Unix - Controle de Tarefas pelo Terminal

O propósito básico de um Shell é a interpretação de comandos do terminal do usuário e a criação de processos para a sua execução. Um comando normalmente exige a criação de um único processo mas o uso de pipes (|) permite que uma única linha de comando exija a criação de vários processos. Um comando também pode se dividir em vários processos após a sua execução.

O controle de jobs foi idealizado para permitir que um usuário possa executar mais de um processo de cada vez. Um job consiste no conjunto dos processos relacionados a um comando ou aos comandos especificados em uma linha de comando com pipeline. Como todos os processos de um job são criados a partir do mesmo processo, eles estão no mesmo grupo de processos. O uso da função setpgid permite a troca do ID do grupo de processos pertencentes a cada job.

O Shell deve dar controle pleno do terminal (leitura) a somente um job de cada vez. Este é o job que executa em foreground. Todos os outros jobs executam em background. Um programa pode ser iniciado em background adicionando-se um & na linha de comando. Sua saída é direcionada ao terminal, possivelmente mixada com a de outros programas, mas ele não pode ler do terminal.

O Shell mantém uma tabela de jobs e o comando jobs lista todos os jobs em background relacionados na tabela de jobs, junto com seu número de job e o seu estado (interrompido ou em execução).

Processos em um job em foreground são terminados pelo caractere de controle (Ctrl+C), que envia o sinal SIGINT para todos os processos do job.

Uma tarefa em foreground pode ser interrompida digitando-se Ctrl+Z (SIGSTP) ao grupo de processos, retornado o controle do terminal ao Shell. Um job interrompido pode ser reiniciado com os comandos embutidos bg, como um job executando em background, ou fg, como um job executando em foreground. Nos dois casos, o Shell tem que redirecionar a E/S de forma apropriada e enviar o sinal SIGCONT aos processos envolvidos.

O comando kill embutido (não o /bin/kill) pode enviar sinais para jobs ou para processos. Jobs são identificados pelo ID de job, especificado por um % na frente do ID.

Dicas

1. Na implementação do Shell, as seguintes funções são importantes:
    a. waitpid
    b. kill
    c. fork
    d. execve
    e. setpgid
    f. sigprocmask
2. Quando implementar os gerenciadores de sinais, envie os sinais SIGTSTP e SIGINT para todos os processos do grupo em foreground usando –pid em vez de pid como argumento da função kill.
3. É interessante utilizar sigprocmask no parser para bloquear o sinal SIGCHLD antes de criar o processo filho e desbloquear este sinal novamente antes de adicionar o filho à tabela de jobs. Uma vez que os filhos herdam as máscaras de bloqueio dos pais, é importante que o filho desbloqueie o sinal SIGCHLD antes da execução de um novo programa. Quando um processo termina, toda a memória e recursos que ele usa são desalocados. Entretanto, a entrada na tabela de processos é mantida até que o processo pai leia o código de status de saída do filho, que pode ser feito com o uso da função wait. Enquanto um processo terminado permanece na tabela de controle de processos, ele é chamado de ZOMBIE. A função wait pode ser executada em resposta ao sinal SIGCHLD, que é enviado ao processo pai quando o filho termina.
4. Quando o Shell é executado a partir de um Shell UNIX padrão, o Shell faz parte do no grupo de processos em foreground do Shell UNIX. Se o Shell cria um filho, por default o filho também fará parte deste grupo. Digitando Ctrl+C, você envia o sinal SIGINT para cada um dos processos do grupo em foreground, o Shell também é terminado. Uma maneira de resolver isso é com a utilização da função setpgid(0,0) depois do fork mas antes do execve, o que coloca o processo filho em um novo grupo com ID igual ao PID do filho. Isto garante que somente o Shell estará no grupo de processos em foreground. Quando o Shell captura um Ctrl+C, ele deve redirecioná-lo para o job em foreground apropriado, isto é, o grupo de processos que representa o job em foreground.

Sinais mais Comuns

SIGABRT - processo abortado
SIGALRM - sinal levantado por alarme
SIGBUS - erro de barramento: "access to undefined portion of memory object"
SIGCHLD - processo filho terminado, parado (*ou continuado)
SIGCONT - continuar se parado
SIGFPE - exceção de ponto flutuante: "erroneous arithmetic operation"
SIGHUP - término de comunicação com terminal
SIGILL - instrução ilegal
SIGINT - interrupção
SIGKILL - kill
SIGPIPE - escrita em um pipe em que nenhum processo lê
SIGQUIT - quit
SIGSEGV - falha de segmentação
SIGSTOP - parada temporária na execução
SIGTERM - término
SIGTSTP - término enviado por um terminal de controle
SIGTTIN - tentativa de leitura ("in") de um processo em "background"
SIGTTOU - tentativa de escrita ("out") de um processo em "background"
SIGUSR1 - definida pelo usuário 1
SIGUSR2 - definida pelo usuário 2
SIGURG - dados urgentes disponíveis em um socket