Roteiro de Atividade de Aprendizagem – Atividade Prática Orientada (APO)

Disciplina: **Arquitetura de Computadores e Sistemas Operacionais**
Atividade: **Integração entre Hardware e Software: Otimização de Recursos Computacionais em Sistemas Reais.**
Temática: **Otimização e gerenciamento eficiente de recursos computacionais**
Professor: **Eduarda Maganha de Almeida**

1 Contextualização

A Atividade Prática Orientada tem como objetivo desenvolver habilidades práticas e conceituais em Arquitetura de Computadores e Sistemas Operacionais para explorar e aplicar estratégias de forma eficaz e criativa. A presente Atividade Prática Orientada (APO) tem como objetivo integrar conhecimentos teóricos e práticos das disciplinas **Arquitetura de Computadores** e **Sistemas Operacionais**, promovendo a aplicação contextualizada desses saberes em cenários reais ou simulados do mundo do trabalho.

Os principais conteúdos abordados nas disciplinas serão inter-relacionados, permitindo ao estudante compreender:

- Tópicos como organização interna dos computadores, funcionamento de processadores, gerenciamento de memória e interconexões de dispositivos.
- Conceitos de gerenciamento de recursos, processos e threads, sistemas de arquivos, e estratégias de escalonamento.

Essa articulação visa explorar a interdependência entre hardware e software para o funcionamento eficiente de sistemas computacionais.

Os objetivos da APO foram elaborados para promover o desenvolvimento de competências em diferentes níveis cognitivos:

- **Conhecimento**: Reconhecer os elementos fundamentais de uma arquitetura de computadores e as funções principais de um sistema operacional.
- **Compreensão**: Explicar como os sistemas operacionais interagem com o hardware para gerenciar recursos e processos.

- **Aplicação**: Resolver problemas práticos de otimização de desempenho em arquiteturas computacionais e sistemas operacionais.
- **Análise e Avaliação**: Avaliar a eficiência de diferentes estratégias de gerenciamento de recursos e propor melhorias.

As atividades foram desenhadas para refletir desafios encontrados no mundo do trabalho, como:

- Solução de problemas em ambientes computacionais reais.
- Desenvolvimento de raciocínio lógico para diagnóstico e manutenção de sistemas computacionais.
- Análise crítica de escolhas arquiteturais e operacionais em diferentes contextos de aplicação.

Por meio de desafios práticos e investigações orientadas, a APO promoverá o aprendizado ativo, estimulando o pensamento crítico, a resolução de problemas e o trabalho colaborativo. As atividades foram estruturadas para incentivar o auto estudo e o autogerenciamento, habilidades indispensáveis em um mercado de trabalho dinâmico e em constante evolução.

A ênfase em competências técnicas aplicadas e habilidades transversais, como organização, comunicação e trabalho em equipe, visa preparar os estudantes para atender às demandas do mercado de trabalho, favorecendo sua empregabilidade e capacidade de adaptação às transformações tecnológicas.

Essa APO representa uma oportunidade única de consolidar os conhecimentos adquiridos ao longo das disciplinas, enquanto desenvolve habilidades práticas que capacitam os estudantes para enfrentar desafios reais com autonomia e eficiência.

A seguir cada etapa das atividades apresentam um resumo referente ao tema que contempla a atividade em questão.

2 Etapas e roteiro da atividade

Etapa 1. A Jornada do Computador: Da construção ao funcionamento

O objetivo dessa atividade 1, é reforçar os conceitos fundamentais de hardware e software, por meio da identificação de palavras-chave que representam partes essenciais de um sistema computacional.

Atividade 1.

Você deverá encontrar as palavras listadas nos 2 caça-palavras propostos. Essas palavras estão relacionadas aos componentes e conceitos essenciais que formam um sistema de computador, desde a **placa-mãe** até os programas que usamos diariamente.

A G R O F R I R I S S T
F L P C L I E E I R A I
I P G U G D N D U C P N
R L L O E L R C S H L T
M A B H R I A E E I I E
W C T I V I F S N P C R
A A E E O U T P U T A F
R M R N T S U M T T T A
E Ã E L S T T R O R I C
T E S O F T W A R E V E
D I S C O R Í G I D O R
L T O A R M E M Ó R I A

ALGORITMO	APLICATIVO	BIOS	CHIP	DISCO RÍGIDO
DRIVER	FIRMWARE	GPU	INPUT	INTERFACE
MEMÓRIA	OUTPUT	PLACAMÃE	REDE	SOFTWARE

ARMAZENAMENTO	BACKPLANE	CACHE	COMPILADOR	COMPUTADOR
LINKER	LOADER	MONTADOR	PIPELINING	PROCESSADOR
RAM	REGISTRADORES	ROM	TRADUTOR	VIRTUALIZAÇÃO

Etapa 2. Simulação de Execução de Threads em um Sistema Multitarefa

O objetivo desta atividade 2 é explorar o conceito de threads em sistemas operacionais, entender como elas se comportam em sistemas multitarefa, e como a alocação de recursos pode impactar o desempenho de um sistema.

