Linux como Ambiente de Programação

Levy G. S. Galvão

1 Processos

- Olhando para processos
  - Identificadores de processos

Cada processo possui um identificador único, conhecido como PID. Cada PID é um número de 16-bits associados a de forma sequencial aos processo criados.

Cada processo possui um processo pai de origem, e que ao todo forma um esquema de árvore no Linux. Cada programa em C/C++ possui seu id de processo. Cada vez que o programa é chamado, um novo id será gerado.

- Visualizando processos ativos

O comando `ps` em sistemas UNIX permite observar todos os processos ativos e seus respectivos PIDs.

- Finalizando processos

Um processo pode ser finalizado ao invocar o comando `kill` seguido do seu PID.

- Criando processos
  - Usando `system`

A função `system` na biblioteca padrão do C permite criar um subprocesso de maneira fácil rodando o Bourne Shell e passa o comando para aquele Shell de execução do programa em C. Isso permite que comandos como `ls`, `ps` etc. sejam executados diretamente do cóodigo em C.

- Usando `fork` e `exec`

No Linux, a função `fork` permite a criação de um processo filho que é uma cópia exata do processo pai. Enquanto isso as funções da família `exec` fazem com que um processo mude de instância de um programa a outro. Ao combinar as duas funções, um novo processo é criado e possui sua instância alterada para que dependa de outro processo pai.

- Agendamento de processos

O Linux agenda processos pai e filhos de forma indepentende, assim não há garantia de qual irá executar primeiro, ou por quanto tempo serão executados

até serem interrompidos. Ele apenas garante que cada processo será executado eventualmente.

- Sinais

Sinais são utilizados para comunicação e manipulação de processos no Linux. Estes são assíncronos, de forma que quando um processo recebe um sinal, este é processado imediatamente. Os sinais possuem manipuladores que devem executar o mínimo de trabalho em resposta a um sinal e retornar ao prorgama principal. Manipuladores de sinais podem ser interrompidos por novos sinais, configurando uma condição de corrida, mas é um tanto raro.

- Terminação de processos

Um processo possui duas formas de ser terminado: o programa em execução solicita a terminação ou o programa chega ao final da execução. Cada processo possui um ID de saída que retorna ao processo pai.

- Esperando pela terminação de processos

Em situaçõe é desejável que o processo pai espere até que seus processos filho terminem. Isso é feito utilizando a família `wait` de funções.

- Chamada de sistema do tipo `wait`

A função `wait` bloqueia o processo pai até que seus filhos terminem de executar.

- Processos zumbis

Um processo filho torna-se um processo zumbi quando o processo pai não invoca a função `wait` para esperar seu término. Esse processo zumbi consiste em um processo que foi terminado, mas não foi limpado, pois como o processo pai não o esperou, seu id não foi obtido e a limpeza executada.

- Limpando processos filhos de forma assíncrona

No caso de processos zumbis, ainda possui uma solução para a sua limpeza. Uma alternativa é o processo pai chamar algumas funções como `wait3` ou `wait4` periodicamente. Estas diferem do `wait`, pois por enquanto que esta bloqueia até que algum filho termine, aquelas não o fazem, permitindo a função executar em modo de não bloqueio, assim limpando um processo filho se caso tenha terminado, mas se não o tiver, a função retornará. O retorno da chamada é o id do processo filho terminado. Caso não tenha processo para ser limpo, será retornado 0.

2 Desenvolvimento com o Kernel Linux

- Sobre Linus e a criação do Linux

O Linus Torvalds desenvolveu a primeira verão do Linux (UNIX like) em 1991, começando por um emulador de terminal e hoje se tornou um kernel altamente

usado, tanto em pequenas quanto grandes máquinas.

- O que torna o Kernel Linux único

O Linux usa uma arquitetura *microkernel* e um kernel monolítico. Também permite carregamento dinâmico de módulos do kernel. Ele também é *preemptive*, permitindo processos serem realocados durante execução. Principalmente, o Linux é gratuito.

- Diferenças ao programar no Kernel
- Não necessita de *libc* or cabeçalhos padrões;
- Usa o GNU C;
- Falta proteção à memória;
- Operações em ponto flutuante;
- *Stack* pequena de tamanho fixo e pré-processada;
- Gerenciamento de processos no Kernel

Um processo pode encontrar em estado de execucação ou estado pronto, mas não em execução. Quando em execução esta pode terminada. Mas quando pronta, pode entrar em execução. Eventos podem atuar modificando o estado do processo para adormecido ou execução. Processos de maior prioridade podem mudar estado de outros processos de menor prioridade.

- Threads no linux

Threads habilitam programação concorrente. O Linux possui uma implementação única de threads, pelo fato de não possuir este conceito, mas sim implementando-as como processos. Essas threads não possuem espaço de endereço, mas sim espaço no kernel.

- Agendamento de processos e algoritmos

Processos são programas ativos. No contexto do agendador, este escolhe qual processo é executado e por quanto tempo. Agendamento de processos é a base para um sistema operacional de múltiplos processos.

Policy é como o agendador de processos é conhecido. Processos podem ser limitados dentro do processador ou por meio de entradas e saídas.

- Chamada de sistema

Uma chamada de sistema é uma forma programática em que o computador requisita um serviço do kernel do OS em que ele é executado, assim interagindo diretamente com o OS. Essas chamadas são acessadas por chamadas de funções.

No Linux cada chamada de sistema possui um identificador númerico.

- Gerenciamento de interrupções pelo Kernel

Interrupções habilitam o *hardware* a sinalizar o processador de alguma tarefa à nível de *hardware*. Os valores de interrupção são comumnete chamados de linhas de solicitação de interrupção (IRQ). Cada IRQ possui um indentificador numérico.

O *interrupt handler* é responsável por atuar no kernel em meio à ocorrência de uma interrupção.

- Regições críticas e condições de corrida

Uma seção crítica é um segmento de código que acessa variáveis compartilhadas e deve ser executada em ação atômica (mais rápida possível e com único processo em execução).

Uma condição de corrida é uma condição indesejável que ocorre quando o dispositivo ou sistema tenta executar múltiplas operações ao mesmo tempo, mas devido sua natureza, as operações devem ser realizadas sequencialmente (não simultâneamente).

- Noção de tempo no Kernel

A passagem do tempo é importante para o kernel. Grande parte de suas funções são motivas pelo tempo e não por eventos.

O kernel deve trabalhar junto ao *hardware* para gerenciar melhor o tempo. O *hardware* possuir temporizadores que o kernel usa para te rnoção do tempo. O kernel também possui funções que lidam com interrupções por temporizadores.

- Nível de abstração do sistema de arquivos

Uma camada de abstração é uma forma de esconder os detalhes de execução de um subsistema de outro, facilitando a interoperabilidade.

A camada de abstração no sistema de arquivos permite o Linux trabalhar com diferentes tipos deles.

O VFS provê um modelo comum de arquivo para representar qualquer sistema de arquivos.