Nolan Alves Siqueira / 2°P-Eng de Software. 10/09/2025

RA:20251243150

Exercícios Teóricos – Sistemas operacionais

1. Um programa é só um arquivo com instruções, guardado no disco, esperando para ser usado. Já o processo é quando esse programa está rodando de verdade, usando memória, CPU e outros recursos.

2. Um processo pode estar em:

- Novo: acabou de ser criado.
- Pronto: esperando a vez de usar a CPU.
- Executando: está rodando na CPU.
- Esperando: parado porque depende de algo, tipo leitura de disco.
- Terminado: já acabou.

3.O PCB é tipo a ficha do processo. Lá ficam guardadas informações como: o identificador dele (PID), em que estado está, valores dos registradores da CPU, memória que usa, arquivos abertos, prioridade e por aí vai.

4. Tudo o que ele estava usando (memória, arquivos, etc.) é liberado e fica disponível para outros processos.

5. fork() cria uma cópia do processo.
exec() troca o conteúdo do processo por um programa novo.

6.O sistema inicia os primeiros processos quando liga. Depois, cada processo pode criar outros usando fork(), e se quiser, substituir o código com exec(). O sistema vai controlando a execução, pausando e trocando processos conforme necessário.

7. Na memória compartilhada, dois processos usam a mesma parte da memória para se comunicar, o que é rápido, mas exige cuidado para não dar conflito.

Na troca de mensagens, eles não mexem diretamente na memória, e sim mandam mensagens um para o outro. É mais seguro, mas um pouco mais lento.

8.

- Pipes: pipe(), read(), write()
- Memória compartilhada: shmget(), shmat(), shmdt()
- Filas de mensagens: msgget(), msgsnd(), msgrcv()
- Sinais: kill(), signal()
- Sockets: socket(), send(), recv()

9. Porque sem isso seria uma bagunça: processos poderiam travar a máquina, não haveria divisão justa do processador, a memória poderia ser usada de forma errada, e a segurança iria embora. O SO organiza tudo e mantém o sistema funcionando bem

.

10. Processos independentes não se importam com os outros, cada um faz o seu trabalho. Já os cooperativos precisam conversar ou trocar informações, então dependem uns dos outros.

11. É um processo que já terminou, mas o sistema ainda guarda um registro dele porque o "pai" não pediu a informação final com wait(). Ele não usa CPU, só ocupa espaço na tabela de processos.

12. Na bloqueante, o processo fica parado esperando a resposta (tipo esperar mensagem chegar).Na não bloqueante, ele pede a informação e continua rodando, mesmo que não tenha resposta ainda.

13. O processo pesado tem sua própria memória e recursos. A thread é mais leve, porque várias threads podem rodar dentro do mesmo processo, dividindo memória mas funcionando como unidades independentes de execução.

14. Porque a CPU precisa dar atenção a vários processos quase ao mesmo tempo. A troca de contexto salva o que um processo estava fazendo e carrega o de outro, para dar a impressão de que todos estão rodando juntos.

15. Vantagens: é rápida e ótima para quando se precisa passar muita informação. Desvantagens: dá mais trabalho porque precisa de sincronização (pra não dar conflito) e pode gerar erros se não for bem controlada.