PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

INF 1316
Relatório de Atividade - Lab 5

2210872 Felipe Antelo Machado de Oliveira
2112171 Pedro Henrique de Oliveira Valentim Gomes

Objetivo:

O laboratório 5, ao todo, é um resumo do conteúdo de sinais POSIX em Linux, que busca escrever diferentes programas relacionados ao controle e tratamento de sinais. Cada exercício pede algo diferente com um objetivo diferente.

Questão 1 - Testar a captura dos sinais enviados por ctrl-c e ctrl-, sendo, respectivamente os SIGINT e SIGQUIT. Deve haver tratamento personalizados para suas funções.

Questão 2 - Tentar interceptar o sinal SIGKILL (kill -9) e verificar se é possível tratar ou ignorar esse sinal.

Questão 3 - Observar o comportamento de processos filhos em 4 diferentes situações e explicar o funcionamento do “filhocidio.c”, além de como o processo pai gerencia o tempo de execução.

Questão 4 - Fazer um programa “calculadora” capaz de realizar os 4 tipos de operações básicas entre 2 números. Devemos também observar o comportamento em divisões por zero e capturar sinais de erro de ponto flutuante, sendo esses chamados de (SIGFPE). O mesmo deve ser feito para números inteiros

Questão 5 - Implementar um código para a alternação de processos filhos a cada segundo em um looping infinito e, após 15 segundos, matar os filhos, interrompendo seus processos.

Estrutura do Programa :

Questão 1 - O código usa usa o signal(SIGINT, intHandler) e signal(SIGQUIT, quitHandler) para capturar sinais. Há um intHandler que imprime imprime "Você pressionou Ctrl-C", assim como um quitHandler que imprime "Terminando o processo..." e finaliza o programa com exit(0). Também é feito um loop infinito (for(EVER);) que mantém o programa ativo a todos os momentos.

Questão 2- É similar ao anterior com a adição de instalar um handler para SIGKILL usando signal(SIGKILL, handler). Importante observar que o programa entra em loop infinito (while(1)) imprimindo "Rodando..." a cada 1 segundo.

Questão 3 - O pai cria um filho usando fork(). O filho executa uma das seguintes possibilidades: Um loop eterno (for(EVER)), um sleep(3) ou um programa sleep5 ou sleep15 via execvp(). Enquanto isso, o pai responde ao aguardar um tempo (sleep(delay)), então se o filho ainda estiver ativo após o tempo, envia SIGKILL e captura a morte do filho com SIGCHLD e o trata no handler childhandler().

Questão 4 - Para essa questão, o código lê dois números reais (double) do usuário. Após isso, realiza e imprime as quatro operações básicas: soma, subtração, multiplicação e divisão. Por fim, instala um handler para SIGFPE com signal(SIGFPE, sigfpe_handler). As respectivas operações são feitas.

Questão 5 - Possuimos 2 programas para esse questão, no qual o “inifinito.c” é um auxiliar para manter o código em loop e o “coordenador.c” é o código central que garante o funcionamento. De forma resumida, após a criação dos 2 processos filhos com fork() e execvp() para rodar programas de loop infinito, O pai Usa kill(pid, SIGSTOP) para pausar um filho e usa kill(pid, SIGCONT) para continuar o outro. Existe uma alternancia da execução a cada 1 segundo, acompanhada de impressões que indicam quais programas estão rodando. Após 15 segundos, o pai mata ambos com SIGKILL e retorna uma mensagem final, finalizando o loop.

Solução do programa:

Questão 1 - Seguindo o desejado, montamos um programa no qual quando Ctrl-C é pressionado, o programa apenas imprime uma mensagem, sem ser encerrado.Após isso, quando Ctrl-\ é pressionado, o programa imprime a mensagem de término e encerra corretamente. Quando os comandos signal() são removidos, tanto Ctrl-C quanto Ctrl-\ encerram imediatamente o processo. Fica claro que a A função signal() é essencial para desviar o comportamento padrão dos sinais utilizados por nós na questão.

