

Modelos para Computação Concorrente

Conceitos de Concorrência
VS
Ambientes e Linguagens

Fonte: material próprio

Fernando Luís Dotti

Noções básicas de Sistemas Operacionais

Sistema operacional como gerente do ciclo de vida de processos

- Criação de processos
 - aloca recursos
- Gerenciamento da execução
 - diversos processos em memória
 - execução: escalonamento
- Término:
 - desaloca recursos

Sistema operacional multi-programado

suporta múltiplos programas em
execução simultaneamente

Escalonamento

escolha de qual processo utiliza o processador em um momento

- diversos processos em memória, prontos para usar a CPU
- caso: mono e multi-processador

Justiça

- o escalonador DEVE garantir que todo processo, repetidas vezes, recebe o direito de executar
 - senão o escalonador está ERRADO
- assim, podemos supor **Justiça Fraca** de um escalonador
- a **Justiça Forte** não é garantida pelo escalonador
 - ele não "interpreta" o programa para saber se um processo, quando escalonado, está habilitado a prosseguir.
 - Isto será resolvido por mecanismos de sincronização de processos, usados pelo programador!

Relembrando

- Justiça Fraca:
 - se um processo está continuamente habilitado a prosseguir, ele progredirá
- Justiça Forte:
 - se um processo continuamente se torna habilitado a prosseguir, ele progredirá

Relembrando

- **Justiça Fraca:** o programa abaixo acaba ?

Algorithm: Stop the loop A	
integer $n \leftarrow 0$ boolean flag \leftarrow false	
p	q
p1: while flag = false p2: $n \leftarrow 1 - n$	q1: flag \leftarrow true q2:

- **Justiça Forte:** o programa abaixo acaba ?

Algorithm: Stop the loop A	
integer $n \leftarrow 0$ boolean flag \leftarrow false	
p	q
p1: while flag = false p2: $n \leftarrow 1 - n$	q1: await ($n==1$) q2: flag \leftarrow true

- Generalização dos conceitos para diversos ambientes e linguagens

Objetivo

Deixar claro que:

os conceitos de concorrência,
as formas de sincronização,
as propriedades, a forma de raciocínio
sobre sistemas concorrentes

