MAC5753 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 10 de Setembro de 2020

Roteiro

Hardware

Conceitos básicos de SO

Hardware

Memória

- Cenário ideal para a memória não ser um gargalo: ser mais rápida do que a CPU executar uma instrução (0ms), ter espaço infinito e custar $0,00
- Não existe tal memória :-(
- Como balancear? Com uma memória hierárquica:
  - Registradores
  - Cache
  - Memória principal
  - Discos de estado sólido
  - Discos magnéticos

Memória

- Registradores
  - Os programas precisam gerenciar os registradores
  - SO copia o conteúdo quando faz mudança de contexto

Memória

- Cache
  - Principalmente controlada pelo hardware
  - Organizada em linhas de cache (cada linha serve para várias linhas da memória principal)
  - Se um conteúdo da memória está na cache: cache hit. Se não está: cache miss e nesse caso terá que ir até os outros níveis da hierarquia até encontrar
  - Pode ter vários níveis de cache (L1, L2, L3, ...)
  - O conceito de cache existe em vários outros dispositivos (tente abrir um arquivo duas vezes no shell e veja que será mais rápido abrir da segunda vez, tente resolver um nome na web duas vezes, ...)

Memória

- Memória principal
  - Muitas vezes chamada de RAM (Random Access Memory) – embora não seja apenas uma das características. Deveria ser Memória de Leitura e Escrita
  - Falaremos mais dela adiante no curso

Memória

- Discos
  - Magnéticos ou SSD
  - Podem ser usados para armazenar arquivos mas também como extensão de memória como memória virtual
  - Para gerenciar essa memória virtual, a MMU (Memory Management Unit) da CPU tem que agir

Dispositivos de E/S

- Cada dispositivo geralmente é formado por um controlador e pelo dispositivo propriamente dito
- O controlador é uma parte do hardware que aceita comandos do SO e intermedia isso para o dispositivo
- O controlador tenta ocultar detalhes do hardware para o SO, mas ainda assim não é trivial entender a interface que ele fornece
- O dispositivo geralmente tem que seguir algum padrão para conversar com diversas interfaces (por exemplo SATA – Serial Attached AT – para discos)

Dispositivos de E/S

- Do lado do SO, a parte dele que conversa com o dispositivo é chamado de driver
- O driver tem que estar no SO porque tem que trabalhar em modo kernel (na grande maioria das vezes)
- Há 3 formas de colocar um novo driver no kernel:
  - Relinkar o kernel com o driver e reiniciar o SO
  - Configurar no sistema de arquivos que o driver precisa ser carregado e reiniciar o SO
  - Carregar novos drivers *on-the-fly*
- No Linux geralmente os drivers são módulos do kernel que podem ser gerenciados com os comandos `lsmod`, `rmmod`, `modprobe` e `insmod`
- No Linux os drivers costumam ficar em `/lib/modules/<versão do kernel>/kernel/drivers/`

Dispositivos de E/S

- 3 formas de realizar as operações de E/S
- Espera ocupada
  - Programa roda uma chamada de sistema
  - kernel traduz a chamada de sistema para uma chamada do driver daquele dispositivo
  - O driver envia a requisição de E/S para o dispositivo e fica de tempos em tempos verificando se o dispositivo terminou
  - Quando o driver verifica que terminou ele põe os dados no local certo e retorna pro SO
  - O SO retorna o controle para o programa

Dispositivos de E/S

- Interrupção
  - O driver inicializa o dispositivo e pede para ele avisar com uma interrupção quando a operação terminar
  - O driver retorna
  - O SO vai realizar outra tarefa
  - Quando o dispositivo termina, o controlador gera uma interrupção e o SO assim fica sabendo que a operação terminou
- DMA (Direct Memory Access)
  - Um hardware especial, o chip de DMA, é usado
  - Nesse caso, o dispositivo se comunica com a memória diretamente sem necessidade da CPU

Conceitos básicos de SO

Realizando o boot no computador (de um modo geral)

- Na placa mãe há um programa chamado de BIOS (Basic Input/Output System)
- A BIOS possui um software de E/S de baixo nível para controlar o teclado, tela, discos, etc...
- A BIOS fica armazenada em uma memória flash (memória não-volátil que pode ser apagada eletricamente) que pode ser atualizada atualmente
- Computadores a partir de 2007 provavelmente vem com UEFI (Unified Extensible Firmware Interface) ao invés de BIOS. Dentre as vantagens: diagnósticos remotos, manutenção mesmo sem SO na máquina e boot seguro (com críticas a esse último)
- Sobre BIOS vs. UEFI:

  `https://pplware.sapo.pt/tutoriais/qual-diferenc%CC%A7a-bios-uefi/`

Realizando o boot no computador (de um modo geral)

- Passos de execução da BIOS:
  - Verifica quanto de memória RAM
  - Verifica se o teclado está ok
  - Verifica os barramentos dos discos
  - Verifica em qual dispositivo vai procurar pelo SO (há uma ordem armazenada em uma memória onde fica a configuração da máquina)
  - O primeiro setor do dispositivo é lido na memória e o conteúdo que está ali (carregador de boot como o grub) é executado
  - O SO é carregado, obtendo informações sobre o hardware da BIOS, e carregando os drivers necessários até habilitar a interação com o usuário

Processos

- Um programa em execução
- Cada processo possui um espaço de endereço (uma área da memória que o processo pode usar)
- Dentro do espaço de endereço: o programa em execução, os dados do programa e a pilha do programa
- Outros recursos também ficam associados a cada processo, como lista de arquivos abertos
- Em resumo, o processo é uma "caixa" com todas informações necessárias para que o programa execute

Processos

- Qual processo vai executar é definido pelo SO. Os processos ficam "brigando" pelo tempo do processador
- Assim, o estado do processo precisa ser mantido em algum lugar para que ele possa retornar ao ponto onde estava de tempos em tempos
- Geralmente há uma chamada tabela de processos que serve para manter esse estado
- Assim, um processo que está suspenso precisa pelo menos do seu espaço de endereço e de uma entrada nessa tabela de processos para continuar sua execução

Processos

- O shell precisa criar novos processos sempre que um novo programa é executado
- Nem sempre todos os comandos digitados no shell levam à criação de um novo processo!
- Chamadas de sistema são necessárias para criar novos processos (`execve` e `fork` são exemplos)
- Quando o processo chega no seu fim, chamadas de sistema são necessárias para finalizá-lo (remover seu espaço de endereço e sua entrada na tabela de processos) (`kill` e `exit` são exemplos)

Obs.: daqui em diante, tudo será apresentado considerando o Linux. Pode ser que muitas coisas também valham para outros SOs "derivados" do UNIX mas não há garantia.

Processos

- Processos criados por outro processo são chamados processos **filho** daquele processo. O que criou é chamado de processo **pai** para o que foi criado.
- Essa ideia de pais e filhos remete a uma árvore e geralmente é assim que os processos são exibidos no SO (utilitário `pstree`)
- Um processo criará outro por exemplo para dividir um trabalho entre vários processos. Para isso é necessário haver comunicação entre eles – IPC (Interprocess communication)

Processos

- Existem várias outras chamadas de sistema para manipular processos como por exemplo solicitar mais memória, esperar um processo filho terminar, etc...

Processos


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char **argv) {
   pid_t childpid;

   if ( (childpid = fork()) == 0) {
      printf("[Sou o processo filho]\n");
      while (1) {
         sleep(1);
         printf("Primeiro processo filho...\n");
      }
   }
   else {
      printf("[Sou o pai. Criei o %d]\n",childpid);
      sleep(3);
   }
   exit(0);
}
```

Processos

- Muitas vezes o SO precisa avisar algo para um processo. Faz isso por meio de **sinais**
- Por exemplo, um sinal de `alarm` pode ser enviado quando o SO precisa avisar para o processo que algo importante precisa ser feito
- Geralmente o processo precisa ter um manipulador de sinal implementado para certos sinais fazerem sentido
- Se o SO precisa matar o processo, um sinal `SIGKILL` (número 9) pode ser enviado
- Sinais estão para software assim como interrupções estão para o hardware
- `man 7 signal`
- Utilitário `kill` no Linux

Processos

- Cada usuário no sistema possui um identificador do seu usuário
- O processo, quando criado, **na maioria das vezes** é executado como o usuário que o executou
- Processos filho mantém o usuário do processo pai
- Isso é importante para que as permissões sejam respeitadas
- O usuário root "pode tudo", por isso muito cuidado com processos que rodam como o usuário root
- No Linux cada usuário possui um número associado (UID – User identification). O root sempre é o usuário 0
- O comando `chown` no Linux manipula usuários e uma chamada de sistema de mesmo nome também

Espaço de endereço

- Cada processo tem seu espaço de endereço que é a área de memória alocada para ele
- O espaço de endereço pode ser maior ou menor que a memória principal (Se for maior, usa memória virtual)
  - Antigamente, o processo não conseguiria executar se fosse maior
- Diretório `/proc` no Linux permite uma "visão" de como está a memória para cada processo