Advanced Linux Programming: Resumo do capítulo 3

Miguel Ravagnani de Carvalho - 12011EAU019

Sistemas Embarcados II

Uberlândia

Janeiro/2022

3. Resumo

3.1. Processos

Em sistemas linux, processos são instâncias de programas. Ao executar qualquer programa, um processo com ID próprio (process ID, ou *pid*). Através de comandos do terminal, é possível verificar informações sobre pids, assim como alterar permissões e até encerrar um processo.

O acesso ao *pid* é especialmente útil para o desenvolvimento à nível de *kernel*, então programas escritos em C ou C++ podem acessar informações sobre o processo através do arquivo <sys/types.h>, que define os tipos de dados utilizados em códigos fonte do sistema. Mesmo quando um mesmo programa é invocado mais de uma vez, cada instância possuirá um *pid* diferente.

O comando ps permite que o usuário acesse, no terminal, uma lista de processos atuais, e diferentes *flags* alteram a quantidade e formato das informações sobre os processos. Através do *pid*, é possível encerrar um processo utilizando o comando *kill.*

3.2. Criando Processos

De todas as maneiras de inicializar um processo, utilizar as funções de *system*, da biblioteca *standard library* para C, é uma maneira simples, mas pouco eficiente e arriscada. A função *system* cria um subprocesso em *Bourne shell* (*/bin/sh*), executando o comando passado como argumento, retornando o status de saída do comando. A dependência do *Bourne shell* significa que a segurança e usabilidade do comando está limitado ao *shell* do sistema, e há pouca garantia de que diferentes distribuições estarão utilizando o mesmo *shell*, uma vez que o próprio caminho */bin/sh* pode ser um link dinâmico para outro *shell*.

O segundo método é o uso de *fork* e *exec*. Quando *fork* é chamado, um processo duplicado, *child* process, é criado. O programa pode distinguir entre os dois processos através do valor de retorno, que é zero no caso do *child* process. Funções *exec* recebem um primeiro argumento que indica o caminho para o programa executado, assim como uma lista de argumentos (arg0, arg1, arg2 e etc) subsequentes, e podem substituir um processo por um outro programa, encerrando o processo anterior. Variações de *exec* podem encontrar programas pelo nome (enquanto para funções que não contém a letra *p* em seus nomes, é necessário fornecer o caminho até o programa alvo). Há funções com a letra *v* que fornecem um vetor de ponteiros não nulos, ou seja, uma lista de argumentos do programa em questão. Processos que chamam a funções *exec* para sua execução, uma vez que não há retorno, a não ser que um erro ocorra (em caso de erro, *exec* retorna -1).

É por isso que um processo que chama *exec* está comumente associado com uma primitiva *fork*. Como *fork* cria um processo filho praticamente idêntico (diferindo em *pid* e possíveis recursos em implementações diferentes) e o novo processo recebe o processo original como processo pai, que possui o *pid* original. Então, *exec* é utilizado para substituir o novo processo por um programa diferente, mas que ainda tem parentesco associado com o original.

3.3. Sinais

Sinais são interrupções de software, que são enviadas à um programa, e indicam que um evento importante ocorreu, variando desde requisições de usuário à acesso ilegal de memória. Um sinal comumente utilizado para interromper execuções de processos no *shell* é o atalho *CTRL+C*, que envia um sinal *SIGINT* ao processo, encerrando-o. Por isso, em um processo, o manipulador de sinais deve ter carga de trabalho mínima, uma vez que a natureza assíncrona e inesperada de um sinal pode interromper o processo em momentos vulneráveis. Recomenda-se evitar ações de *input* e *output* em um manipulador de sinais. Casos como esse tornam-se mais delicados em aplicações que utilizam de processamento paralelo e processamento concorrente.

3.4. Encerrando Processos

Um processo é encerrado quando este chama a função de saída ou *main* retorna. Em casos de um sinal de interrupção, o processo é encerrado. Processos com parentescos não necessariamente seguem uma ordem definida de execução, e isso ocorre devido a natureza *multitasking* de sistemas Linux. Então, caso seja necessaŕio que um processo pai espere um processo filho se encerrar, a função *wait* é recomendada.

Mas é possível que, enquanto um processo filho está ocorrendo, o processo pai, que chama *wait* seja encerrado externamente. Nesse caso, o processo filho é um processo zumbi. Neste caso, o processo pai não pôde limpar o processo filho, e então, traços deste ainda podem ser encontrados no sistema.

Para limpeza, existem sinais dedicados à alertar o processo pai sobre o fim de execução de um processo filho.