são os mesmos

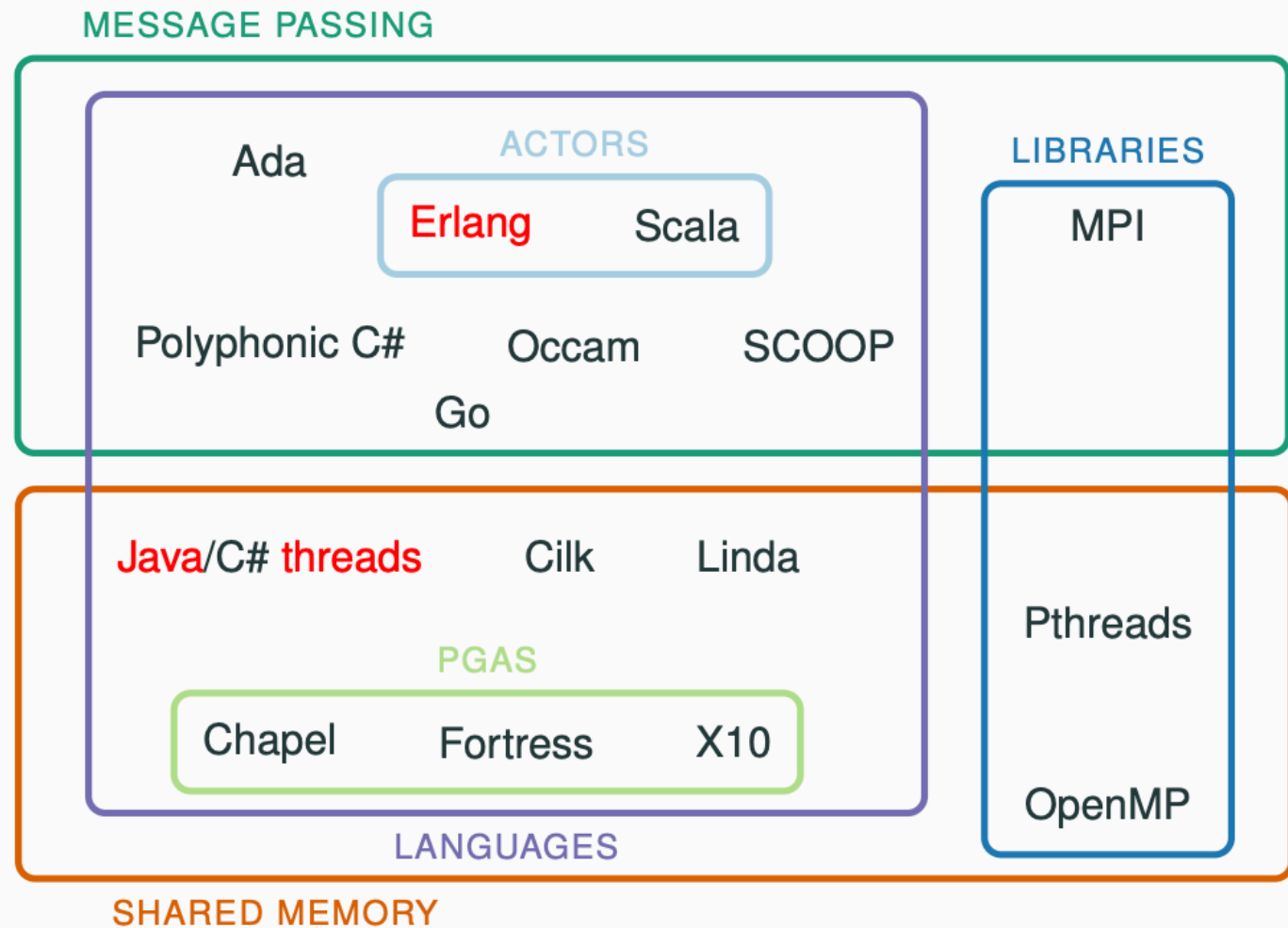
em

diferentes linguagens e plataformas

Conceitos

- Processos
 - compostos de passos sequenciais
 - Comunicam por canais e/ou Memória compartilhada
- Semântica de entrelaçamento
- Propriedades de canais
- Propriedades da memória compartilhada
- Raciocínio sobre atomicidade
- Propriedades de corretude e progresso

Concurrency languages galore



Muitas linguagens que suportam canais, no esquema acima, também suportam memória compartilhada

Linguagens e Plataformas

- Linguagens:
 - Go, Java, C , C++, C#, Python, **etc. etc.**
 - Suportam processos concorrentes e memória compartilhada
 - Canais podem ser suportados por alguma função de biblioteca adicional
- Ambientes
 - qualquer sistema operacional
 - redes quaisquer: rápidas, longa distância, ...

Exemplos

Em **um nodo** computacional:

- Go: processos, canais, variáveis compartilhadas
- Java: threads, modelo produtor/consumidor sobre memória compartilhada, objetos compartilhadas
- C: assim como java
- C: criação de processos a nível de S.O. com fork(), join(), comunicação com pipe(), declaração de memória compartilhada a nível de núcleo do sistema

Exemplos

Criação e finalização de processos

Exemplos

Go

go rotina()

estamos
vendo isso ...

```
package main

import (
    "fmt"
    "time"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go say("world")
    say("hello")
}
```

