|  | **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. 04 do livro “Sistemas Operacionais: Conceitos e Mecanismos” |

CAPÍTULO 04 - O CONCEITO DE TAREFAS

1. O que significa *time sharing* e qual a sua importância em um sistema operacional?

**Resposta: O termo "time sharing" refere-se a uma técnica de gerenciamento de recursos de um sistema operacional que permite que múltiplos usuários ou processos compartilhem o tempo de uso de uma única unidade de processamento (CPU) de forma eficiente. Em um ambiente de time sharing, o sistema operacional distribui pequenos intervalos de tempo da CPU entre os processos em execução, alternando rapidamente entre eles. Esse esquema de compartilhamento de tempo torna possível que os processos aparentemente rodem simultaneamente, mesmo em um sistema com apenas uma CPU, criando a ilusão de multitarefa e resposta em tempo real para os usuários.**

**A importância do time sharing em um sistema operacional é considerável, pois ele maximiza a utilização da CPU, reduz o tempo de resposta para processos interativos e melhora a experiência do usuário. Essa técnica é especialmente fundamental em sistemas de uso múltiplo, onde vários usuários ou aplicações necessitam de processamento frequente e quase simultâneo, como em servidores e ambientes multiusuários. Além disso, o time sharing permite uma alocação justa e eficiente de recursos, tornando os sistemas mais robustos e adequados para diversas aplicações interativas e de alto desempenho.**

1. Como e com base em que critérios é escolhida a duração de um *quantum* de processamento?

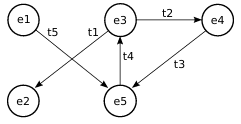
**Resposta: A duração de um *quantum* de processamento é cuidadosamente escolhida pelo sistema operacional com base em critérios que equilibram desempenho e eficiência de resposta. O *quantum* representa o intervalo de tempo que cada processo tem para executar antes de ser pausado para dar lugar a outro processo, e essa duração afeta diretamente a fluidez do sistema e a satisfação dos usuários em sistemas interativos.**

**Um dos critérios principais na escolha do *quantum* é o tipo de aplicação e o perfil de carga de trabalho esperado pelo sistema. Para sistemas interativos ou de tempo real, como servidores de aplicativos, onde os usuários precisam de uma resposta rápida, *quantuns* mais curtos são preferíveis. Um *quantum* curto reduz o tempo de espera entre a execução de processos, permitindo que o sistema responda rapidamente a novos comandos e interações, mas também aumenta a sobrecarga associada à troca de contexto, que ocorre a cada interrupção de *quantum*. Esse balanceamento entre um tempo de resposta rápido e a minimização de trocas de contexto é um dos desafios de sistemas interativos.**

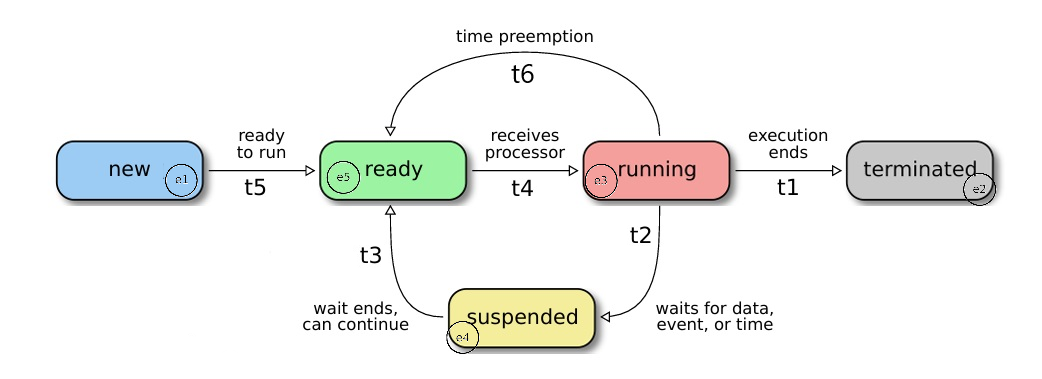
**Em contrapartida, em sistemas de processamento em lote, como servidores de processamento de dados, *quantuns* mais longos podem ser vantajosos. Com um *quantum* maior, cada processo tem mais tempo para progredir em suas operações, reduzindo o número de trocas de contexto e, consequentemente, a sobrecarga associada. Assim, processos com operações extensivas de cálculo ou manipulação de dados podem ser beneficiados por um *quantum* maior, pois eles não necessitam de interação constante com o usuário e podem tolerar períodos mais longos de processamento contínuo.**

**Outros fatores importantes na escolha do *quantum* incluem o desempenho do hardware e o design do sistema operacional. Em sistemas com CPUs de alto desempenho, onde as trocas de contexto ocorrem muito rapidamente, o impacto de um *quantum* curto é menor, o que permite uma resposta mais rápida em sistemas interativos sem grandes perdas de eficiência. Já sistemas operacionais modernos, que implementam políticas de escalonamento dinâmico, muitas vezes ajustam o *quantum* de forma adaptativa, dependendo do comportamento do processo. Isso permite que processos interativos recebam *quantuns* mais curtos, enquanto processos de longo prazo recebem quanta mais extensos, maximizando o desempenho geral do sistema.**

1. Considerando o diagrama de estados dos processos apresentado na figura a seguir, complete o diagrama com a transição de estado que está faltando (t6) e apresente o significado de cada um dos estados e transições.



