Sistemas Operacionais: Conceitos e Mecanismos
Luca Gouveia – 2019

Capítulo 2
Gerência de atividades – Respostas

1 – PCB é uma estrutura de dados que serve para armazenar as informações relativas ao contexto e os demais dados necessários à gerência de uma tarefa presente no sistema.

2 – O Time Sharing é o tipo de processamento onde se alterna entre diferentes processos de forma que o usuário tenha a impressão de que todos os processos estão sendo executados simultaneamente, permitindo a interação simultânea com múltiplos processos em execução.

3 – A cada processo é atribuído um intervalo de tempo, o quantum, no qual ele é permitido executar; Se no final do quantum o processo não terminou, a CPU sofre uma preempção e outro processo entra para executar; quando um processo termina o seu quantum, ele é colocado no final da fila.

4 –

- ➔ e1 = Criação da tarefa, e carregamento dos itens necessários para sua execução.
- ➔ e2 = A tarefa foi terminada, e já pode ser removida da memória.
- ➔ e3 = A tarefa está sendo executada.
- ➔ e4 -= A tarefa foi suspensa, pois depende de dados externos para ser executada, assim, fica esperando alguma sincronização para poder ser executada ou fica esperando o tempo passar.
- ➔ e5 = A tarefa está na memória, e pronta em uma fila aguardando ser executada apenas aguardando o processador liberar.
- ➔ t1 - A tarefa termina sua execução ou aborta por ocorrência de algum erro.
- ➔ t2 - A tarefa está sendo executada, mas necessita de algum dado externo ou sincronização para poder continuar, então, ela passa por essa transição.
- ➔ t3 - A tarefa já conseguiu o recurso necessária, e volta para a fila no estado de "pronta".
- ➔ t4 - A tarefa foi "selecionada" e está sendo executada no processador.
- ➔ t5 - A tarefa sai do estado de nova, e fica pronta esperando para ser executada pelo processador.
- ➔ t6 - A tarefa está pronta para ser executada, porém não pode ser executada por algum problema e acaba sendo removida da fila.

5 -

- ➔ a - Sim.
- ➔ b - Sim.

- ➔ c - Não.
- ➔ d - Não.
- ➔ e - Não.
- ➔ f - Sim.
- ➔ g - Não.
- ➔ h - Não.

6 -

- ➔ [N]O código da tarefa está sendo carregado.
- ➔ [P]A 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.
- ➔ [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.

7 - Valor de x: 2 Valor de x: 2 Valor de x: 2 Valor de x: 2 O tempo de execução do código aproximadamente foi entre 24 e 25 segundos. O resultado da saída está na coluna ao lado.

8 - Ao executar a linha de comando ‘a.out 4 3 2 1’ são impressos 5 “X” no terminal(“XXXXX”)

9 - De forma geral, cada fluxo de execução do sistema, seja associado a um processo ou no interior do núcleo, é denominado thread. Threads servem para se executar mais de um processo ao mesmo tempo.

10 -

Vantagens: Leve 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 um thread de usuário solicitar uma operação de E/S (recepção de um pacote de rede, por exemplo) o 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. Outro problema desse modelo 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: as threads de núcleo. Assim, uma aplicação com 100 threads de usuário irá receber o mesmo tempo de processador que outra aplicação com apenas um thread

(considerando que ambas as aplicações têm a mesma prioridade). Cada thread da primeira aplicação ir a, portanto receber 1/100 do tempo que recebe o thread único da segunda aplicação, o que não pode ser considerado uma divisão justa desse recurso.

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 um carga significativa ao núcleo do sistema, inviabilizando aplicações com muitas tarefas (como grandes servidores Web e simulações de grande porte).

12 -

- ➔ [a]Têm a implementação mais simples, leve e eficiente.
- ➔ [b]Multiplexa os threads de usuário em um pool de threads de núcleo.
- ➔ [b]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 implementação mais complexa.

14 - É o tipo de escalonamento preemptivo mais simples e consiste em repartir uniformemente o tempo da CPU entre todos os processos prontos para a execução. Os processos são organizados numa fila circular, alocando-se a cada um uma fatia de tempo da CPU, igual a um número inteiro de quanta. Caso um processo não termine dentro de sua fatia de tempo, ele é colocado no fim da fila e uma nova fatia de tempo é alocada para o processo no começo da fila. O escalonamento circular é muito simples, mas pode trazer problemas se os tempos de execução são muito discrepantes entre si. Quando existirem muitas tarefas ativas e de longa duração no sistema, as tarefas curtas terão o seu tempo de resposta degradado, pois as tarefas longas reciclaram continuamente na fila circular, compartilhando de maneira equitativa a CPU com tarefas curtas.

15 - E = tq / tq + ttc

16 - Para evitar a inanição(quando um processo nunca assegura um recurso) e garantir a proporcionalidade expressa 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.