Sistemas Distribuídos

FELIPE CUNHA

Introdução

Conceito de Sistemas Distribuídos:

Conjunto de computadores autônomos interligados em rede e executando programas com o objetivo de disponibilizar recursos aos usuários.

- Recursos: software, hardware e dados

Programação distribuída:

Programação distribuída consiste em se espalhar tarefas computacionais por vários programas, processos ou processadores.

Introdução

Aspectos importantes:

- Os computadores de um sistema distribuído são autônomos. Para haver um sistema distribuído TEM de haver REDE!
- Visão do usuário: um único sistema integrado

Introdução

Exemplos:

- Serviço de Arquivo ou de Impressão
- Aplicação Cliente/Servidor
- WWW (sistema distribuído de larga escala)

Sistema Centralizado: oposto de Sistema Distribuído

- Exemplo: sistema mainframe, acessado via “terminais burros”

Sistemas Distribuídos: Motivação e Vantagens

Motivação:

- Avanços na tecnologia de redes e de microprocessadores
- Mundo moderno possui aplicações inerentemente distribuídas (ex.: bancos)

Vantagens (sobre sistemas centralizados):

- Econômica: Microprocessadores oferecem uma relação custo/benefício melhor que a de mainframes ou computadores de grande porte.

Sistemas Distribuídos: Vantagens

Vantagens (sobre sistemas centralizados):

- Desempenho: Um sistema distribuído pode alcançar um desempenho (velocidade, throughput) maior que a de um único computador de grande porte.
- Algumas aplicações são inerentemente distribuídas: Envolvem diversos computadores autônomos.

Sistemas Distribuídos: Vantagens

Vantagens (sobre sistemas centralizados):

- Confiabilidade e tolerância a falhas: Se uma máquina trava ou sai da rede, o sistema como um todo pode continuar o processamento.
- Escalabilidade: O poder de computação pode ser incrementado aos poucos, independente de arquitetura ou de fabricante.

Sistemas Distribuídos: Desvantagens

Complexidade de desenvolvimento: Aplicações distribuídas envolvem diversos fatores tais como consistência, verificabilidade e sistemas de troca de mensagens inexistentes em aplicações centralizadas. A área ainda não está completamente desenvolvida.

Rede: A rede pode saturar ou causar outros problemas.

Segurança: Compartilhamento pode gerar acessos não autorizados a dados secretos.

Computação Paralela x Computação Distribuída

Computação Paralela

- Comunicação via memória compartilhada
- Objetivo: aumentar throughput (vazão) ou reduzir tempo de serviço

Computação Distribuída

- Descentralizar sistema
- Compartilhar recursos fisicamente dispersos
- Integrar recursos fisicamente dispersos

Computação Paralela x Computação Distribuída X Redes de Comunicação

Característica	Redes	S. D.	S. P.
Parece um único processador virtual?	Não	Sim	Sim
Todos tem que rodar um mesmo SO?	Não	Não	Sim
Quantas cópias do SO existem	Várias	Várias	1
Como é feita a comunicação?	Arquivos compartilhados	Troca de Mensagens	Memória compartilhada
É necessária padronização de protocolos?	Sim	Sim	Não
Existe apenas uma fila de tarefas?	Não	Não	Sim
Compartilhamento de arquivos tem semântica bem definida?	Normalmente não	Sim	Sim

Computação Distribuída

Comunicação via troca de mensagens

Troca de mensagens tem custo não desprezível, além de sujeita a falhas e atrasos

Nenhum nodo tem conhecimento do estado global do sistema em um dado instante

Objetivo: compartilhar recursos fisicamente separados

Sistemas Distribuídos: Principais Características

Compartilhamento de recursos

Transparência: “esconder” dos usuários a distribuição física dos recursos

- Acesso: recursos locais e remotos acessados igualmente
- Localização: usuários não sabem localização dos recursos
- Migração: recursos podem se mover sem trocar de nomes
- Replicação: usuários não sabem o nº de cópias de um recurso
- Concorrência: vários usuários compartilham recursos automaticamente
- Paralelismo: aplicações podem rodar em paralelo sem o conhecimento do usuário

