&modificationDate=1561467040820&api=v2. Acesso em 01/12/2020.

©CANONICAL UBUNTU. ***Security Certifications.*** Disponível em https://ubuntu.com/security/certifications. Acesso em 01/12/2020.

AMOROSO, DANILO. **O que são processos de um sistema operacional e por que é importante saber.** Disponível em https://www.tecmundo.com.br/memoria/3197-o-que-sao-processos-de-um-sistema-operacional-e-por-que-e-importante-saber.htm. Acesso em 02/12/2020.

REDAÇÃO CANALTECH. **O que é *Thread*?** Disponível em https://canaltech.com.br/produtos/o-que-e-thread/. Acesso em 02/12/2020.

COLETTA, ALEX DE FRANCISCHI. **Escalonamento de Processos.** Disponível em https://alexcoletta.eng.br/artigos/escalonamento-de-processos/. Acesso em 02/12/2020.

WIKIPÉDIA, A ENCICLOPÉDIA LIVRE. **Linguagem de *script***. Disponível em https://pt.wikipedia.org/wiki/Linguagem_de_script. Acesso em 02/12/2020.

PROF DE SIQUEIRA, FERNANDO. **Gerência de Memória**. Disponível em https://sites.google.com/site/proffernandosiqueiraso/aulas/9-gerencia-de-memoria. Acesso em 02/12/2020.

ALENCAR, FELIPE. **Entenda o que é sistema de arquivos e sua utilidade no PC e no celular.** Disponível em https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2016/02/entenda-o-que-e-sistema-de-arquivos-e-sua-utilidade-no-pc-e-no-celular.html. Acesso em 02/12/2020.

PROF. DR. ZAMBIASI, SAULO POPOV. **Sistema de Arquivos.** Disponível em https://www.gsigma.ufsc.br/~popov/aulas/so1/cap10so.html. Acesso em 02/12/2020.

CANAL SYNNEX WESTCON. **Virtualização E *Cloud Computing* Estão Relacionadas?** Disponível em https://blogbrasil.westcon.com/virtualizacao-e-cloud-computing-estao-relacionadas. Acesso em 02/12/2020.

WIKILIVROS. **Sistemas operacionais/Gerência de dispositivos de entrada e saída.** Disponível em https://pt.wikibooks.org/wiki/Sistemas_operacionais/Ger%C3%AAncia_de_dispositivos_de_entrada_e_sa%C3%ADda. Acesso em 02/12/2020.