

Programador programa código fonte

Compilador converte em código de máquina e guarda na imagem executável juntamente a todos os dados estáticos e valores iniciais.

Para rodar o programa o S.O copia as instruções e dados da imagem executável para a memória física

O OS separa memória para o stack de execução. Isso armazena o endereço de variáveis locais durante chamadas de procedimento.

O OS então prepara o heap para estruturas de dados dinâmicas que o programa precise.

O O.S já tem que ter feito a mesma coisa consigo: Carregam-se na memória com seu próprio stack e heap vide imagem.

Logo: O processo é uma estrutura de dados implementada pelo Kernel.

Logo: Cada programa pode ter 0, 1 ou mais processos executando. Para cada instância existe um processo com sua

própria cópia do programa na memória.

Como o S.O dá conta monitorar os diferentes processos? R: Usando uma estrutura de dados chamada de PCB: Process Control Block

Guarda todas as informações que o S.O precisa sobre um processo: Onde está na memória, onde a imagem executável está no disco, quem pediu p/ executar, quais os privilégios o processo tem, etc.

L. O que é uma thread? R: Uma sequência lógica de instruções que executa código de um S.O ou de uma aplicação.

↳ Logo: Um processo executa um programa que consiste de um ou mais threads rodando em um entorno protegido.

↳ O que é dual mode operation? R: Uma forma de garantir a segurança do S.O. O S.O tem dois modos, de usuário e de Kernel. No modo de usuário o processador checa para ver se o processo pode executar aquele código. No modo Kernel o S.O executa com asseguranças desativadas.

↳ A dificuldade de debugar o kernel foi a origem do desenvolvimento de máquinas virtuais.

↳ Que hardware o Kernel precisa para proteger os usuários mas também rodar de forma eficaz? R: Instruções privilegiadas: Um processo pode mudar o seu nível de privilégio ao executar uma System call (SC). A SC transfere o controle para o Kernel. As instruções possíveis apenas no Kernel mode são chamadas de instruções privilegiadas

Proteção de memória: O processador tem dois registradores extras chamados Base e Bound, o Base especifica o começo da região da memória do processo na memória física enquanto o Bound delimita o fim. Cada vez que o processador busca uma instrução ele verifica se está entre Base e Bound e lança uma exceção caso não esteja.

Para contornar uma série de problemas que surgem da manipulação de memória: Compartilhamento, endereço físico e fragmentação. Os processadores in-

cluem um nível de "indireção" chamado Endereço virtual. Que é a oparência de que o programa tem a memória toda para si, enquanto o sistema aloca a memória física como for melhor para si.

**Interrupções síncronas:** Quase todos os sistemas possuem um equipamento chamado hardware timer. Em sistemas multiprocessadores há um timer para cada CPU.

## ↳ Tipos de transferência de modo.

↳ Como o O.S troca de user mode para kernel mode?

↳ User para Kernel mode: Interrupções, exceções e System Calls.

↳ Kernel para User: Novo processo, resume após interrupt, exceção em Syscall, mudança para um processo diferente e user-level-mpcall.

## ↳ API: Como o UNIX cria processos: fork e exec.

↳ Fork: Cria uma cópia do processo pai com toda sua estrutura: Um processo filho que, a menos que explicitado, roda junto com o pai, geralmente é manipulado com wait e exec

↳ Exec: Carrega um outro programa no endereço de memória, copia os args para a memória no endereço e inicializa o contexto de hardware deixando-o no ponto "start". É o comando responsável por rodar um outro programa em um processo já existente.

↳ Wait: É espera para um processo terminar, é parametrizado com o ID do filho.

## ↳ Input/Output (I/O)

↳ Uniformidade: Todos os I/O's de dispositivos, operações de arquivo e comunicação interprocesso usam as mesmas System calls: **Open, close, read e write**

↳ Open antes do uso: Unix chama "open" antes de realizar qualquer I/O, isso permite verificar permissão de acesso e realizar quaisquer registros.

↳ Orientado a bytes: Todos os dispositivos são acessados com arrays de byte. Também os canais de acesso de arquivo e canais de comunicação também se dão em bytes

↳ Kernel-buffered reads: Dados streamados de um teclado, por exemplo, são armazenados em um buffer de Kernel e devolvidos às aplicações sob demanda.

