



SISTEMAS EMBARCADOS II

Resumo Capítulo 3

VILSON CAMILO BORGES DE MORAES NEVES

12011EAU020

Uberlândia/MG

03 de março de 2024

Sumário

3 - Processes	3
3.1 – Looking at Processes	3
3.1.1 – Process ID's	3
3.1.2 - Viewing Active Processes	3
3.1.3 - Killing a Process	3
3.2 - Creating Processes	3
3.2.1 - Using <code>system</code>	4
3.2.2 - Using <code>fork</code> and <code>exec</code>	4
3.2.3 - Process Scheduling	4
3.4 - Process Termination	4
3.4.1 - Waiting for Process Termination	5
3.4.2 - The <code>wait</code> System Calls	5
3.4.3 - Zombie Processes	6
3.4.4 - Cleaning Up Children Asynchronously	6

3 - Processes

3.1 – Looking at Processes

Analizando os processos do computador.

3.1.1 – Process ID's

Há uma exclusividade nos processos do sistema Linux, referidos como *pid*. Eles têm característica de 16bit e quase todos têm um processo pai.

Foi executado um programa na linguagem C de impressão. Nele é observada a questão do Id de processo. Chamando o programa várias vezes, em cada uma delas um ID de processo diferente é relatado para cada processo novo. Ao contrário do que acontece caso chamando o programa sempre no mesmo shell, onde o ID permaneceria o mesmo.

3.1.2 - Viewing Active Processes

Esta sessão fala sobre a visualização dos processos ativos do sistema. A função *ps* exibe todos os processos em execução.

No exemplo de execução do livro, o comando mostrou dois processos referentes a execução do próprio programa.

Há um comando que mostra mais detalhes: “*ps -e -o pid*”

-e: instrui o ps a exibir todos os processos em execução.

-o *pid* e *ppid*: diz qual informação mostrar de cada processo.

3.1.3 - Killing a Process

O comando *kill* “mata” um processo, fazendo-o terminar ao menos que o mesmo esteja relacionado a um sinal SIGTERM

3.2 - Creating Processes

Duas técnicas serão comentadas nesta sessão para a criação de um novo processo, sendo uma simples e uma complexa, envolvendo rapidez, flexibilidade e segurança.

3.2.1 - Using system

A maneira mais fácil é usando esta função *system*, da biblioteca C padrão. O sistema cria um subprocesso e entrega o comando para o shell de execução.

Neste capítulo há uma execução demonstrativa onde chama o comando *ls* para exibir um conteúdo do diretório raiz.

A função *system* retorna o status de saída do comando shell.

3.2.2 - Using fork and exec

Quando é chamado *fork*, é criado um processo filho (*child*). O programa pai continua sendo executado a partir deste ponto em que esta função é chamada. E o processo filho executa também o mesmo programa do lugar.

Nesta sessão foi rodada uma aplicação demonstrativa onde o primeiro bloco de instrução “if” é executado apenas no processo pai, enquanto em “else” atua no processo filho.

Já as funções *exec* substituem o programa em execução por um processo em outro programa. Quando um programa chama *exec*, esse processo para imediatamente e começa a executar um novo programa do início.

3.2.3 - Process Scheduling

O Linux agenda os processos pai e filho independentemente, e não há garantia de qual será executado primeiro, ou por quanto tempo ele executará antes que o Linux o interrompa e deixe o outro processo (ou algum outro processo no sistema) é executado.

É possível especificar que um processo é menos importante (recebendo uma prioridade mais baixa) atribuindo a ele um valor mais alto.

Você pode usar o comando *renice* para alterar a gentileza de um processo em execução.

Para alterar a gentileza de um processo em execução, a função *nice* é utilizada. Seu argumento é um valor de incremento, que é adicionado ao valor de gentileza do processo que o chama.

3.4 - Process Termination

Normalmente, um processo termina com o programa em execução chamando a função de saída ou a função principal do programa retorna. Cada processo possui um código de saída:

um número que o processo retorna ao seu pai. O código de saída é o argumento passado para a função de saída ou o valor retornado de principal.

Um processo também pode terminar de forma anormal, em resposta a um sinal. Por exemplo, os sinais SIGBUS, SIGSEGV e SIGFPE fazem com que o processo seja encerrado. Outros sinais são usados para encerrar um processo explicitamente.

O sinal SIGINT é enviado a um processo quando o usuário tenta encerrá-lo digitando Ctrl + C em seu terminal. O sinal SIGTERM é enviado pelo comando kill. A disposição padrão para ambos é encerrar o processo. Ao chamar a função de aborto, um processo envia a si mesmo o Sinal SIGABRT, que termina o processo e produz um arquivo principal. O sinal de terminação mais poderoso é o SIGKILL, que termina um processo imediatamente e não pode ser bloqueado ou controlado por um programa.

Qualquer um desses sinais pode ser enviado usando o comando kill especificando um extra sinalizador de linha de comando.

Nesta sessão foi demonstrada aplicação onde, embora o tipo de parâmetro da função de saída seja int e a função retorne um int, o Linux não preserva os 32 bits completos do código de retorno. Dentro na verdade, deve-se usar códigos de saída apenas entre 0 e 127. Os códigos de saída acima de 128 têm um significado diferente, e quando um processo é encerrado por um sinal, seu código de saída é 128 mais o número do sinal.

3.4.1 - Waiting for Process Termination

Em algumas situações é desejável que o processo pai espere até um ou mais processos filho foram concluídos. Isso pode ser feito com a família de espera do sistema chamadas. Essas funções permitem a conclusão de um processo de execução e a habilitação do processo pai para recuperar informações sobre a rescisão de seu filho.

Existem quatro diferentes chamadas de sistema na família de espera, e opta-se por obter poucas ou muitas informações sobre o processo que foi encerrado ou a preocupação com qual processo filho foi encerrado.

3.4.2 - The *wait* System Calls

Analisando esta sessão, várias chamadas de sistema semelhantes estão disponíveis no Linux, que são mais flexíveis ou fornecem mais informações sobre a saída do processo filho. A função *waitpid* pode ser usada para aguardar a saída de um processo filho específico, em vez

de qualquer processo filho. A função *wait3* retorna estatísticas de uso da CPU sobre a saída do processo filho e o *wad* a função *wait4* permite que você especifique opções adicionais sobre quais processos aguardar

3.4.3 - Zombie Processes

Se um processo filho termina enquanto seu pai está chamando uma função de espera, o processo filho desaparece e seu status de término é passado para seu pai por meio da chamada de espera. Quando um processo filho termina e o pai não está chamando *wait*, ele não desaparece porque essas informações sobre seu encerramento seriam perdidas.

Em vez disso, quando um processo filho termina, ele se torna um processo zumbi (Zombie Process), que é um processo encerrado porém não limpo

Nesta sessão é rodado um código de demonstração onde lista-se os processos. É lista do processo pai e um outro processo zumbi.

3.4.4 - Cleaning Up Children Asynchronously

Esta sessão apresenta a limpeza dos processos filhos para que o processo pai continue a execução.

No script demonstrativo da sessão, o manipulador de sinal armazena o status de saída do processo filho em uma variável global, a partir do qual o programa principal possa acessá-lo