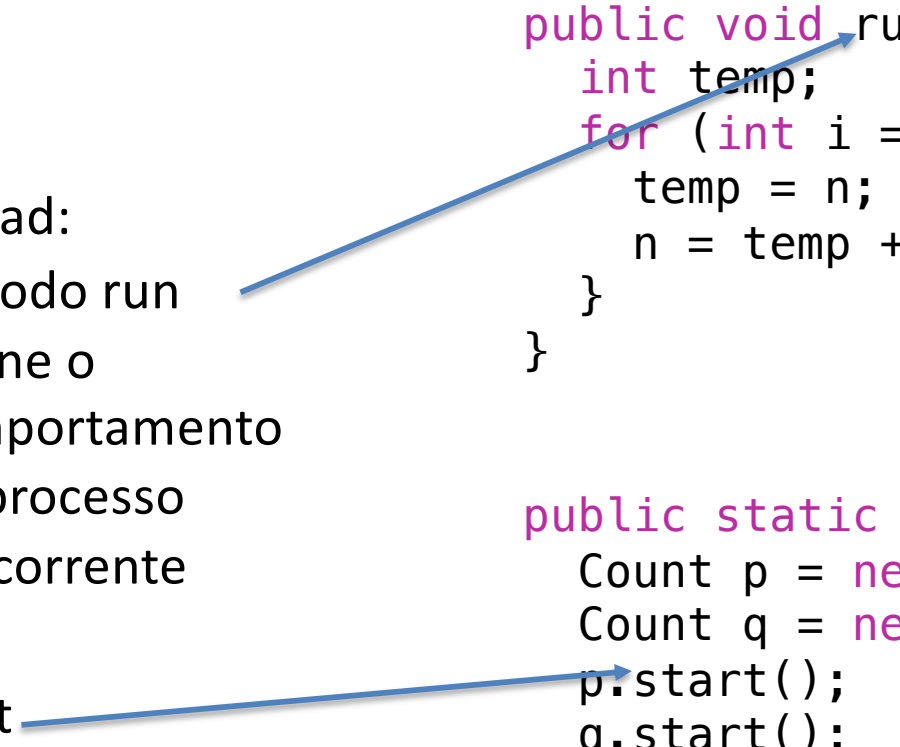

Exemplos

Java

thread:
método run
define o
comportamento
do processo
concorrente

start
inicia o
processo concorrente
executando run()

```
class Count extends Thread {  
    static volatile int n = 0;  
  
    public void run() {  
        int temp;  
        for (int i = 0; i < 10; i++) {  
            temp = n;  
            n = temp + 1;  
        }  
    }  
  
    public static void main(String[] args) {  
        Count p = new Count();  
        Count q = new Count();  
        p.start();  
        q.start();  
        try {  
            p.join();  
            q.join();  
        }  
        catch (InterruptedException e) { }  
        System.out.println("The value of n is " + n);  
    }  
}
```



Exemplos

C

thread:

um procedimento
define o
comportamento
de um processo
concorrente.

thr_create

inicia o
processo concorrente
executando o código
do procedimento
referenciado no
terceiro parâmetro (t0).

thr_join espera uma
thread acabar

```
#include <thread.h>
#include <synch.h>

thread_t tid1, tid2, tid3;

long a=0, b, c ;

void * t0(){
    long i ;
    for (i=0; i<1000000; i++){
        a = a + 5 ;
    }
    printf("Encerrei a t0 %d\n", sizeof(int));
}

void * t1(){
    long i ;
    for (i=0; i<1000000; i++)
    {
        a = a + 2;
    }
    printf("Encerrei a t1\n");
}

main(){
    int result, i ;
    printf("Eu sou a thread main\n");
    result=thr_create(NULL, 0, t0, NULL, THR_NEW_LWP, &tid1);
    result=thr_create(NULL, 0, t1, NULL, THR_NEW_LWP, &tid2);
    thr_join((thread_t)0, &tid3, NULL);
    printf("A primeira thread que morreu foi a %d\n", tid3);
    thr_join((thread_t)0, &tid3, NULL);
    printf("A segunda thread que morreu foi a %d\n", tid3);
    printf("O valor de a e: %d\n", a);
}
```

Exemplos

C com Fork/Wait

Processos

Pesados

Fork
clona processo;
filho é cópia do pai;
executa imediatamente,
a partir do ponto do fork.

pai e filho se
diferenciam somente
pelo pid;
não compartilham
nada.

```
#include <unistd.h>
main()
{
    int pid, pid2, pid3;
    printf("Antes do Fork – meu identificador
           de processo eh : %d\n", getpid());

    pid = fork();
    // processo filho ee copia identica do pai.
    // unica diferenca ee o valor de retorno pid:
    // para o pai, pid=id do filho, para filho pid=0
    // então o filho "nasce" executando depois do fork.
    printf("Depois do Fork – meu identificador
           de processo eh : %d\n", getpid());
    if (pid!=0)
    {
        printf("Eu sou o pai, meu pid: %d\n", getpid()) ;
        printf("Aqui deve ir o codigo soo do pai \n");
        pid2 = wait(0);
        printf("Esperei o filho morrer.Pid dele: %d\n", pid2);
    }
    else
    {
        printf("Eu sou o filho, meu pid eh: %d\n", getpid());
        printf("Aqui deve ir o codigo soo do filho \n");
        pid3 = wait(0);
        printf("Nao tenho filho, wait retorna -1
               imediatamete: %d\n", pid3);
    }
    printf("os dois executam isso \n");
}
```

