Atividade em sala 08

Vinícius de Oliveira Peixoto Rodrigues (245294)

Outubro de 2022

Questões 1 e 2

É o modo privilegiado do kernel, que faz acesso a funções e dados críticos que não são acessíveis no user mode. Essa restrição de acesso é feita pelo próprio processador que, ao ser inicializado, é informado sobre quais regiões da memória são de acesso privilegiado do kernel.

Questão 3

O objetivo disso é fazer a separação de privilégios entre o kernel e os outros programas, de modo que programas em user mode não consigam ter acesso a estruturas de dado ou a funções do kernel. Isso é muito importante do ponto de vista de segurança, visto que se um agente malicioso conseguisse alterar as regiões de memória do kernel, poderia por exemplo injetar código malicioso diretamente no kernel.

Questão 4

São funções oferecidas pelo sistema operacional que fornecem uma interface com o kernel, permitindo executar operações como criação/gerenciamento de arquivos, processos, etc.

Questão 5

Um exemplo onde é necessário usar syscalls é o uso de `sleep` para colocar um programa em standby. Um exemplo onde não é necessário (e é até pouco recomendável) usar syscalls é para abrir arquivos (onde é melhor usar a função `fopen` da `glibc`).

Questão 6

O `fork` cria uma cópia do processo atual (incluindo o espaço de memória não read-only, file descriptors, entre outros), enquanto o `execv` substitui a imagem do processo atual pela de outro processo.

Questão 7

Um executável moderno (especialmente os que rodam em um sistema operacional como o Linux) tem incontáveis seções, mas algumas das mais importantes são:

- Code (`.txt`): onde o código a ser executado é guardado
- Data (`.data`): onde são guardados os dados (variáveis globais e estáticas inicializadas, assim como variáveis estáticas constantes)
- Bss (`.bss`): onde são guardadas variáveis estáticas não-inicializadas

Questão 8

Porque cada processo (nó) só tem um pai, mas pode ter vários filhos. Isso caracteriza uma árvore.

Questão 9

Porque os dois possuem `pid` distintos e entradas distintas na tabela de processos.

Questão 10

No estado "ativo"um processo está competindo pelo tempo de CPU, mas não está sendo executado no momento. No estado "em execução"o programa está de fato sendo executado no momento.

Questão 11

É preciso que ele finalize sua execução e que o processo pai seja informado da sua finalização (por meio da chamada `wait`).

Questão 12

Porque elas contém seções (seja de código ou de dados) de recursos compartilhados cujo acesso deve ser controlado e feito por um só agente por vez.

Questão 13

Sim, mas a responsabilidade de garantir consistência (e.g., controle de acesso) é do programador.

Questão 14

O controle de acesso deve ser feito por meio de spinlocks/mutexes/semáforos.

Questão 15

Um semáforo é basicamente um contador que implementa duas primitivas, `give` (incrementa o contador) e `take` (decrementa o contador). Quando o contador chega a 0, a chamada `take` se torna bloqueante até algum outro agente incrementar novamente o valor do contador por meio da primitiva `give`.

Eles servem para garantir consistência no acesso de recursos compartilhados, de modo que para acessar uma região crítica o programa primeiro espera pelo semáforo, e ao sair da região libera o semáforo para outros agentes usarem.

Questão 16

É o equivalente da primitiva `take`, que decrementa o valor do contador.

Questão 17

O escalonador basicamente decide quando e quais processos vão ser executados dentro de um intervalo de tempo (*quanta*) da CPU.

Questão 18

Ele permite setar um valor numérico (denominado *niceness*) que define a prioridade de um processo (quanto menor o valor, maior a prioridade).

Questão 19

É comum enviar um `SIGKILL` para matar prematuramente um processo (seja porque ele travou, está gastando muito recursos, ou qualquer outra razão).

Questão 20

Normalmente se define um `sighandler_t` (que é uma função do tipo `void handler(int signum)`) que "captura"o sinal e decide como tratá-lo.