↳ Kernel-buffered writes: O Kernel também armazena dados de saída para entrega sob demanda

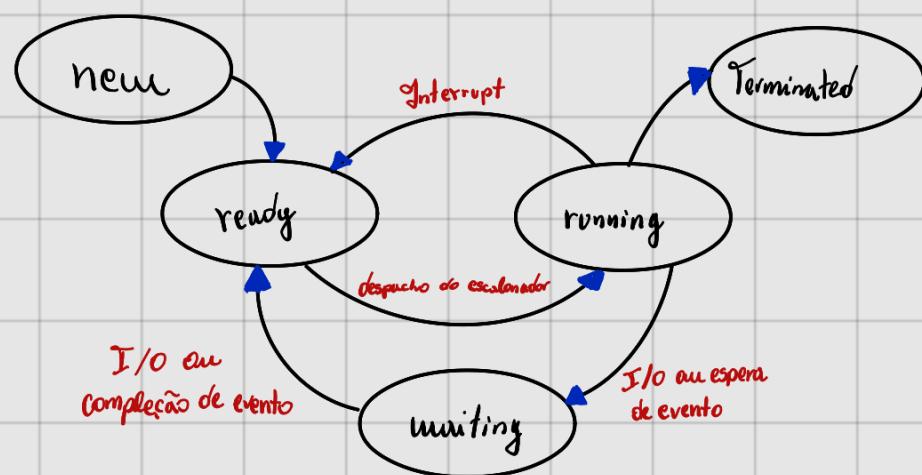
↳ Close explícito: Quando uma aplicação termina sua tarefa ela mesma chama close. O OS pode chamar o garbage collector e decrementar o contador de referência.

## ↳ Comunicação interprocessual:

↳ Pipe: É um buffer de Kernel que possui write e read, o pipe acaba quando qualquer lado terminar ou fecha.

↳ Trocar o file descriptor: Podemos manipular os file descriptors de um processo filho para fazer com que ele leia / escreva seu conteúdo em um pipe ou arquivo.

## ↳ Schedulers (escalonadores):



↳ Preemptivo: Pode interromper qualquer tarefa a qualquer tempo. É possível devido ao PCB.

↳ Não-preemptivo / cooperativo: A tarefa executa até terminar ou explicitamente ceder o processador.

## ↳ Métricas para algoritmos de escalonadores

↳ Throughput médio: Número de tasks executadas em um segundo

↳ Average Job Wait Time (AJWT): Tempo médio que um processo precisa esperar antes de ser agendado pela primeira vez.

↳ Average Job Completion Time (AJCT): Tempo médio precisa esperar antes de terminar completamente.

↳ Eficiência de CPU: A porcentagem do tempo em que a CPU realiza trabalho útil.

## ↳ Algoritmos escalonadores:

↳ **FCFS\***: First come first served. A ordem em que os processos iniciam (start) é a mesma em que têm acesso ao processador.

↳ **SJF**\*: Shortest Job First. O escalonador analisa as tarefas que estão prontas (ready), a que for mais curta é a que tem acesso ao processador. \* Ambos são cooperativos.

↳ **LCFS**: last come first served. ↳ **Round Robin (RR)**: Trocar entre tasks em intervalos de tempo fixo e também em uma ordem fixa. A chave é o compartilhamento de tempo em que cada intervalo é chamado de time quantum ou time slice, em média um slice dura 10 a 100 ms.

\* A eficiência de CPU não é boa nesse algo.

↳ **Como lidar com starvation?** ↳ **Envelhecimento de prioridade**: A prioridade (privilegio?) de um processo pode aumentar, garantindo que ele não morra de fome.

↳ **No linux, quais os schedulers utilizados?**

↳ **O(n)**: Usa uma Single linked list para armazenar os processos. A cada troca de contexto o escalonador calcula o "goodness" de cada processo, quem tem o maior goodness, executa.

↳ **O(1)**: linux tem valores de prioridade entre 0 e 139, 0 é o mais alto. O intervalo de 0 a 99 é re-

Servado para atividades de Kernel-level. Lembrar de valores "Nice". Também é uma linked list, no entanto, cada valor de prioridade tem sua linked list. A cada slice, a fila / processo mais prioritário é escolhido.

↳ **CFQ (Completely Fair Scheduler)**:

