1. Defina Sistemas Operacionais e explique sua importância.
   Sistemas Operacionais são sistemas responsáveis por facilitar a interação entre o software e o hardware. Eles permitem que qualquer pessoa utilize um sistema sem necessariamente ter conhecimentos técnicos sobre o funcionamento do hardware. No caso, a interação é feita pelo sistema operacional, que pode ter uma interface de usuário mais desenvolvida e intuitiva ou uma interface mais simples, como um terminal, que requer conhecimentos específicos de software.

2. Como seria a atividade de escrever um programa de computador sem poder contar com um SO?
   O programador precisaria se preocupar com as especificidades do hardware do dispositivo a ser programado. Além disso, o processo de desenvolvimento seria mais lento, uma vez que o SO permite que o desenvolvedor escreva e execute um código várias vezes, tendo a resposta da execução dada pelo SO; sem ele, esse processo é menos otimizado.

3. Como um Sistema Operacional se relaciona com a Arquitetura de Computadores?
   Um sistema operacional faz o gerenciamento de recursos, como memória, CPU, dispositivos de entrada e saída e armazenamento. Ele depende das características da arquitetura do computador, como conjuntos de instruções, modos de operação (usuário e kernel), gerenciamento de interrupções, acesso à memória etc. Além disso, ele abstrai os componentes físicos para os usuários e desenvolvedores, dispensando a necessidade de conhecimento específico em hardware para a utilização de um dispositivo.

4. Explique o que representa um processo e qual seu papel no SO. Codifique 3 processos com tarefas à sua escolha.
   Um processo é uma tarefa a ser executada no SO. Ele ajuda o SO a não se sobrecarregar com execuções indefinidas. O processo é uma tarefa dinâmica, pois, ao contrário de um programa, que é único em sua execução, um processo pode apresentar diversas mudanças constantes, de acordo com a execução dos demais processos na lista e com os recursos disponíveis.

```
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")
```

5. O que é multiprogramação? Qual o impacto para o SO? Explique o conceito e exemplifique.
   A multiprogramação é o processo de carregar programas na memória de uma só vez. Isso faz com que o SO consiga alternar rapidamente entre os programas, resultando em uma execução quase que simultânea. Isso permite que o SO maximize o uso de CPUs e reduza o tempo de espera por operações de entrada e saída.
   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.

6. O que é o compartilhamento de recursos no contexto do SO? Qual a importância da concorrência?
   A concorrência é importante no contexto do SO pois permite que, através do compartilhamento de recursos, um recurso do computador (memória,

processador ou outro) seja destinado à tarefa mais necessitada. Isso faz com que recursos não sejam monopolizados por um único processo, o que poderia levar a uma falha ou travamento no sistema.

7. 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.

8. Explique 4 algoritmos de escalonamento de processos e codifique 1 método para cada técnica.
   - FIFO (First In, First Out): os processos são executados por ordem de tempo de chegada, em ordem crescente. Ou seja, o primeiro a chegar na lista é o primeiro a ser executado.

```
def fifo(processos):
     # Ordena pelo tempo de chegada
     for _, _, _, _, func in sorted(processos, key=lambda
     x: x[0]):
          func()
```

   - LIFO (Last In, First Out): semelhante ao FIFO, os processos são executados por ordem de tempo de chegada, mas em ordem decrescente. Isto é, o último a chegar na lista é o primeiro a ser executado.

```
def lifo(processos):
     # Ordena pelo tempo de chegada e pela menor duração.
     Executa a partir do último a chegar
     for _, _, _, _, func in sorted(processos, key=lambda
     x: (-x[0], x[1])):
          func()
```

   - SJF (Shortest Job First): execução por menor tempo de duração do processo. Os processos mais curtos em tempo de execução são executados e descartados primeiros, deixando os mais custosos por último.

```
def sjf(processos):
```

```
    # Ordena pelo tempo de chegada e pela menor duração
    for _, _, _, _, func in sorted(processos, key=lambda
    x: (x[0], x[1])):
        func()
```

- Prioridade: execução por ordem de prioridade. Se cada processo possui um nível de prioridade de 1 a 5, onde 1 é o mais urgente, por exemplo, eles serão ordenados por ordem crescente de prioridade, de forma que os de prioridade nível 1 sejam executados primeiro, depois os de nível 2 e assim por diante.

```
def prioridade(processos):
    # Ordena por prioridade, tempo de chegada e tamanho.
    Executa os de maior prioridade primeiro
    for _, _, _, _, func in sorted(processos, key=lambda
    x: (-x[3], x[0], x[2])):
        func()
```

9. Por que um SO depende dos estados do processo? Quais os benefícios de organizar o ciclo de vida dos processos em estados?
   Os estados dos processos ajudam o SO a gerenciar quais recursos liberar para determinados processos. Por exemplo, quando um processo em execução, que está utilizando recursos do computador (como uma CPU, por exemplo), é finalizado, o seu estado é atualizado e, assim, o SO entende que pode alocar aquele recurso para um outro processo que esteja necessitando, esteja ele em espera ou em execução.

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

    O ciclo de vida de um processo representa seus estados durante sua existência no sistema operacional. O escalonamento de processos é a técnica usada pelo SO para gerenciar a execução desses processos, determinando qual processo será executado primeiro de acordo com algum parâmetro, garantindo eficiência no uso do processador. Já o compartilhamento de recursos envolve o controle de CPU, memória e dispositivos entre processos concorrentes, evitando conflitos e garantindo que múltiplos processos possam operar simultaneamente e de forma otimizada, sem monopolizar recursos.