Seguindo a lógica a seguir, onde:



**(e1) Nova:** A tarefa está sendo criada, i.e. seu código está sendo carregado em memória, junto com as bibliotecas necessárias, e as estruturas de dados do núcleo estão sendo atualizadas para permitir sua execução.

**(e5) Pronta:** A tarefa está em memória, pronta para iniciar ou retomar sua execução, apenas aguardando a disponibilidade do processador. Todas as tarefas prontas são organizadas em uma fila (fila de prontas, ready queue ou run queue), cuja ordem é determinada por algoritmos de escalonamento.

**(e3) Executando:** O processador está dedicado à tarefa, executando suas instruções fazendo avançar seu estado.

**(e4) Suspensa:** A tarefa não pode executar porque depende de dados externos ainda não disponíveis (do disco ou da rede, por exemplo), aguarda algum tipo de sincronização (o fim de outra tarefa ou a liberação de algum recurso compartilhado) ou simplesmente espera o tempo passar (em uma operação sleeping, por exemplo).

**(e2) Terminada:** O processamento da tarefa foi encerrado e ela pode ser removida da memória do sistema.

**(t5) Nova → Pronta:** ocorre quando a nova tarefa termina de ser carregada em memória, juntamente com suas bibliotecas e dados, estando pronta para executar.

**(t4) Pronta → Executando:** esta transição ocorre quando a tarefa é escolhida pelo escalonador para ser executada (ou para continuar sua execução), dentre as demais tarefas prontas.

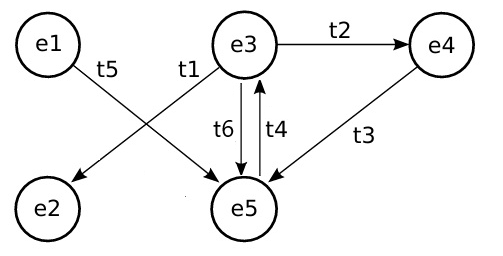
**(t6) Executando → Pronta:** esta transição ocorre quando se esgota a fatia de tempo destinada à tarefa (ou seja, o fim do *quantum*); como nesse momento a tarefa não precisa de outros recursos além do processador, ela volta à fila de tarefas prontas até recebê-lo novamente.

**(t2) Executando → Suspensa:** caso a tarefa em execução solicite acesso a um recurso não disponível, como dados externos ou alguma sincronização, ela abandona o processador e fica suspensa até o recurso ficar disponível.

**(t3) Suspensa → Pronta:** quando o recurso solicitado pela tarefa se torna disponível, ela pode voltar a executar, portanto volta ao estado de pronta para aguardar o processador (que pode estar ocupado com outra tarefa).

**(t1) Executando → Terminada:** ocorre quando a tarefa encerra sua execução ou é abortada emconsequência de algum erro (acesso inválido à memória, instrução ilegal, divisão por zero, etc.). Na maioria dos sistemas a tarefa que deseja encerrar avisa o sistema operacional através de uma chamada de sistema (no Linux é usada a chamada exit).

O diagrama de estados se comportaria da seguinte forma:



1. Indique se cada uma das transições de estado de tarefas a seguir definidas é possível ou não. Se a transição for possível, dê um exemplo de situação na qual ela ocorre (N: Nova, P: pronta, E: executando, S: suspensa, T: terminada).
   * *E → P* **Possível**
   * *E → S* **Possível**
   * *S → E* **Não é possível**
   * *P → N* **Não é possível**
   * *S → T* **Possível**
   * *E → T* **Possível**
   * *N → S* **Não é possível**
   * *P → S* **Não é possível**

**Resposta:**

*(E → P)* é possível porque ela ocorre, por exemplo, quando a tarefa executando atinge o limite de tempo do *quantum* de processamento. Nesse caso, o escalonador interrompe a execução da tarefa e a coloca de volta na fila de prontas para esperar uma nova alocação de CPU.

*(E → S)* é possível porque ela ocorre quando a tarefa em execução solicita um recurso indisponível no momento, como dados de um disco ou de uma rede, ou precisa esperar o fim de outra tarefa. Ao entrar em suspensão, ela fica no estado "Suspensa" até que o recurso necessário esteja disponível.

*(S → T)* é possível, embora rara, porque ela pode ocorrer, por exemplo, se a tarefa em estado suspenso sofre um erro crítico ou é forçada a encerrar sua execução devido a um comando do sistema, o que a leva diretamente ao estado "Terminada".

*(E → T)* é possível porque ela ocorre naturalmente quando uma tarefa completa sua execução ou encerra-se devido a um erro, como uma divisão por zero ou uma tentativa de acesso a memória inválida.

1. Relacione as afirmações abaixo aos respectivos estados no ciclo de vida das tarefas (N: Nova, P: Pronta, E: Executando, S: Suspensa, T: Terminada):

[ N ] O código da tarefa está sendo carregado.

[ P ] As tarefas são ordenadas por prioridades.

[ E ] A tarefa sai deste estado ao solicitar uma operação de entrada/saída.

[ T ] Os recursos usados pela tarefa são devolvidos ao sistema.

[ P ] A tarefa vai a este estado ao terminar seu *quantum*.

[ P ] A tarefa só precisa do processador para poder executar.

[ S ] O acesso a um semáforo em uso pode levar a tarefa a este estado.

[ E ] A tarefa pode criar novas tarefas.

[ E ] Há uma tarefa neste estado para cada processador do sistema.

[ S ] A tarefa aguarda a ocorrência de um evento externo.