|  | **INSTITUTO FEDERAL DO RN**  **Campus Natal-Central** |
| --- | --- |
| **Disciplina:** Sistemas Operacionais |
| **Professor(a):** Leonardo Ataide Minora |
| **Discente:** Lucas Pinheiro da Costa **Matrícula:** 20231014040023 |
| **Curso:** TADS **Semestre:** 2024.2 |
| **Resolução:** Lista de exercícios do Cap. 05 do livro “Sistemas Operacionais: Conceitos e Mecanismos” |

CAPÍTULO 05 - IMPLEMENTAÇÃO DE TAREFAS

1. Explique o que é, para que serve e o que contém um TCB - Task Control Block.

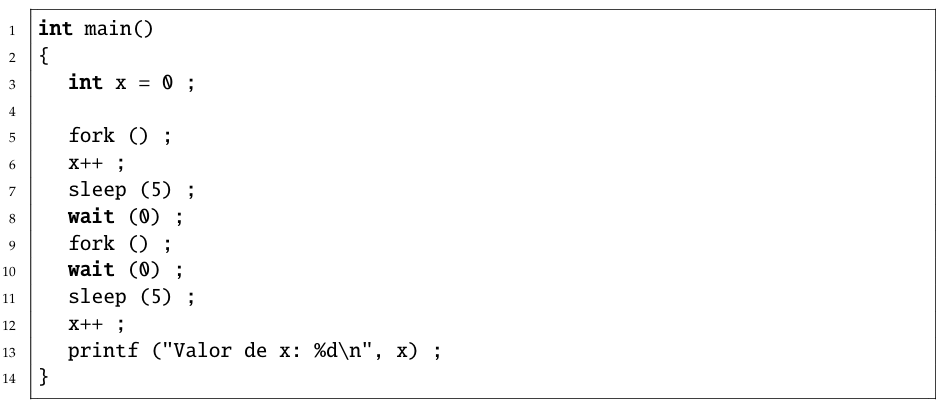
**Resposta: O Task Control Block (TCB) é uma estrutura de dados essencial dentro de um sistema operacional que serve para armazenar todas as informações relevantes sobre uma tarefa ou processo em execução. Ele funciona como uma espécie de "cartão de identidade" de cada tarefa, possibilitando ao sistema operacional gerenciar e monitorar o estado e o progresso de todas as tarefas de maneira eficaz.**

**O TCB contém diversos dados críticos sobre a tarefa, que são acessados e atualizados pelo sistema operacional conforme necessário. Primeiramente, ele armazena o identificador do processo (PID), um número único que diferencia cada tarefa das demais no sistema. Outro dado fundamental é o estado da tarefa (como "Pronta", "Executando" ou "Suspensa"), permitindo ao sistema operacional saber exatamente em qual ponto do ciclo de vida a tarefa se encontra. O TCB também armazena o contador de programa, que indica a próxima instrução a ser executada pela tarefa caso ela retorne ao processador, além de informações sobre o contexto de execução, como registradores e ponteiros de pilha, que garantem que o processo pode ser retomado exatamente de onde parou.**

**Outros elementos armazenados no TCB incluem as permissões de acesso e as informações sobre os recursos que o processo utiliza, como memória alocada, arquivos abertos, e prioridades para escalonamento. Com essas informações, o sistema operacional pode organizar e priorizar o uso dos recursos de maneira eficaz. Em sistemas de múltiplos processos, o TCB também facilita a troca de contexto, salvando e restaurando informações cruciais de uma tarefa para outra quando ocorre uma mudança de execução.**

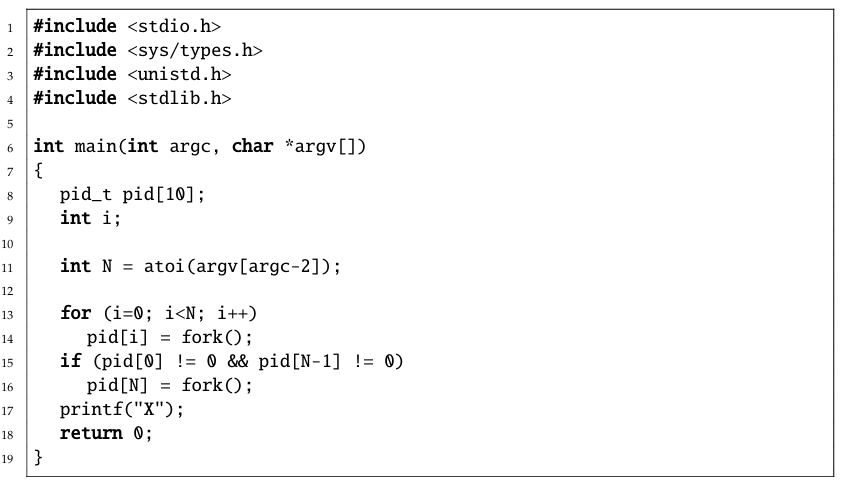
**Assim, o TCB é fundamental para o gerenciamento de processos, assegurando que o sistema mantenha um controle rigoroso e organizado sobre o status de cada tarefa, maximizando a eficiência e a segurança na utilização dos recursos do sistema.**

1. Desenhe o diagrama de tempo da execução do código a seguir, informe qual a saída do programa na tela (com os valores de x) e calcule a duração aproximada de sua execução.