Questão 2- O programa busca agir como na questão anterior, agora com a intercepção do SIGKILL. Entretanto, apesar do handler ser instalado para outros sinais, o SIGKILL não pode ser capturado ou tratado. É somente ao enviar kill -9 <pid>, o processo é imediatamente terminado pelo sistema operacional (o que era esperado).

Questão 3 - //a- for(EVER) /* filho em loop eterno */

//filho vai entrar em loop eterno e o pai vai matar o filho quando acabar o

tempo especificado no delay

//b- sleep(3) /* filho dorme 3 segundos */

//filho vai dormir por 3 segundos e o pai vai matar o filho quando acabar o tempo especificado no delay

//c- execvp(sleep5) /* filho executa o programa sleep5 */

//filho vai executar o programa sleep5 q faz com que ele de sleep 5 e o pai vai matar o filho quando acabar o tempo especificado no delay

//d- execvp(sleep15) /* filho executa o programa sleep15 */

//filho vai executar o programa sleep15 q faz com que ele de sleep 15 e o pai vai matar o filho quando acabar o tempo especificado no delay

Vale observar que o execvp() substitui o processo filho pelo programa externo corretamente, além de que O tratamento de SIGCHLD garante a liberação de recursos e evita processos zumbis. Sendo assim, o controle de tempo do pai funciona como esperado

Questão 4 - Para a soma, subtração e multiplicação, as operações funcionam normalmente. Entretanto, ao realizar a divisão, se o divisor for 0, não ocorre a ativação do sinal SIGFPE, mas sim o resultado especial inf (infinito), permitindo que o programa continue a execução normalmente e imprima o valor inf.

Em outras palavras, para números do tipo float, a divisão por zero gera inf, sem a geração de SIGFPE.

Por outro lado, ao trabalhar com inteiros, onde a divisão por zero é inválida, ocorre a ativação do sinal SIGFPE, e, caso o programa não tenha um handler instalado, o processo é encerrado abruptamente.

Questão 5 - O processo pai alterna o controle dos filhos, simulando a ideia de "escalonamento por tempo". O controle é feito manualmente via sinais de parada e continuação. Existe um pequeno “delay” pelo tempo necessário para matar o programa com a utilização de SIGKILL não imediato (não utilização de kill -9)

Observação e conclusões:

As conclusões gerais que temos é que o laboratório gira em torno de como sinais controlam processos no Unix/Linux e a importância de tratar eventos assíncronos corretamente. Ou seja, é necessário saber Como criar processos filhos, gerenciá-los e encerrar sua execução de forma segura e controlada. Abaixo segue algumas observações feitas pela dupla ao longo dos diferentes exercícios que

possam ser interessantes de se observar:

- O uso de signal() permite modificar o comportamento padrão de sinais, possibilitando capturar, ignorar ou definir ações customizadas.

- Sem signal(), o sistema operacional adota o comportamento padrão para cada sinal, que geralmente resulta em terminação do processo.

- SIGINT (Ctrl-C) normalmente termina o processo, mas pode ser capturado para realizar ações alternativas (ex.: imprimir uma mensagem).

- SIGQUIT (Ctrl-\) termina o processo e pode gerar um core dump; também pode ser capturado para tratamento específico.

- SIGKILL (kill -9) é inevitável: não pode ser capturado, tratado nem ignorado. Ele é utilizado para forçar o encerramento de processos em qualquer estado.

- SIGFPE é gerado por erros de ponto flutuante, como divisão inteira por zero. Pode ser capturado para evitar que o programa termine abruptamente.

- Processos podem terminar sozinhos (ex.: sleep(3)) ou precisar ser forçados a terminar (SIGKILL).

- Usar fork() seguido de execvp() substitui o código do filho por um novo programa.

- Controle de processos com sinais de parada e continuação (SIGSTOP e SIGCONT) simula um escalonador de CPU manual.

- Preempção manual é feita alternando a execução dos filhos, controlando-os diretamente com sinais enviados pelo pai.

- O tratamento correto de sinais exige atenção à sincronização entre pai e filhos, principalmente em programas que criam múltiplos processos.

- Após o envio de SIGKILL aos filhos, o processo pai deve utilizar wait() para garantir que a memória dos processos mortos seja liberada corretamente (evitar processos zumbis), isso pode causar pequenos delays.