Universidade Federal de Mato Grosso do Sul

Faculdade de Computação

UFMS

REDES DE COMPUTADORES

LABORATÓRIO

LAB -05

2 - 2012

profa. Hana K. S. Rubinsztejn - hana@facom.ufms.br

Conteúdo

- Introdução
- Tratamento de Sinais
  - Tratando Zumbis
- Exercício

Introdução

- Na aula passada foi visto como implementar um servidor concorrente multiprocesso utilizando *fork*().

Implementando Servidores Concorrentes - Multiprocessos

- **Código geral:**

```
1    pid_t pid;
2    for( ; ; )
3    {
4        connfd = accept(listenfd, ...);
5        if((pid = fork()) == 0)                    // Verifica se é pai ou filho
6        {
7                                                   // Só o filho executa esse código
8            close(listenfd);                       // Fecha o socket ouvinte
9            faz_alguma_coisa(connfd);              // Processa a solicitação
10           close(connfd);                         // Fecha a conexão
11           exit(0);
12       }
13       close(connfd);                             // O Pai fecha o socket conectado
14   }
```

Introdução

- Devemos tratar o término dos filhos criados pelo processo pai. Se não fizermos isso, eles ficarão na pilha de processos do Kernel como "zumbis" ou "defuntos".
- O propósito do estado zumbi é manter as informações sobre os filhos para que o pai busque-as posteriormente. Essas informações incluem o ID do Processo filho, seu status de término (sucesso, problemas) e as informações sobre a utilização de recursos pelo filho (tempo de CPU, memória, etc).
- Qual o problema de processos zumbis?
  - Ocupam espaço e podem acabar travando o servidor

Introdução

- Vamos fazer um exemplo e verificar isso.
  - Insira um sleep(20) no servidor de hora/data e conecte 3 clientes nele.
  - Em seguida, utilize o comando "***ps –ax | grep <nome_programa>***" para verificar os processos existentes.

Sinais

- Para resolver o problema devemos tratar os sinais gerados por estes processos de forma correta.
- Um **sinal** é uma notificação a um processo de que um evento ocorreu. As vezes, os sinais são chamados de ***interrupções de software***. Normalmente, eles costumam acontecer de forma assíncrona e podem ocorrer:
  - Entre os processos
  - do kernel do SO para um processo.

Tratando o Término dos Processos Filhos

- Ao terminar, um processo filho envia um sinal SIGCHLD ao seu processo pai.
  - No nosso exemplo, o que está acontecendo quando o processo filho termina de atender a requisição do cliente é que ele dispara um SIGCHLD e o pai não trata este sinal.
- Vamos introduzir o conceito de manipulador de sinais
  - função *signal*

Tratamento de sinais

- Função *signal*:

```
#include <sys/types.h>
#include <signal.h>

int signal(int signum, sighandler_t manipulador);
```

  - Esta função é utilizada para tratar sinais, onde "*signum*" é o sinal que estamos recebendo e "*manipulador*" é uma função manipuladora de sinais.
  - Retorno: o valor do manipulador se OK ou o sinal SIG_ERR, caso contrário.

Sinais

- SIGHUP (1) Corte: sinal emitido aos processos associados a um terminal quando este se ``desconecta''. Ele é também emitido a cada processo de um grupo quando o chefe termina sua execução.
- SIGINT (2) Interrupção: sinal emitido aos processos do terminal quando as teclas de interrupção (INTR ou CTRLc) do teclado são acionadas.
- SIGQUIT (3)* Abandono: idem com a tecla de abandono (QUIT ou CTRLD).
- SIGKILL (9) Destruição: arma absoluta para matar os processos. Não pode ser ignorada, nem interceptada (veja SIGTERM para uma morte mais suave para processos)

