MAC0422 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 14 de Setembro de 2020

Roteiro

Conceitos básicos de SO

Um pouco mais sobre chamadas de sistema

Mais um pouco sobre processos

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

Um pouco mais sobre chamadas de sistema

Para manipular a execução de outros processos


```
while (1) {
  type_prompt();
  read_command(command, parameters);

  if (fork() != 0) {
     /* Codigo do pai */
     ...
  } else {
     /* Codigo do filho */
     execve(command,parameters,0);
  }
}
```

Para manipular a execução de outros processos


```
while (1) {
  type_prompt();
  read_command(command, parameters);

  if (fork() != 0) {
     /* Codigo do pai */
     waitpid(-1,&status,0);
  } else {
     /* Codigo do filho */
     execve(command,parameters,0);
  }
}
```

Variáveis de ambiente

- Costumam ser úteis para facilitar a execução de alguns processos
- Podem ser vistas como mais uma forma de permitir comunicação entre processos
- O `execve` recebe a lista de variáveis de ambiente como terceiro parâmetro
- O comando `export` no linux permite atribuir valores a variáveis de ambientes e o comando `echo` permite imprimir na tela o valor de uma variável de ambiente

Mais um pouco sobre processos

Resumindo o que já sabemos

- Processos são programas em execução
- Processos competem pelos recursos do computador
- Os processadores chaveiam rapidamente de processo em processo dando a impressão de que tudo está executando ao mesmo tempo (se tiver mais de um processador, de fato isso acontece)

Algumas observações sobre o modelo de processo

- Uma linha do tempo que mostre o processo que está usando a CPU dificilmente vai ser igual mesmo se você executar os mesmos programas duas vezes seguidas
- As decisões sobre qual o próximo processo que vai executar são tomadas por uma parte do SO chamada de **escalonador de processos**
- Exemplo: antes de ler arquivos de backup de uma fita magnética, importante esperar a unidade de fita alcançar a velocidade de rotação. Rodar um laço 1000 vezes para esperar isso acontecer nem sempre vai dar certo (A CPU não vai ficar só rodando esses 1000 laços)

Algumas observações sobre a criação de processos

- Nem sempre os processos precisam de um terminal ou uma janela para interagirem com os usuários. Servidores de serviços da Internet são um exemplo. Eles costumam rodar em segundo plano e geralmente são iniciados na inicialização do SO. Os processos desses servidores são chamados de daemons
- Em clones do Unix, a criação de um processo é feita apenas com a chamada de sistema `fork`. O que o `execve` faz, embora na prática seja a criação de um processo para rodar aquele programa, é mudar a "imagem" do processo que o executou na memória e executar essa "imagem". Por isso que fazer o `execve` sem um `fork` antes finalizaria um shell (Testem isso).
- Da manpage do execve: *"execve() does not return on success, and the text, data, bss, and stack of the calling process are overwritten by that of the program loaded."*

Algumas observações sobre o término de processos

- Processos podem terminar de 4 formas:
  - Saída normal (voluntária)
  - Saída por causa de erro (voluntária) – relacionado com o que o programa faz. Por exemplo, arquivo não existe
  - Erro fatal (involuntária) – acesso a endereço de memória inexistente, divisão por zero
  - Morto por outro processo (involuntária)

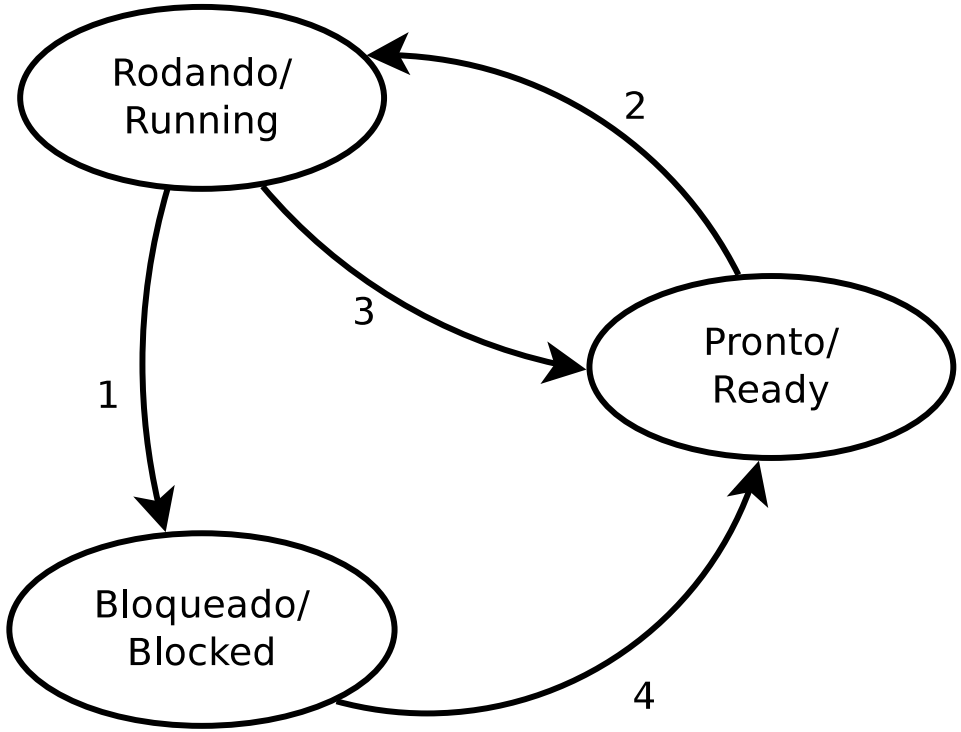
Estados de um processo

- É comum processos bloquearem durante as suas execuções (por exemplo no caso de pipe conectando dois processos, o segundo processo pode estar pronto para processar a entrada mas a entrada pode não estar pronta)

```
tail -f /var/log/syslog | grep daniel
```

- Um processo também pode ser bloqueado porque o SO escalonou outro processo para usar a CPU
- O comando `ps` informa o estado de cada processo em execução

Estados de um processo



1: Processo bloqueia esperando entrada
2: Escalonador escolhe este processo
3: Escalonador escolhe outro processo
4: Entrada disponível

O escalonador de processos no fim das contas é a base sobre a qual os processos são executados