Relembrando os conceitos.

Processos e threads são conceitos fundamentais em sistemas operacionais, mas eles possuem características distintas:

- Processos: Um processo é uma instância de execução de um programa. Cada processo possui seu próprio espaço de memória, onde mantém variáveis e dados independentes dos outros processos. Ele também possui seus próprios recursos, como registradores de CPU, pilha e tabela de arquivos. Como cada processo é isolado, a comunicação entre processos requer mecanismos como pipes ou sockets.

- Threads: Uma thread, por sua vez, é a menor unidade de execução dentro de um processo. Todas as threads de um processo compartilham o mesmo espaço de memória, o que significa que podem acessar variáveis e recursos do processo de forma mais eficiente. No entanto, isso também implica que se uma thread modificar uma variável compartilhada, todas as outras threads dentro do mesmo processo serão afetadas.

Vantagens das Threads

As threads oferecem várias vantagens em termos de desempenho e eficiência:

- Execução Simultânea: Threads permitem que múltiplas tarefas sejam executadas simultaneamente. Como todas as threads de um processo compartilham a mesma memória, o tempo de comunicação entre elas é reduzido, permitindo que tarefas sejam feitas em paralelo de maneira mais eficiente.
- Uso Eficiente de Recursos: Como as threads compartilham o mesmo espaço de memória e outros recursos do processo, o overhead para criar e destruir threads é muito menor em comparação com processos. Isso torna a execução mais leve e rápida, pois não há necessidade de alocar novas áreas de memória para cada thread.
- Desempenho em Sistemas Multicore: Em sistemas com múltiplos núcleos de processamento, as threads podem ser executadas em paralelo em diferentes núcleos, aproveitando melhor a capacidade do hardware e aumentando o desempenho do sistema, especialmente em tarefas computacionalmente intensivas.
- Menor Sobrecarga de Comunicação: Como as threads compartilham a mesma memória, a comunicação entre elas é mais rápida e menos custosa. Isso é útil em aplicações que exigem um alto grau de colaboração entre diferentes tarefas.

Em resumo, enquanto os processos oferecem isolamento e independência, os threads promovem maior eficiência e desempenho superior ao permitir a execução paralela de múltiplas tarefas dentro do mesmo espaço de memória. As threads são especialmente úteis em sistemas multitarefa e em aplicações que exigem rápida comunicação e uso otimizado de recursos, como servidores web, aplicativos de redes sociais e jogos.

A atividade 2, será uma simulação de execução de threads em um sistema multitarefa. Iremos simular diferentes cenários em que threads disputam a utilização de recursos (como a CPU) e observarão como o desempenho do sistema muda com diferentes abordagens de escalonamento.

Os alunos devem observar e analisar como o número de threads e o tempo de execução de cada uma afetam o desempenho do sistema em termos de tempo de execução total, tempo de resposta e tempo de espera.

Ferramenta Sugerida: Utilize simuladores de escalonamento de threads disponível online, ou um código em Python ou outra linguagem de fácil compreensão para simular o comportamento das threads.

DICA: Nas nossas aulas ao vivo, faremos juntos estas atividades

Atividade 2.

Passos para a Simulação de Execução de Threads:

a) **Definir o Cenário de Simulação**

Antes de iniciar a simulação, defina o cenário e os parâmetros a serem utilizados. Considere o seguinte:

Número de Threads: Quantas threads você deseja simular (ex: 3, 5, 10).

Tarefas das Threads: Cada thread realiza uma tarefa simples, como um cálculo matemático, leitura de dados ou uma pausa (sleep) para simular um tempo de execução.

Prioridade das Threads: Algumas threads podem ter maior prioridade, enquanto outras têm prioridade menor.

Tempo de Execução (Burst Time): Defina o tempo de execução de cada thread em milissegundos (ms).

b) **Escolher a Ferramenta ou Linguagem de Programação**

Você pode escolher entre:

Simuladores online: Como o **Thread Scheduling Simulator**, que permite simular o comportamento das threads com diferentes algoritmos de escalonamento.