Sistemas Distribuídos: Principais Características

Escalabilidade

- Aplicações não necessitam de mudanças quando o tamanho do sistema aumenta
- Contra-exemplos: DDD (inclusão do código da operadora), IPv4 (4 bytes) x IPv6 (16 bytes)

Tolerância a Falhas

- Disponibilidade: fração de tempo que o sistema está disponível
  - Importante: evitar estruturas centralizadas
- Tolerância a Falhas: capacidade de um sistema detectar uma falha e então proceder de uma das seguinte formas:
  - Terminar corretamente
  - Contornar a falha (usuários não conseguem percebê-la)

Sistemas Distribuídos: Principais Características

Abertura (Openess)

- Capacidade de ser estendido e interoperar com outros sistemas
- Importante: interfaces públicas e aderentes a padrões
- Padrões: HTTP, CGI, SMTP, ODBC, DNS

Tipos de programação distribuída

Distribuição de dados:

- Uma única tarefa em uma grande quantidade de dados.
- Dados são distribuídos entre processos.
- Todos os processos executam a mesma tarefa.
- Ex.: Descascar pilha de batatas.
  - Real: detecção de bordas

Distribuição de algoritmo:

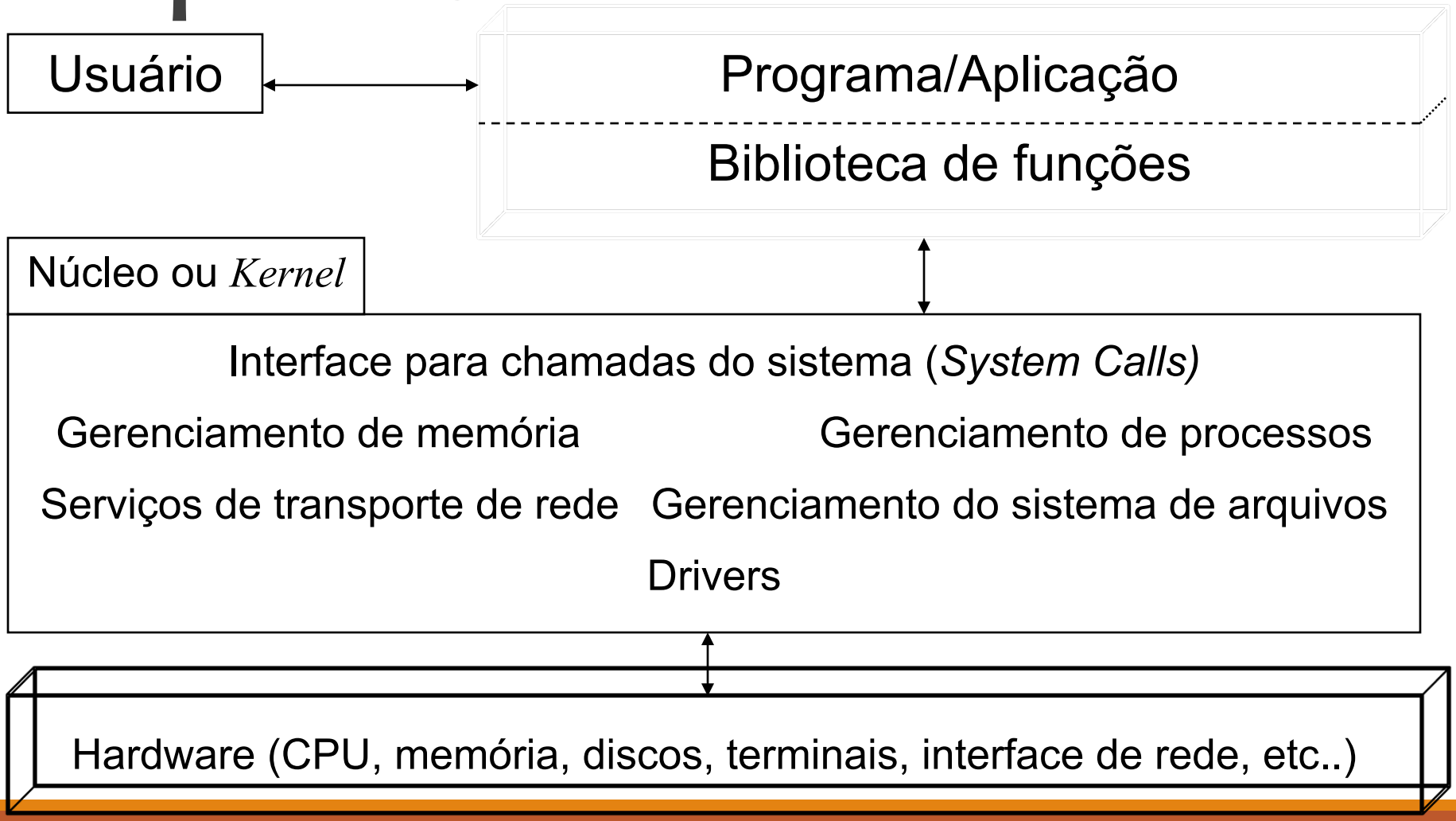
- Linha de produção (pipeline)
- Cada processo executa uma sub-tarefa específica que é parte de um todo.
- Ex.: Lavar pratos. Real: SGBD de 3 camadas.

