Nome: Isaque Kaio de Araújo Rodrigues
Matrícula: 20172014040034

1) O núcleo do sistema operacional mantém descritores de processos, denominados PCBs (Process Control Blocks), para armazenar as informações referentes aos processos ativos.
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)
5) Não, não, sim, não, não, sim, sim, 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)
9) Thread é um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se auto dividir em duas ou mais tarefas. Threads servem para executar mais de um processo ao mesmo tempo.
10) Uma das **vantagens** é que isso facilita o desenvolvimento, visto que torna possível elaborar e criar o programa em módulos, experimentando-os isoladamente no lugar de escrever em um único bloco de código. Outro benefício dos threads é que eles não deixam o processo parado, pois quando um deles está aguardando um determinado dispositivo de entrada ou saída, ou ainda outro recurso do sistema, outro thread pode estar trabalhando. **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) 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 grandes servidores Web e simulações de grande porte).
12) [a]Tem 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.
13)
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) Mesmo com a aplicação de prioridades e algoritmos melhor implementados, alguns processos ainda correm o risco de sofrer *starvation* (ficar muito tempo sem receber a CPU) por isso em determinando momento pode ocorrer o que chamamos de ***aging*** (O ***aging*** ocorre quando a prioridade de um processo vai se alterando com o "tempo de vida" do mesmo, controlando o *starvation*), que muda momentaneamente a prioridade de um processo que não é executado há muito tempo e joga sua prioridade para a mais alta possível para que ele seja atendido, logo após as prioridades voltam ao normal. Outro caso em que prioridades são alteradas é quando um programa de baixa prioridade começou a fazer uso de algum periférico

de entrada e saída antes de outro de prioridade alta. Neste caso processos de alta prioridade são obrigados a esperarem os de baixa terminar sua E/S para poderem usar este periférico.

17)

18)

19) A inversão de prioridades consiste em processos de alta prioridade serem impedidos de executar por causa de um processo de baixa prioridade. Um exemplo de como pode ocorrer uma inversão de prioridades:

1.Em um dado momento, o processador está livre e é alocado a um processo de baixa prioridade pb;

2.durante seu processamento, pb obtém o acesso exclusivo a um recurso R e começa a usá-lo;

3.pb perde o processador, pois um processo com prioridade maior que a dele (pm) foi acordado devido a uma interrupção;

4.pb volta ao final da fila de tarefas prontas, aguardando o processador; enquanto ele não voltar a executar, o recurso R permanecerá alocado a ele e ninguém poderá usá-lo; 5.Um processo de alta prioridade pa recebe o processador e solicita acesso ao recurso R. como o recurso está alocado ao processo pb, pa é suspenso até que o processo de baixa prioridade pb libere o recurso. Neste momento, o processo de alta prioridade pa não pode continuar sua execução, porque o recurso de que necessita está nas mãos do processo de baixa prioridade pb. Dessa forma, pa deve esperar que pb execute e libere R, o que justifica o nome inversão de prioridades.

20)