Sistemas Operacionais I

Unix Pipes

Prof. Carlos Eduardo de B. Paes
Departamento de Computação
Pontifícia Universidade Católica de São Paulo

Introdução

- IPC - InterProcess Communication
- Pipes: forma mais antiga de IPC no Unix
- Historicamente half-duplex (unidirecional)
- Somente podem ser usadas entre processos com antecessor comum

PIPES

- Tradução: canalização
- Normalmente um pipe é criado por um processo que chama o fork e o utiliza entre pai e filho
- Comumente usados em shells para conecta a saída padrão de um processo com a entrada de outro

PIPES

- Exemplo: $ cat nomes | sort
  - ambos os comandos (cat e sort) executam concorrentemente
  - processo cat escreve o conteúdo que é lido por sort
  - o pipe automaticamente bufferiza o conteúdo
  - kernel pode suspender o processo que escreve no buffer caso fique cheio

PIPES

- Um pipe é criado pela função pipe:

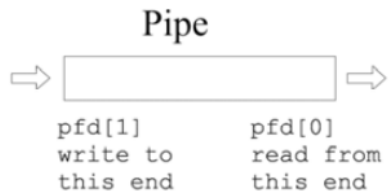
#include <unistd.h>

int pipe(int fd[2]);

- Retorna 0 se OK, -1 em caso de erro
- Dois descritores de arquivos são retornados no argumento:
  - fd[0] é aberto para leitura
  - fd[1] é aberto para escrita
  - A saída de fd[1] é a entrada de fd[0]

PIPES

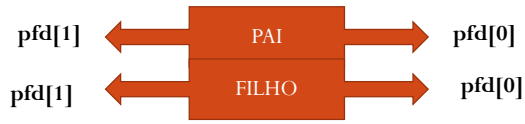
- E/S com pipes
  - Operações bloqueantes:
    - write(pfd[1], buf, size);
    - read(pfd[0], buf, size);
  - Operações Pipe em um único processo é praticamente inútil
  - Normalmente fork é chamado depois



PIPES

- Um pipe aberto antes do fork é compartilhado entre os dois processos
- Antes do fork

- Depois do fork

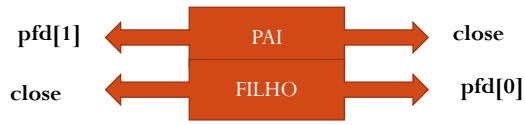


PIPES e Fork

- Dois extremos de leituras e dois de escrita
- Ambos processos podem escrever no pipe e ler do pipe
- Qual processo recebe o que é imprevisível
- Para um comportamento previsível um dos processos fecha sua entrada e outro sua saída
- Teremos um pipe simples de novo.

PIPE e Fork

- Se o pai quer escrever para um filho, o pai fecha seu fim de leitura e o filho fecha seu fim de escrita
- O pai escreve no pfd[1] e o filho lê do pfd[0]
- Quando o processo de comunicação acabar, o pai fecha seu fim de escrita. Filho recebe 0 na próxima leitura



Exemplo

```
#include <stdio.h>
#include <unistd.h>
#define MAXLINE 4096
int main(void)
{
        int n, fd[2];
        pid_t pid;
        char ine[MAXLINE];
        if (pipe(fd) < 0)
            printf("erro no pipe!\n");
        if ( (pid = fork()) < 0)
            printf("erro no fork\n");
        else if (pid > 0) {/* pai */
            close(fd[0]); /* fecha extremidade não usada*/
            write(fd[1], "hello world\n", 12);
        } else {/* filho */
            close(fd[1]); /* fecha extremidade não usada*/
            n = read(fd[0], line, MAXLINE);
            write(STDOUT_FILENO, line, n);
        }
        exit(0);
        }
```

PIPE e Fork

- Frases devem terminar com NULL (\0) ou nova linha
- Para comunicação bidirecional cria-se dois pipes
- Quando um escritor envia mais de uma mensagem de tamanho variável via pipe, ele deve ter um protocolo para indicar o fim da mensagem ao leitor

PIPE e Fork

- Exemplos de protocolos:
  - enviar o tamanho da mensagem (em bytes) antes de enviar a mensagem
  - terminar a mensagem com um caractere especial, tal como \n ou \0

Exec e dup

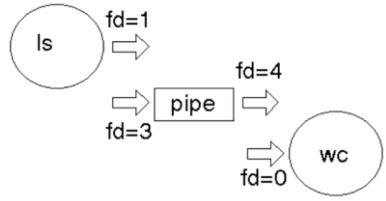
- Pipe funciona pois dois processos que sabem os descritores de cada extremo do pipe
- E se um processo se substitui com exec, como ele saberá o descritor?
- Normalmente processos recebem de stdin(0) e escrevem para stdout(1)

Exec e dup

- Exemplo: implementando em C "ls | wc"
  - Cria-se um pipe e chama-se o fork
  - pai chama exec para "ls" e filho chama exec para "wc"
  - "ls" normalmente escreve em 1 e "wc" normalmente lê de 0
  - Como associar a saída padrão com a saída de um pipe e a entrada padrão com a entrada de um pipe?

Exec e dup

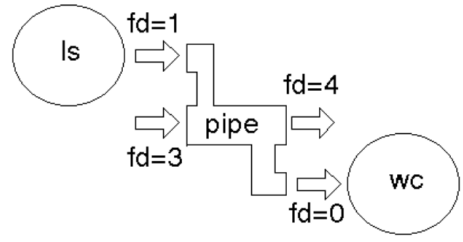
- Exemplo: implementando em C "ls | wc"



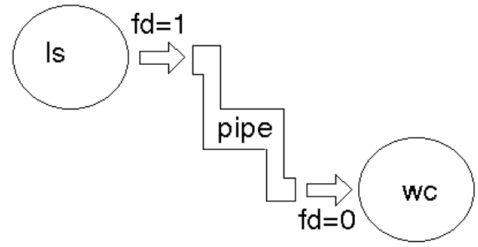
- Chamada de sistemas dup2 recebe um descritor existente e outro que "ele queira ser"
- No exemplo fd=3 gostaria de ser fd=1 e fd=4 gostaria de ser fd=0

Exec e dup

- Após dup2



- Após fechamento dos extremos não usados



Exemplo

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
    int pfd[2];
    pipe(pfd);
    if (fork() == 0) {
        close(pfd[1]);
        dup2(pfd[0], 0);
        close(pfd[0]);
        execlp("wc", "wc", (char *) 0);
    } else {
        close(pfd[0]);
        dup2(pfd[1], 1);
        close(pfd[1]);
        execlp("ls", "ls", (char *) 0);
    }
    exit(0);
}
```


Exercícios

1. Crie um processo filho que recebe uma mensagem “Sou filho” enviada pelo pai e apresenta na tela.
2. Crie dois processos pai e filho. O pai envia para o filho “ping” que retorna para o pai “pong”. As mensagens recebidas são mostradas na tela.
3. Usando pipes implemente uma versão C para a seqüência de comandos “ps | sed 1d | wc -l”.

Exercícios

4. Faça um programa que passa os argumentos recebidos na linha de comando via pipe para seu filho. O filho apresenta os argumentos na tela.
5. Faça um programa que recebe do usuário, via teclado um nome e uma idade. Esses dados são enviados via pipe para um processo filho que mostra os dados na tela.