Sinais

- SIGSEGV (11)* Emitido em caso de violação da segmentação: tentativa de acesso a um dado fora do domínio de endereçamento do processo.
- SIGALRM (14) Relógio: emitido quando o relógio de um processo para. O relógio é colocado em funcionamento através da primitiva `alarm()`
- SIGTERM (15) Terminação por software: emitido quando o processo termina de maneira normal. Pode ainda ser utilizado quando o sistema quer por fim à execução de todos os processos ativos.
- **SIGCHLD (17) Morte de um filho: enviado ao pai pela terminação de um processo filho**
- SIGSTOP (19) Suspende: suspender processo

Tratando o Término dos Processos Filhos

- **Exemplo:** Processo filho termina sem o processo pai tratar seu término.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  int main()
4  {
5          if (fork() != 0)
6                  while(1) ;      // processo pai
7          exit(0);
}
```

Digitem: ps

Tratando o Término dos Processos Filhos

- **Exemplo:** O pai ignora o sinal SIGCHLD, e seu filho não vai mais se tornar um processo zumbi.

```
1  #include <stdio.h>
2  #include <signal.h>
3  #include <unistd.h>
4  int main()
5  {
6          signal(SIGCHLD,SIG_IGN) ;/* ignora o sinal SIGCHLD */
7          if (fork() != 0)
8                  while(1) ;
9          exit(0);
10 }
```

Digitem: ps

Tratando o Término dos Processos Filhos

- Toda vez que chamamos *fork*() para criar um filho, a função que deve ser chamada para impedir que eles se tornem zumbis é *waitpid*().
- Por isso vamos criar nossa função manipuladora que irá chamar *waitpid*() toda vez que o sinal SIGCHLD aparecer durante nosso programa.
- Vamos criar então uma função que vai ser utilizada pela ***signal***, para tratar nosso sinal:
  - `signal(SIGCHLD, sig_chld);`
  - onde `sig_chld()` é a nossa função que vai manipular este sinal.

Tratando o Término dos Processos Filhos

- Função: waitpid()

```
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *statloc, int opcoes);
```

- Retorno: O ID de processo se OK ou -1 caso contrario.
- O *statloc* contém o retorno do estado de terminação de sua execução: terminou de forma normal, abrupta ou foi exterminado por outro sinal.
- O pid usado na função indica por qual processo devemos esperar
  - -1 indica para esperar qualquer processo
- Em *opções*, o mais comum é o WNOHANG, que informa o sistema operacional para não bloquear enquanto não houver nenhum filho terminado.

Tratando o Término dos Processos Filhos

- Vamos agora recodificar nosso exemplo do servidor de hora para suportar de fato este conceito de sinais.
- Vamos adicionar os passos necessários e as modificações esperadas

1. Inclusão de cabeçalhos adicionais

```
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
```

  - Estes cabeçalhos são referentes a chamada usada por *signal*() e *waitpid*().

Tratando o Término dos Processos Filhos

2. Função manipuladora de sinais:

```
/* Trata o sinal enviado pelo sistema */
void sig_chld(int sinal)
{
   pid_t pid;
   int stat;

   while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
       printf("filho %d terminou.\n", pid);
   return;
}
```

A utilização de *printf*() ou qualquer coisa que use a saída e entrada padrões em manipuladores de sinais apenas são utilizadas para depuração.

Tratando o Término dos Processos Filhos

3. Modificação na *main*() do servidor

```
int main(int argc, char *argv[])
{
  int listenfd, connfd;
  ...

  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  ...

  bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
  listen(listenfd, LISTENQ);

  signal(SIGCHLD, sig_chld); /* vai tratar os filhos
                                zumbis */
  for( ; ; )
  ...
}
```

Exercícios

- a) Altere o servidor de hora concorrente para tratar a morte dos processos filhos utilizando *signal*
- **5) Entregar no moodle:**
  - Altere o Batepapo simples para que o cliente trate a interrupção do sistema dada por SIGINT.
  - Quando o usuário no cliente digitar ***ctrl^c*** deve ser perguntado ao usuário se ele realmente quer sair. Caso ele diga que sim, deve ser enviado ao servidor a palavra "sair" e o cliente deve terminar como se o usuário tivesse digitado "sair". Caso diga que não, deve continuar o batepapo normalmente.