Algoritmos Distribuídos

Características:

- Nenhum nodo tem acesso ao estado completo do sistema
- Nodos tomam decisões baseadas em seu estado local
- Comunicação via troca de mensagens
- Falha de um nodo não aborta o algoritmo
- Não assume-se existência de um relógio global

Usuário e o Sistema Operacional



Usuário e o Sistema Operacional

Kernel: Núcleo do sistema operacional.

- Drivers: executam o acesso direto ao hardware.
- Sistema de arquivos: prove interface para acesso e armazenamento dos dados.
- Gerenciamento de processos: responsável por escalonar e compartilhar os recursos e o tempo entre os processos.
- Gerenciamento de memória: trata do compartilhamento da memória física entre os processos.
- Serviços de transporte de rede: provê comunicação máquina-máquina ou processo-processo através da rede.

Processos e Programas

Programa:

- uma lista de instruções que especificam uma seqüência de comandos a serem executados.
- entidade passiva.

Processo:

- programa em execução.
- uma tarefa que tem controle sobre um espaço de endereçamento.
- entidade ativa.

Processos

Atributos:

- Process ID (PID): o identificador de processo é um inteiro único que identifica um processo.
- Parent Process ID (PPID): o identificador do processo pai possui o PID do processo que criou o novo processo.
- Real User ID (UID): identificador do usuário que iniciou o processo.
- Effective User ID: usuário que possui os direitos de acesso ao processo.
- Diretório corrente
- Tabela de file descriptors: identifica todos os canais de dado abertos (arquivos, pipes, FIFO, etc...)

