Insper

21 - Sinais II - bloqueio e revisão

Sistemas Hardware-Software - 2019/1

Igor Montagner

Parte 1 - mascarando sinais

Um desafio comum ao trabalhar com captura de sinais é sua natureza assíncrona: não só um handler de um sinal A pode ser interrompido pelo handler de um sinal B como nosso código também pode ser interrompido a qualquer momento.

Exercício conceitual: qual é a diferença entre *bloquear* e *ignorar* um sinal?

Bloquear voce acaba recebendo o sinal mas este nao pode influenciar no codigo no momento.
Ignorar voce recebe no entanto esse sinal é inutil para o seu codigo, portanto ele é ignorado

Abra o arquivo `signal-mask1.c` e analise seu código. Vamos observar o quê ocorre quando usamos variáveis globais compartilhadas entre um programa e seus handlers de sinal.

Exercício: Rode este programa e execute a seguinte sequência de comandos. Você precisará de dois terminais abertos e enviará sinais usando o comando `kill`. ***Se dois handlers usarem a mesma global pode dar problema.***

1. Envie o sinal SIGINT para o programa. O quê foi printado?

   O STATUS É 1

2. Envie o sinal SIGTERM para o programa. O quê foi printado?

   O STATUS É 2

3. Envie de novo SIGINT. Foi printado algo? Por quê?

   O STATUS É 3 ***SE VOCE CHAMAR NO MEIO DOS STATUS UM COMANDO O STATUS VAI AUMENTAR DE MANEIRA "ESTRANHA".***

Exercício: O campo `sa_mask` permite bloquear sinais enquanto os handlers executam. Modifique `signal-mask1.c` para que *SIGTERM* seja bloqueado enquanto o handler de *SIGINT* roda. Repita então o código acima e veja que não há mais conflito na variável global compartilhada.

Dica: Consulte `man sigsetops` para ver como preencher a variável `sa_mask`.

Exercício: Note, porém, que se invertermos a ordem dos envios de sinais do primeiro exercício ainda temos problemas! O que fizemos não permite que `SIGINT` seja interrompido por um `SIGTERM`, mas permite que um `SIGTERM` seja interrompido por um `SIGINT`! Corrija esta situação.

Vamos agora trabalhar com a chamada `sigprocmask`. Esta chamada permite que bloqueemos a recepção de um sinal em qualquer momento do programa. Isto é especialmente útil para casos em que uma porção do programa que executa códigos sensíveis a tempo não pode ser interrompida a não ser em casos extremos (`SIGKILL`, que não pode ser capturado). Veja o programa *signal-mask2.c*, mostrado abaixo.

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void funcao_importante() {
    sleep(20);
    printf("Fim da funcao!\n");
}

int main() {

    printf("Entrando em funcao_importante!\n");
    printf("O programa não será interrompido até seu fim!\n");
    // bloqueie os sinais

    // libere os sinais
    printf("Tente dar Ctrl+C agora!\n");

    sleep(50);

    return 0;
}
```

Exercício: Use a chamada `sigprocmask` para impedir que o programa receba um Ctrl+C durante a execução da `funcao_importante`.

Exercício: Pressione Ctrl+C durante a execução de `funcao_importante`. Escreva abaixo os prints mostrados neste experimento e explique o porque isto ocorre.

Parte 2 - revisão geral

Faremos uma revisão geral criando um mini bash. **Entrega (opcional)**: 23/05

1. Seu programa deverá imprimir o prompt abaixo e ler um comando a ser executado

   inspersh>
2. Cada comando é executado em um processo separado usando `fork + exec`.
   - Se o `exec` der problema mostre uma mensagemm de erro.
3. O processo do shell espera o comando terminar e imprime
   - Acabou com return: %d\n - valor de retorno do processo
   - Acabou com erro: %s\n - string com nome do erro
4. O shell bloqueia todos os sinais enviados, mas seus filhos devem recebê-los os sinais.
5. Se o shell receber o sinal `SIGUSR1` ele deverá terminar com a mensagem “Terminando shell”.
6. **Extra**: seu shell permite passar argumentos para os programas chamados.