Python com Threads: Se preferir codificar, você pode usar o módulo **threading** do Python para criar e gerenciar threads.

Exemplo:

```
import threading

import time


# Função para simular uma tarefa que uma thread irá realizar

def tarefa(thread_id, tempo_execucao):

    print(f"Thread {thread_id} iniciada, executando por {tempo_execucao} segundos...")

    time.sleep(tempo_execucao)  # Simula o tempo de execução da thread
```

```
    print(f"Thread {thread_id} finalizada!")

# Lista de tempos de execução para as threads
tempos_de_execucao = [2, 4, 1, 3, 5]  # Tempo de execução para cada thread (em segundos)

# Criar threads
threads = []
for i, tempo in enumerate(tempos_de_execucao):
    thread = threading.Thread(target=tarefa, args=(i+1, tempo))
    threads.append(thread)
    thread.start()

# Esperar todas as threads finalizarem
for thread in threads:
    thread.join()

print("Simulação de execução de threads concluída.")
```

Explicação do Código:

threading.Thread(): Cria uma nova thread. Cada thread executará a função tarefa, que recebe um ID de thread e o tempo de execução simulado (com o time.sleep()).

start(): Inicia a execução das threads.

join(): Garante que o programa aguarde a finalização de todas as threads antes de terminar.

Você pode modificar a simulação de várias maneiras para experimentar diferentes cenários:

- **Ajustando a Prioridade das Threads:** Embora o Python não tenha suporte direto a prioridades de threads, você pode simular isso ajustando o tempo de execução ou controlando a ordem em que as threads são iniciadas.
- **Usando Algoritmos de Escalonamento:**
    - **FIFO (First-In-First-Out):** As threads são executadas na ordem em que são criadas.

- **Round Robin:** As threads são executadas em ciclos, com cada thread recebendo uma fatia de tempo de CPU.
- **Prioridade:** Alterar o tempo de espera para simular o comportamento de threads de alta e baixa prioridade.

Como Fazer a Atividade:

a) **Iniciar a Simulação:**

Abra o simulador e crie um novo cenário de execução com **3 threads** de exemplo. Defina diferentes tempos de execução para cada thread (por exemplo, Thread 1: 5ms, Thread 2: 10ms, Thread 3: 3ms).

b) **Alterar Prioridades (SUGESTÃO):**

Modifique a prioridade das threads (Thread 1 de alta prioridade, Thread 2 de média, Thread 3 de baixa prioridade) e observe como o tempo de execução total e a distribuição de CPU muda.

c) **Adicionar mais Threads (SUGESTÃO):**

Aumente o número de threads para 5 ou 10 e observe como o tempo de execução geral é afetado. Tente simular um sistema com mais threads do que núcleos de CPU disponíveis.

d) **Analisar Resultados:**

Anote os tempos totais, de resposta e de espera de cada execução e compare-os para verificar se há algum padrão ou tendência. Discuta os resultados observados e como a alocação de threads afeta o desempenho do sistema.

Etapa 3. Decifrando o Sistema: O Papel do Kernel no Gerenciamento de Recursos

Relembrando os conceitos.

Kernel: É o núcleo do sistema operacional e tem a responsabilidade de gerenciar os recursos do computador, como a CPU, memória, e dispositivos de entrada e saída. O kernel atua como intermediário entre o hardware e o software, controlando e alocando recursos de forma eficiente. Ele gerencia a execução de processos, manipulação de dispositivos, e segurança do sistema.

Processos: São programas em execução no sistema. Um processo é uma instância de um programa que foi carregado na memória e está sendo executado pela CPU. Cada processo tem seu próprio

espaço de memória e recursos. O sistema operacional gerencia múltiplos processos simultaneamente, garantindo que eles sejam executados sem interferir uns nos outros.

Threads: São unidades menores dentro de um processo. Um processo pode ter múltiplas threads, que compartilham o mesmo espaço de memória, mas executam tarefas diferentes em paralelo. Threads são mais leves do que processos e permitem que programas realizem múltiplas tarefas simultaneamente, aumentando a eficiência.

Gerenciamento de Dispositivos: Refere-se ao controle e à interação do sistema operacional com os dispositivos de hardware, como discos rígidos, impressoras e teclados. O sistema operacional utiliza **drivers** para comunicar-se com esses dispositivos e garantir que eles funcionem corretamente. O gerenciamento de dispositivos envolve a alocação de recursos para as E/S (entrada/saída), controle de acesso a dispositivos compartilhados, e otimização do desempenho.

