

Seg Ter Qua Qui Sex Sáb Dom

11

01.

O núcleo do sistema operacional mantém descriptores de processos, denominados PCBs (Process Control Blocks), para armazenar as informações referentes aos processos ativos. Associando-se tarefas a processos, o descriptor (TCB) de cada tarefa pode ser bastante simplificado: para cada tarefa, basta armazenar seu identificador, os registradores do processador e uma referência ao processo ao qual a tarefa está vinculada.

02.

Alguns programas podem inviabilizar o funcionamento do S.O., pois a tarefa em execução nunca termina nem solicita operações de entrada/saída, monopolizando o processador e impedindo a execução dos demais tarefas. Para resolver essa questão, surgiu o conceito de time-sharing, através do sistema CTSS. Nessa solução, cada atividade que tem o processador recebe um limite de tempo de processamento, denominado quantum. Esgotado todo o quantum, a tarefa em execução perde o processador e volta para uma fila de tarefas "prontas", que estão na memória aguardando sua oportunidade de executar.

03.

A duração atual do quantum depende muito do tipo de sistema operacional; no Linux ela varia de 10 a 200 milissegundos, dependendo do tipo e prioridade da tarefa.

05.

E → P: Possível.

E → S: Possível.

S → E: Impossível.

P → N: Possível.

S → T: Impossível.

E → T: Possível. Ex.: Divisão por 0.

N → S: Impossível.

P → S: Impossível.

06.

[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 o seu quantum.

[E] 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.

09.

Threads são cada fluxo de execução do sistema, seja associado a um processo ou no interior do núcleo, é denominado thread. Elas servem para a execução de mais de um processo ao mesmo tempo.

10.

Vantagens: Ilhe e de fácil implementação. Como o núcleo somente considera uma thread, a carga de gerência imposta ao núcleo é pequena e não depende do número de threads dentro da aplicação.

Desvantagens: Como essas operações são intermediadas pelo núcleo, se uma thread de usuário solicitar uma operação de entrada e/ou saída, a thread de núcleo correspondente será suspenso até a conclusão da operação, fazendo com que todos os threads de usuário associados ao processo parem de executar enquanto a operação não for concluída. Um outro problema intrínseco às threads, diz respeito à divisão de recursos entre as tarefas. O núcleo do sistema divide o tempo do processador entre os fluxos de execução que ele conhece e gerencia: os threads de núcleo. Assim, uma aplicação com 100 threads de usuário receberá o mesmo tempo de processador que outra aplicação com apenas 1 thread (considerando que ambas aplicações possuam a mesma prioridade).

11.

O modelo de threads 1:1 (multi-thread) é adequado para a maioria das situações e atende bem às necessidades das aplicações interativas e servidores de rede. No entanto, é pouco escalável: a criação de um grande número de threads impõe uma carga significativa ao núcleo do sistema, inviabilizando aplicações com muitas tarefas (como grande servidores Web e simulações de grande porte).

12.

- [a] Tem a implementação mais simples, leve e eficiente.
- [b] Multiplexa as threads de usuário em um pool de threads de núcleo.
- [c] Pode impor uma carga muito pesada ao núcleo.
- [a] Não permite explorar a presença de várias CPUs pelo mesmo processo.
- [c] Permite uma maior concorrência sem impor muita carga ao núcleo.
- [b] É o modelo implementado no Windows NT e seus sucessores.
- [a] Se um thread bloquear, todos os demais têm de esperar por ele.
- [c] Cada thread no nível do usuário tem sua correspondente dentro do núcleo.
- [c] É o modelo com implementações mais complexas.

14.

É um dos algoritmos mais simples de agendamento de

processos em um S.O., que atribui frações de tempo para cada processo em partes iguais e de forma circular, manipulando todos os processos sem prioridades. Escalonamento Round-Robin é simples e fácil de implementar. A adição da preempção por tempo aos escalonamentos FIFO dá origem a outro algoritmo de escalonamento popular, conhecido escalonamento por revezamento, ou Round-Robin.

Ex.: FIFO (first in, first out): Onde como seu próprio nome já diz, o 1º que chega será o 1º a ser executado.

16.

Para evitar a inanição (quando um processo nunca asegura um recurso) e garantir a proporcionalidade expressas através das prioridades estáticas, um fator interno denominado envelhecimento (task aging) deve ser definido. O envelhecimento indica há quanto tempo uma tarefa está aguardando o processador e aumenta sua prioridade proporcionalmente. dessa forma, o envelhecimento evita a inanição dos processos de baixa prioridade, permitindo a eles obter o processador periodicamente.

19.

Inversão de prioridades: As tarefas devem ser definidas seguindo uma prioridade nominal ou estática. Quando as tarefas são dependentes, a inversão é inevitável.

Mutromça de prioridades: As tarefas também devem ser defi-

11

Seg Ter Qua Qui Sex Sáb Dom

nidas possuindo prioridade nominal ou estética. Utilização de política de atribuição de prioridade fixa em conjunto com uma prioridade dinâmica em ativa.