1. Indique quantas letras “X” serão impressas na tela pelo programa abaixo quando for executado com a seguinte linha de comando: “a.out 4 3 2 1”

O comando *a.out* resulta da compilação do programa a seguir:



1. O que são *threads* e para que servem?

**Resposta: Threads são as menores unidades de execução dentro de um processo em um sistema operacional. Cada thread representa uma sequência independente de instruções que o processador pode executar. Em outras palavras, uma thread é uma subdivisão de um processo que compartilha recursos como memória e arquivos com outras threads do mesmo processo, mas possui seu próprio contador de programa, registradores e pilha.**

**Threads servem para permitir que um processo realize várias tarefas simultaneamente, o que é especialmente útil em aplicações que precisam realizar operações paralelas ou que têm várias tarefas independentes a serem executadas. Por exemplo, em um navegador web, uma thread pode ser responsável por exibir o conteúdo na tela, enquanto outras podem carregar páginas em segundo plano, executar scripts ou gerenciar downloads, tudo sem bloquear a interface para o usuário.**

**O uso de threads melhora a eficiência e a performance do sistema, já que várias threads de um processo podem ser executadas em paralelo em diferentes núcleos de um processador multicore, ou trocadas rapidamente em um único núcleo. Dessa forma, as threads otimizam o uso dos recursos, permitindo que programas respondam mais rapidamente e tenham um desempenho melhor em tarefas complexas.**

1. Quais as principais vantagens e desvantagens de *threads* em relação a processos?

**Resposta: As threads oferecem diversas vantagens em relação aos processos, especialmente em termos de desempenho e compartilhamento de recursos. A principal vantagem das threads é que elas compartilham o espaço de memória e outros recursos do processo, o que permite comunicação e sincronização mais rápidas e com menor sobrecarga. Como não precisam de toda a estrutura de um processo independente, as threads podem ser criadas, trocadas e encerradas mais rapidamente do que os processos, tornando-as ideais para realizar várias tarefas em paralelo dentro de uma mesma aplicação.**

**No entanto, o compartilhamento de recursos entre threads também pode ser uma desvantagem, especialmente em termos de segurança e estabilidade. Se uma thread tiver um erro ou falha, pode comprometer o funcionamento de todas as outras threads do mesmo processo, pois elas compartilham o mesmo espaço de memória e recursos. Outro desafio é a necessidade de sincronização cuidadosa para evitar problemas de concorrência, como condições de corrida, onde threads competem por recursos ao mesmo tempo, o que pode levar a resultados inconsistentes. Em contrapartida, processos têm isolamento mais forte e independência, mas custam mais em termos de recursos e tempo de troca de contexto.**

**Essas características tornam as threads vantajosas para tarefas que exigem desempenho e compartilhamento de dados, enquanto processos são mais adequados para tarefas que precisam de maior isolamento e segurança.**

1. Forneça dois exemplos de problemas cuja implementação *multi-thread* não tem desempenho melhor que a respectiva implementação sequencial.

**Resposta:** Existem cenários onde o uso de múltiplas threads não traz ganhos de desempenho e, em alguns casos, pode até degradar a eficiência. Dois exemplos típicos são:

1. **Cálculos Estritamente Sequenciais:** Em algoritmos com forte dependência sequencial, como o cálculo de termos da sequência de Fibonacci usando um método recursivo simples, a implementação multi-thread dificilmente supera a execução sequencial. Cada termo depende do cálculo dos termos anteriores, o que cria dependências que não podem ser executadas em paralelo sem um custo extra de sincronização entre as threads. Como o paralelismo é limitado por essa dependência, o overhead gerado para gerenciar múltiplas threads torna a abordagem multi-thread menos eficiente que a versão sequencial.
2. **Leitura e Escrita de Arquivos com I/O Limitado:** Se o sistema depende de operações de entrada e saída (I/O) para ler ou gravar dados em disco, e o desempenho é limitado pela velocidade do disco (ou seja, o sistema é I/O-bound), adicionar threads geralmente não melhora o desempenho. Por exemplo, um programa que lê um grande arquivo de maneira sequencial e realiza operações leves sobre cada linha do arquivo, mesmo se multithreaded, será limitado pelo tempo que o disco leva para buscar e entregar os dados. Nesse caso, o acesso simultâneo pode até piorar o desempenho, pois as threads competem pelo mesmo recurso de I/O, criando um gargalo.
3. Associe as afirmações a seguir aos seguintes modelos de threads: *many-to-one* (N:1); *one-to-one* (1:1); *many-to-many* (N:M).
   1. Tem a implementação mais simples, leve e eficiente: (N:1)
   2. Multiplexa os *threads* de usuário em um *pool* de *threads* de núcleo: (N:M)
   3. Pode impor uma carga muito pesada ao núcleo: (1:1)
   4. Não permite explorar a presença de várias *CPUs* pelo mesmo processo: (N:1)
   5. Permite uma maior concorrência sem impor muita carga ao núcleo: (N:M)
   6. Geralmente implementado por bibliotecas: (N:1)
   7. É o modelo implementado no Windows NT e seus sucessores: (1:1)
   8. Se um *thread* bloquear, todos os demais têm de esperar por ele: (N:1)
   9. Cada *thread* no nível do usuário tem sua correspondente dentro do núcleo: (1:1)
   10. É o modelo com implementação mais complexa: (N:M)
4. Considerando as implementações de *threads* N:1 e 1:1 para o trecho de código a seguir, **a) desenhe os diagramas de execução**, **b) informe as durações aproximadas de execução** e **c) indique a saída do programa na tela**. Considere a operação *sleep()* como uma chamada de sistema (*syscall*). A chamada *thread\_create* cria uma nova *thread*, *thread\_exit* encerra a *thread* corrente e *thread\_join* espera o encerramento da *thread* informada como parâmetro.