Observe os cenários seguintes, para resolver a atividade 3. Essas situações exemplificam como o kernel é responsável por tomar decisões críticas que afetam a performance, a segurança e a estabilidade do sistema operacional e do ambiente computacional.

Cenário 1: Alocação de Memória

Situação: Um aplicativo solicita mais memória para armazenar dados temporários.

Decisão do Kernel: O kernel verifica a quantidade de memória disponível e decide como alocar esse espaço. Se a memória RAM estiver cheia, ele pode utilizar o **swap**, movendo dados temporariamente para o disco rígido (memória virtual), ou priorizar a alocação para processos mais críticos.

Impacto: A decisão do kernel impacta diretamente no desempenho, pois mover dados para o disco rígido pode reduzir a velocidade de processamento.

Cenário 2: Gerenciamento de Processos (Escalonamento)

Situação: O sistema tem múltiplos processos prontos para execução, mas apenas uma CPU disponível.

Decisão do Kernel: O kernel precisa decidir qual processo será executado, utilizando um algoritmo de escalonamento (como **FIFO**, **Round Robin** ou **Prioridade**). Ele pode escolher dar prioridade a processos interativos (como um aplicativo de navegador) ou a processos de longa execução (como um serviço de fundo).

Impacto: O desempenho do sistema e a experiência do usuário são afetados, já que a alocação de CPU pode influenciar a latência e a fluidez de operações.

Cenário 3. Interrupções de Hardware

Situação: O teclado ou o mouse envia uma interrupção para o sistema para sinalizar que o usuário realizou uma ação (como pressionar uma tecla ou mover o mouse).

Decisão do Kernel: O kernel recebe a interrupção e decide o que fazer com ela. Em sistemas multitarefa, ele pode interromper a execução do processo atual para atender à interrupção e garantir que a entrada do usuário seja processada rapidamente.

Impacto: O kernel deve gerenciar a interrupção de forma eficiente para garantir que a resposta do sistema seja rápida e sem perda de dados ou erros.

Atividade 3.

Com base no CENÁRIO ATIVIDADE 3 proposto a seguir, você deve:

- **Explicar o problema identificado.**
- **Propor uma solução fundamentada em conceitos técnicos.**
- **Refletir sobre como essa solução impacta o desempenho geral do sistema.**

CENÁRIO ATIVIDADE 3:

"Um processo está travado esperando por recursos que outro está utilizando. Qual solução o kernel pode implementar para resolver o problema?"

3 Materiais complementares e anexos

3.1 Materiais obrigatórios

- MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de sistemas operacionais**. LTC, 2004.
- TANENBAUM, Andrew S.; WOODHULL, Albert S. **Sistemas Operacionais: Projetos e Implementação**. Bookman Editora, 2009.

3.2 Leituras e materiais complementares

- OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. Sistemas operacionais. **Revista de informática teórica e aplicada. Porto Alegre. Vol. 8, n. 3 (dez. 2001), p. 7-39**, 2001. https://lume.ufrgs.br/handle/10183/19242
- MAZIERO, Carlos A. Sistemas operacionais: conceitos e mecanismos. **Livro aberto**, v. 28, 2014. https://www.researchgate.net/profile/Carlos-Maziero/publication/343921399_Sistemas_Operacionais_Conceitos_e_Mecanismos/links/626bd87c0df856128f868125/Sistemas-Operacionais-Conceitos-e-Mecanismos.pdf

4 Cronograma de orientações e Critérios de Avaliação

Etapas	Período ideal para buscar orientação
Etapa 1. A Jornada do Computador: Da construção ao funcionamento	Aula ao vivo I - Unidade I
Etapa 2. Simulação de Execução de Threads em um Sistema Multitarefa	Aula ao vivo II - Unidade I e Unidade II
Etapa 3. Decifrando o Sistema: O Papel do Kernel no Gerenciamento de Recursos	Aula ao vivo II – III - IV - Unidade I - Unidade II – Unidade III e Unidade IV
Entrega final.	Unidade I - Unidade II – Unidade III e Unidade IV

5 Critérios de Avaliação

Critério avaliado	Nota
Etapa 1.A Jornada do Computador: Da construção ao funcionamento ● Cumprimento dos requisitos solicitados na Atividade 1	10%
Etapa 2: Simulação de Execução de Threads em um Sistema Multitarefa ● Cumprimento dos requisitos solicitados na Atividade 2.	40%
Etapa 3: Decifrando o Sistema: O Papel do Kernel no Gerenciamento de Recursos Cumprimento dos requisitos solicitados na Atividade 3	50%