UniCEUB

Gustavo Balbino de Souza

RA:22000880

Lista e Lab - Sistemas Operacionais

Brasília

2020

Lista:

1) Como funciona a interface homem máquina em um sistema operacional?

R: Por meio de um painel no qual permite o usuário se comunicar com a máquina.

2) O que são sistemas operacionais de kernel monolítico? Descreva a interação entre os programas hospedados pelo sistema operacional e o hardware para esse tipo de SO.

R: Sistemas Operacionais monolíticos são aqueles os quais o SO inteiro é executado como um único programa no modo núcleo. Kernel monolítico é o modelo no qual a maioria de seus recursos são executados pelo próprio kernel.

3) O que são sistemas operacionais baseados em camadas? Descreva a interação entre os programas hospedados pelo sistema operacional e o hardware para esse tipo de SO.

R: São SO onde existem camadas dependentes umas das outras, também existindo hierarquia entre as camadas; Os níveis superiores prestam serviço aos inferiores, obedecendo a níveis de complexidade. (Operador, Programa usuário, Gerenciamento de E/S, Comunicação operador-processo, Gerenciamento de memória e Alocaçaõ do processador e multiprogramação)

4) O que são sistemas operacionais baseados em microkernel? Descreva a interação entre os programas hospedados pelo sistema operacional e o hardware para esse tipo de SO.

R: SO baseados em microkernel permitem que um cliente solicite um servidor e a resposta seja processada remotamente.

5) Para cada um desses tipos de SO, apresente um caso de uso onde a arquitetura do kernel favorece a sua utilização.

R:

- Monolítico: MS-DOS, uso em computadores comuns

- Camadas: Uso em computadores os quais necessitam realizar tarefas de forma hierárquica.

- Microkernel: Uso em servidores.

6) Explique a diferença entre os seguintes conceitos:

(a) Tarefas;

(b) Processos;

(c) Threads.

R:

a- As tarefas compartilham área de memória.

b- Os processos são executados em espaços de memória separados.

c- As threads compartilham o espaço de endereçamento e dados.

7) Apresente um exemplo prático para ilustrar as diferenças que você definiu.

R: Um software servidor Web multithread.

8) O que é a abordagem de Von Neumann? Como ela afeta a distribuição de tarefas em um Sistema Operacional?

R: A abordagem de Von Neumann seria sua arquitetura, na qual possui o seguinte cicloo de execução: A unidade de controle define a próxima instrução, após isso ela (unidade de controle) busca a instrução do programa na memória principal. Será decodificada a instrução para a linguagen na qual a ULA possa interpretar.

9) Quais são os possíveis estados dos processos em um sistema operacional?

R: Os possíveis estados são: run (está sendo executado no processador), ready (já possui todos os recursos para ser executado), sleep (bloqueado esperando algum recurso para ser executado), "zumbi" (criado por um programa e finalizado antes do resultado do processo) e parado (recebeu ordem do administrador para interromper a execução).

10) Qual é a diferença entre programação sequencial e multiprogramação? Apresente um exemplo para ilustrar suas justificativas.

R: A programação sequencial realiza um conjunto predeterminado de comandos de forma sequencial, enquanto a multiprogramação é a execução em paralelo de multiplos programas.

11) Descreva como é a utilização do processador em cada um dos exemplos, descrevendo os possíveis estados dos processos em cada etapa.

R: Na programação sequencial o processador irá trabalhar em sequência em cada etapa do processo, e na multiprogramação o processador irá trabalhar simultaneamente em vários processos. Os estados são: run, ready, sleep, zumbi e parado.

12) Descreva, com exemplos, como ocorre o problema da exclusão mútua e apresente duas possíveis soluções.

R: No momento em que um processo tenta acessar uma região crítica enquanto outro processo está utilizando a região, ocorre a exclusão mútua. Exemplos de soluções: Semáforos e Monitores.

13) O que são chamadas de sistema (SYSCALL)?

R: É a interface de programação para os serviços do SO.

14) Qual a diferença entre uma chamada trap e uma interrupção?

R: Interrupções são interrupções de hardware, enquanto traps são interrupções invocados por software.

15) Para um programados, uma chamada de sistema se parece com qualquer outra rotina de biblioteca. É importante que um programador saiba quais rotinas de biblioteca resultam em chamadas de sistema? Sob quais circunstâncias e por quê?

R: Sim, é necessário que um programador saiba as rotinas de biblioteca a fim de evitar erros.

16) A tabela da Figura 1 apresenta um comparativo entre a API em C Win32 e o API Unix POSIX. Considerando o que você sabe sobre chamadas de sistema, apresente um exemplo prático onde a comunicação entre processos é dificultada pela incompatibilidade da API Win32.

R: A chamada da interface, onde no Unix a interface é muito semelhante às chamadas ao sistema, diferente do Windowns , o qual as chamadas da interface são totalmente diferente das chamadas ao sistema.

17) Considere a chamada read apresentada em sala de aula. Descreva o comportamento do SO em cada passo de sua execução, assim como o fluxo dos dados na memória principal.

R:

1- Armazena os bytes;

2 - Carrega no buffer;

3- Gera o descritor de arquivo (fd);

4- Chama a rotina de biblioteca (call);

5- Executa a instrução TRAP. Nesse momento a chamada é promovida ao modo kernel;

6- Passa a instrução paar um endereço específico do kernel;

7- Ativa o endereço específico para a chamada (registradores);

8- Rotina de tratamento das chamadas de sistema;

9- Retorna para a instrução trap. Podem também bloquear o programa que a chamou;

10- Retorna ao programa do usuário;

11- Limpa a pilha.

Lab:

1) Crie um programa que execute uma tarefa na máquina e consuma o máximo de processador possível.

```
#include <stdio.h>



int main() {
        int i = 0;
        for (i=0;i<=1000;i++){
```

```
                printf("%d \n",i);
        }


}
```

2) Construa um exemplo, na linguagem C, onde os processos pai e filho trocam algum tipo de mensagem.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void a()
{

    if (fork() == 0)
        printf("Olá do filho!\n");



    else
        printf("Olá do Pai!\n");
}
int main()
{
    a();
    return 0;
}
```

3)