Exemplos

comunicação entre processos

canais:

algumas linguagens.

Go, ML,

memória compartilhada:

todas linguagens que suportam threads

Go, Java, C, Python, **etc. etc.**

Exemplos

Em **mais de um nodo** computacional:

- Go, Java, C, outros:
- Em qualquer caso é possível usar bibliotecas para trocar mensagens entre processos em diferentes nodos computacionais
 - Sockets: padrão nos mais diversos ambientes
 - MPI: menos frequente - uso em mensagens para computação paralela em agregados de máquinas

Exemplos

Go usando sockets

```
package main

import "net"
import "fmt"
import "bufio"
import "strings" // only needed below for sample processing

func main() { // SERVER
    fmt.Println("Launching server...")
    ln, _ := net.Listen("tcp", ":8081") // listen on all interfaces
    conn, _ := ln.Accept() // accept connection on port
    for { // run loop forever (or until ctrl-c)
        // will listen for message to process ending in newline (\n)
        message, _ := bufio.NewReader(conn).ReadString('\n')
        // output message received
        fmt.Print("Message Received:", string(message))
        // sample process for string received
        newmessage := strings.ToUpper(message)
        // send new string back to client
        conn.Write([]byte(newmessage + "\n"))
    }
}

.....

func main() { // CLIENT
    // connect to this socket
    conn, _ := net.Dial("tcp", "127.0.0.1:8081")
    for {
        // read in input from stdin
        reader := bufio.NewReader(os.Stdin)
        fmt.Print("Text to send: ")
        text, _ := reader.ReadString('\n')
        // send to socket
        fmt.Fprintf(conn, text + "\n")
        // listen for reply
        message, _ := bufio.NewReader(conn).ReadString('\n')
        fmt.Print("Message from server: "+message)
    }
}
```

Exemplos

C com MPI

nro processos

definido no

ambiente MPI;

cada processo

tem um rank;

conforme rank

tem um

comportamento;

processos em nodos

diferentes. usam

send/receive

```
#include <stdio.h>
#include "mpi.h"
main(int argc, char** argv)
{
    int my_rank; /* Identificador do processo */
    int proc_n; /* Número de processos */
    int source; /* Identificador do proc.origem */
    int dest; /* Identificador do proc. destino */
    int tag = 50; /* Tag para as mensagens */

    char message[100]; /* Buffer para as mensagens */
    MPI_Status status; /* Status de retorno */
    MPI_Init (&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank != 0)
    {
        sprintf(message, "Greetings from process %d!", my_rank);
        dest = 0;
        MPI_Send (message, strlen(message)+1, MPI_CHAR, dest,
                  tag, MPI_COMM_WORLD);
    }
    else
    {
        for (source = 1; source < proc_n; source++)
        {
            MPI_Recv (message, 100, MPI_CHAR, source, tag,
                      MPI_COMM_WORLD, &status);
            printf("%s\n", message);
        }
    }
    MPI_Finalize();
}
```

Questões comuns

- Em **todos os exemplos**, os mesmos conceitos de concorrência se aplicam igualmente:
 - atomicidade
 - entrelaçamento,
 - espaço de estados, transições
 - seção crítica, uso de soluções de sw, hw, semáforos, monitores
 - deadlock, livelock, starvation, justiça

Criação e finalização de threads

- Em **todos os exemplos**, os mesmos conceitos de concorrência se aplicam igualmente:
 - atomicidade
 - entrelaçamento,
 - espaço de estados, transições
 - seção crítica, uso de soluções de sw, hw, semáforos, monitores
 - deadlock, livelock, starvation, justiça