1. Defina Sistemas Operacionais e explique sua importância.

Um Sistema Operacional (SO) é um software que gerencia hardware, software e recursos do sistema, fornecendo uma interface para o usuário e garantindo a execução eficiente dos programas. Sua importância está na abstração do hardware, controle de processos e recursos, segurança e usabilidade do sistema.

1. Como seria a atividade de escrever um programa de computador sem poder contar com um SO?

Sem um SO, o programador precisaria gerenciar diretamente o hardware, controlando memória, CPU e dispositivos de entrada/saída manualmente. Isso tornaria o desenvolvimento extremamente complexo, exigindo conhecimento detalhado da arquitetura do computador.

1. Como um Sistema Operacional se relaciona com a Arquitetura de Computadores?

O SO interage diretamente com a arquitetura do computador, controlando o uso da CPU, memória, dispositivos de entrada/saída e barramentos. Ele traduz comandos de alto nível para instruções que o hardware compreende, otimizando a execução dos programas.

1. Explique o que representa um processo e qual seu papel no SO. Codifique 3 processos com tarefas à sua escolha.

Um processo é um programa em execução que possui seu próprio espaço de memória e recursos. No SO, ele é gerenciado pelo escalonador e pode estar em diferentes estados (pronto, em execução, bloqueado etc.).

|  |
| --- |
| import multiprocessing  def soma*()*:      resultado = 5 + 3      print(f"Soma: 5 + 3 = {resultado}")  def subtracao*()*:      resultado = 10 - 4      print(f"Subtração: 10 - 4 = {resultado}")  def saudacao*()*:      usuario = "João"      print(f"Hello, {usuario}!")  if \_\_name\_\_ == "\_\_main\_\_":      p1 = multiprocessing.Process(target=soma)      p2 = multiprocessing.Process(target=subtracao)      p3 = multiprocessing.Process(target=saudacao)      p1.start()      p2.start()      p3.start()      p1.join()      p2.join()      p3.join()      print("Processos finalizados") |

1. O que é multiprogramação? Qual o impacto para o SO? Explique o conceito e exemplifique.

Multiprogramação é a capacidade do SO de executar múltiplos processos simultaneamente ao alternar rapidamente entre eles. Isso aumenta a utilização da CPU e melhora a eficiência do sistema.

Exemplo: um sistema que executa editores de texto enquanto faz download de arquivos.

Exemplo:

* Sem o uso de multiprogramação: um SO que possui duas tarefas precisa aguardar o encerramento da primeira para que a segunda seja executada. Caso a primeira tenha operações de entrada e saída, o SO aguardará a finalização da primeira antes de executar o segunda.
* Com o uso de multiprogramação: ao executar a primeira tarefa, caso ela tenha operações de entrada e saída ou utilize um recurso por muito tempo sem avançar, o SO pode colocá-la em espera e partir para a segunda tarefa, voltando para a primeira quando necessário e sem precisar utilizar recursos para carregar as tarefas em memória novamente.

1. O que é o compartilhamento de recursos no contexto do SO? Qual a importância da concorrência?

Compartilhamento de recursos permite que múltiplos processos acessem CPU, memória e dispositivos simultaneamente. A concorrência é crucial para otimizar o desempenho e garantir que várias tarefas sejam executadas eficientemente sem conflitos.

1. Qual a missão de um escalonador de processos? Qual sua importância para o SO?

Um escalonador de processos é responsável por decidir a ordem de prioridade de execução de processos com base em requisitos específicos, como ordem de chegada do processo, nível de prioridade, tempo de duração etc. Isso ajuda o SO a gerir melhor os recursos do computador (memória e processador, por exemplo) e o tempo, de forma a executar os processos em lista da forma mais otimizada e econômica possível.

1. Explique 4 algoritmos de escalonamento de processos e codifique 1 método para cada técnica.

1. FCFS (First Come, First Served) – Processos executam na ordem de chegada.

2. SJF (Shortest Job First) – Prioriza o processo com menor tempo de execução.

3. Round Robin – Alterna entre processos em fatias de tempo fixas.

4. Prioridade – Executa processos com base em prioridade definida.

Exemplos de código para cada algoritmo:

|  |
| --- |
| from queue import Queue  def fcfs(processos):  for p in processos:  print(f"Executando processo {p}")  time.sleep(1)  processos = ["P1", "P2", "P3"]  fcfs(processos) |

|  |
| --- |
| def sjf(processos):  # Ordena por tempo de execução  processos.sort(key=lambda x: x[1])  for p in processos:  print(f"Executando {p[0]} por {p[1]} segundos")  time.sleep(p[1])  sjf([("P1", 3), ("P2", 1), ("P3", 2)]) |

|  |
| --- |
| def round\_robin(processos, quantum):  while processos:  processo = processos.pop(0)  if processo[1] > quantum:  print(f"Executando {processo[0]} por {quantum} segundos")  time.sleep(quantum)  processos.append((processo[0], processo[1] - quantum))  else:  print(f"Executando {processo[0]} por {processo[1]} segundos (finalizado)")  time.sleep(processo[1])  processos = [("P1", 5), ("P2", 3), ("P3", 6)]  quantum = 2  round\_robin(processos, quantum) |

|  |
| --- |
| def prioridade(processos):  # Maior prioridade primeiro  processos.sort(key=lambda x: x[1], reverse=True)  for p in processos:  print(f"Executando {p[0]} (Prioridade {p[1]})")  time.sleep(1)  prioridade([("P1", 2), ("P2", 1), ("P3", 3)]) |

1. Por que um SO depende dos estados do processo? Quais os benefícios de organizar o ciclo de vida dos processos em estados?

O SO depende dos estados do processo (novo, pronto, executando, bloqueado, finalizado) para gerenciar a CPU e otimizar o uso de recursos. Isso melhora a eficiência e evita sobrecarga.

1. Explique a relação existente entre ciclo de vida de processos, escalonamento de processos e compartilhamento de recursos.

O ciclo de vida do processo determina quando um processo será escalonado e como os recursos são compartilhados. O escalonamento escolhe qual processo usar a CPU, enquanto o compartilhamento de recursos garante que múltiplos processos rodem de forma eficiente sem conflitos.

1. Compare os modelos de Processo e Threads.

Processos são executados de forma independente, cada um com seu próprio espaço de memória e recursos. Isso garante maior segurança e isolamento, mas consome mais memória e torna a criação e destruição mais demoradas. Se não houver CPUs suficientes, o sistema operacional alterna entre os processos.

Threads compartilham o mesmo espaço de memória dentro de um processo, permitindo comunicação mais rápida e menor consumo de recursos. Elas podem ser executadas em paralelo, dependendo do suporte do sistema e do processador. No entanto, apresentam menor isolamento, ou seja, um erro em um thread pode afetar as demais. Além disso, o compartilhamento de memória exige mecanismos de sincronização para evitar problemas como condições de corrida.