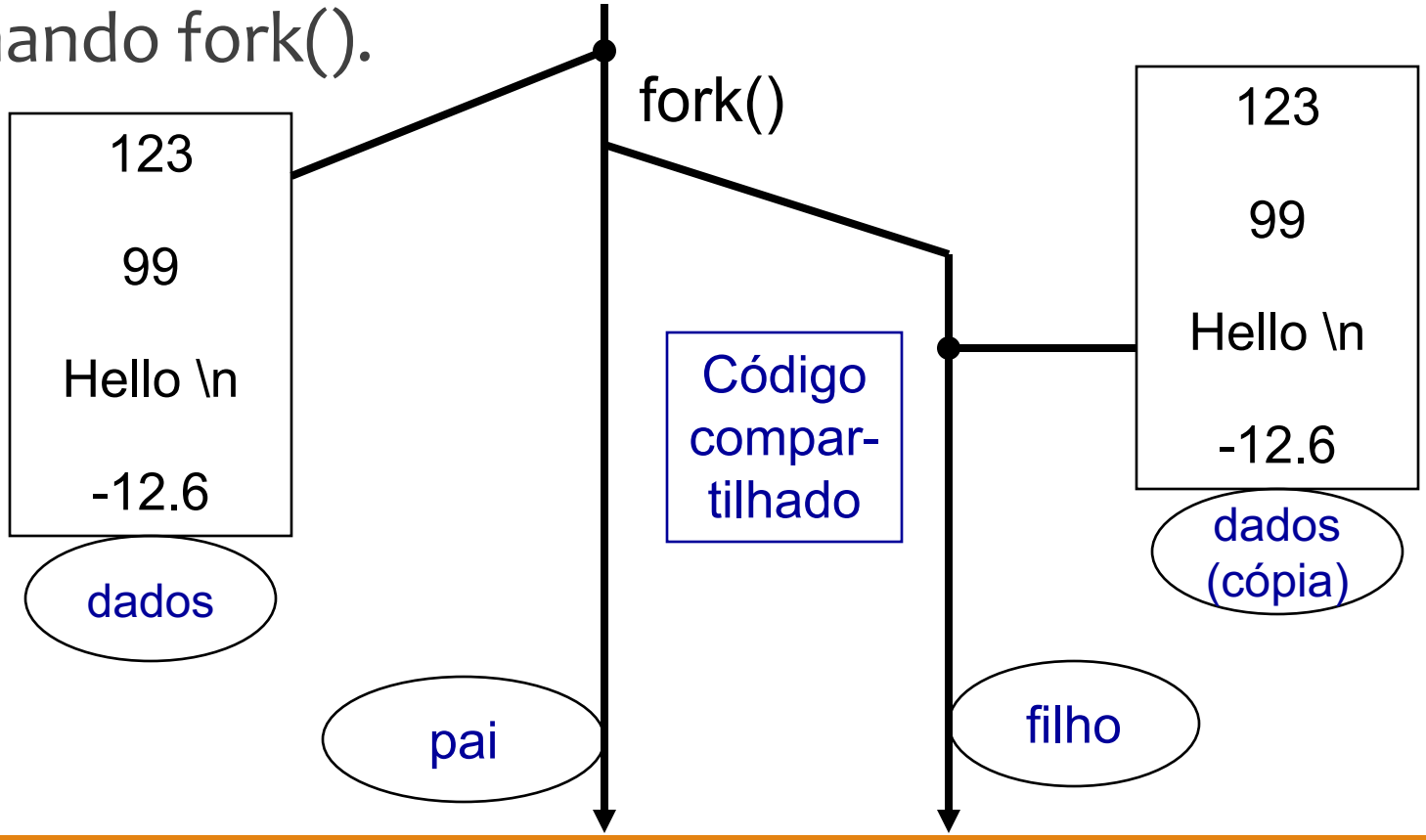
Processos

Atributos:

- Variáveis de Ambiente
- Espaço de código: região de memória onde o programa está carregado
- Espaço de dados: região de memória onde as variáveis globais e os dados estáticos estão carregados.
- Stack (pilha): região de memória para variáveis locais
- Heap: região de memória para dados alocados dinamicamente.
- Prioridade: parâmetro que controla o escalonamento do processo
- Disposição dos sinais: dados tipo flag que identificam a quais sinais o processo deverá responder

Processos Pai e Filho

Um processo pai cria um processo filho através do comando fork().



Processos Pai e Filho

Atributos herdados:

- PID $\rightarrow$ PPID
- Real UID, Effective UID, Variáveis de Ambiente

Atributos copiados:

- Dados estáticos, Pilha, Heap, File descriptors (se for ponteiro para arquivo é compartilhado), tratamento dos sinais

Atributos compartilhados:

- Código, ponteiros para arquivos.

Comunicação entre Processos

Comunicação entre processos (IPC) de um mesmo sistema centralizado: utiliza memória compartilhada

- Pipes
- FIFO
- Fila de Mensagens
- Memória Compartilhada

Pipes

Forma mais antiga de IPC (Unix, década 70)

Pipe: fila de comunicação unidirecional gerenciada pelo kernel

Criação de um pipe:

- int pipe (int d [2]);

Escrita e leitura

int write (d[1], “hello world”);

int read (d[0], buffer, tam_buffer);

Pipes

Pipes para comunicação em um mesmo processo não são de grande utilidade

Pipes entre processos diferentes:

- Criado por um processo pai
- Em seguida, pai executa um fork()
- Então, pipe é usado para comunicação entre pai e filho

Exemplo: Pai → Filho

```
int main () {
  int n, d[2];
  pid_t pid;
  char line [MAX_LINE];
  if (pipe (d) < 0)
     printf (“Erro na criação do pipe”);
  else ((pid = fork()) < 0)
     printf (“Erro no fork”);
```

Exemplo: Pai → Filho

```
else if (pid > 0) {  // processo pai: pid= PID filho
    close (d[0]);
    write (d[1], "hello world\n");
 }
 else { // processo filho: pid = 0
     close (d[1]);
     n= read (d[0], line, MAX_LINE);  // síncrona
     write (1, line, n);                          // 1 = stdout
 }
}  // main
```

Pipes

Fluxo bidirecional: dois pipes

Comando Unix: who | sort | lpr

Desvantagens:

- Unidirecional (half-duplex)
- Somente pode ser usado entre processos pai e filho

FIFO

First In, First Out

Também chamados de named pipes , ou seja, possuem um nome, o que permite que sejam usados entre processos que não tenham um ancestral comum.

Manipulados como se fossem arquivos

- Criação: int mkfifo (const char *name, mode_t mode)
- Outras funções: open, read, write, close

Fila de Mensagens

Lista de mensagens armazenada pelo kernel

Permite comunicações que não sejam FIFO

Mensagens:

```
struct {
long mtype;
char text [512];
}
```

Fila de Mensagens

Recebimento de Mensagens:

- int msgrcv (int msgid, void *ptr, size_t nbytes,
  long type, int flags)
- type = 0: retorna 1a msg (ordem FIFO)
- type > 0: retorna 1a msg cujo campo “mtype” for igual a “type”
- type < 0: retorna 1a msg cujo campo “mtype” é o menor valor menor que ou igual ao valor absoluto de “type”

Fila de Mensagens

Vantagens:

- Bidirecionais
- Recebimento por mensagens tendo como base seu tipo (nem sempre é FIFO)

Desvantagem:

- Comandos próprios para manipulação (diferentes dos comandos tradicionais de IO)

Memória Compartilhada

Exemplo: Envio de arquivos de um servidor para um cliente via pipe (ou outra estrutura)

Problema: número de cópias do arquivo

- 1a cópia: buffer de leitura do kernel para buffer do servidor
- 2a cópia: buffer do servidor para pipe
- 3a cópia: pipe para buffer do cliente
- 4a cópia: buffer do cliente para buffer de escrita do kernel

Memória Compartilhada

Solução: memória compartilhada entre cliente e servidor (MC)

Apenas duas cópias do arquivo:

- 1a cópia: do buffer de leitura do kernel para MC
- 2a cópia: da MC para buffer de escrita do kernel

Desvantagem:

- Sincronização a cargo do programador (via semáforos etc)

OBRIGADO!

Material de aula baseado nas aulas do prof.: Hugo, Max e Livro Couloris.