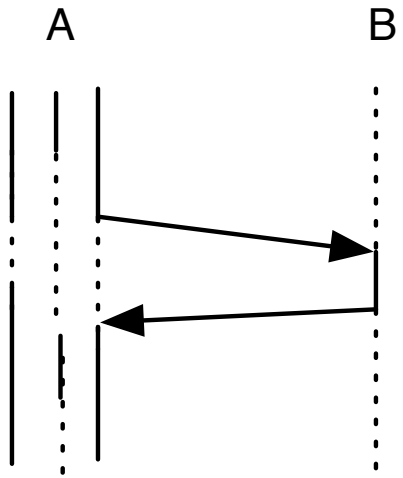
Sistemas Distribuídos

Chamada Remota de Procedimento – cont.

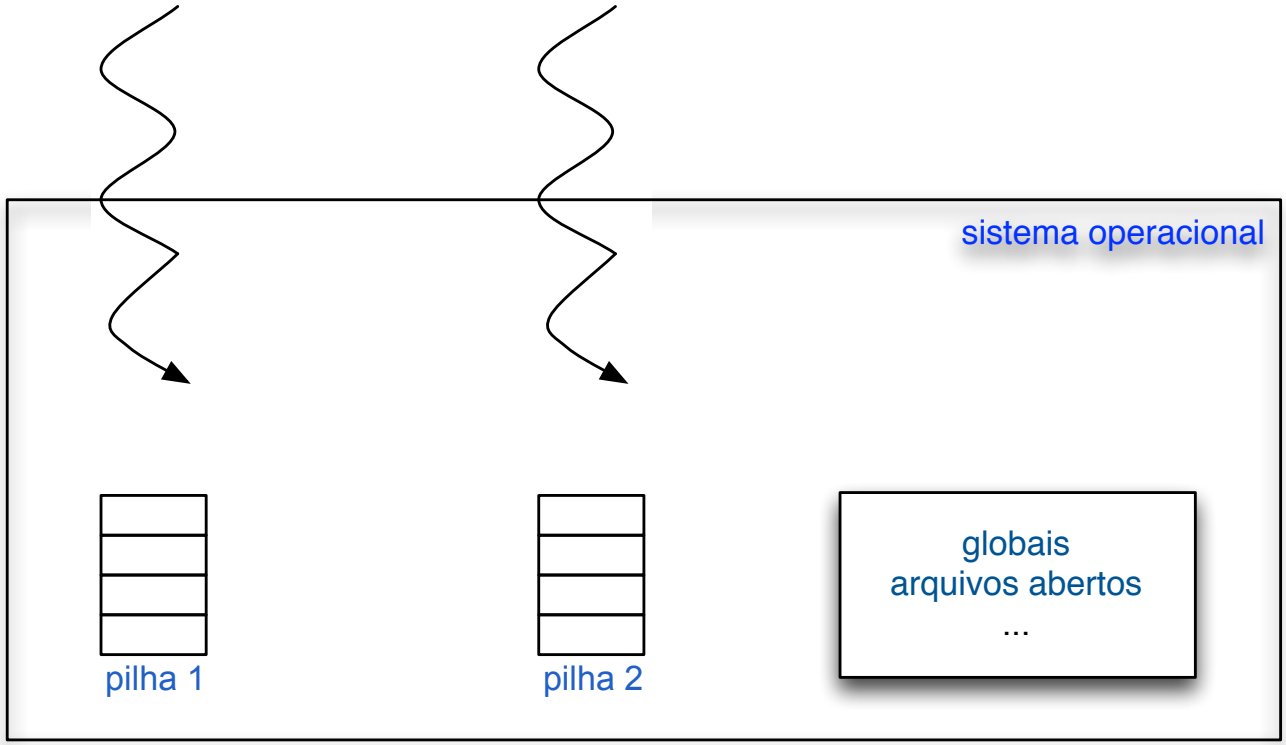
março de 2021

RPC: concorrência

- como sobrepor tempo de chamada com processamento?

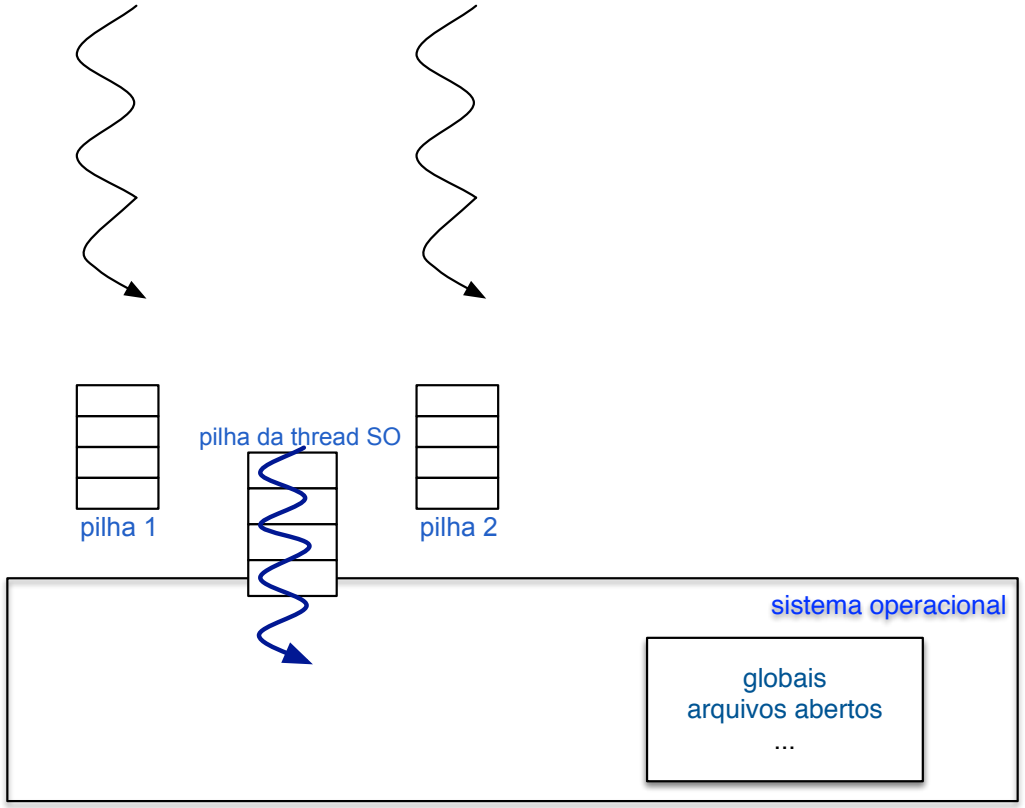


Threads de sistema operacional



RPC: concorrência

- como sobrepor tempo de chamada com processamento?
  - outra opção: threads cooperativos



Corotinas assimétricas: Lua

```
function ping ()
  for i = 1,10 do
    print ("ping", i)
    coroutine.yield()
  end
  print("fim")
end

co = coroutine.create (ping)
while coroutine.resume(co) do
  print("pong")
end
```

Escalonamento cooperativo

- pontos de possíveis entrelaçamento ficam explícitos no código
- por outro lado, necessidade de transferência explícita

- tipicamente transferência é encapsulada em chamada a biblioteca

Escalonamento cooperativo

```
tasks = {}
...
function ping ()
  i = 0
  for i = 1, 10 do
     print ("ping --", i)
     i = i+1
     coroutine.yield()
  end
end
function pong ()
  i = 0
  for i = 1, 10 do
     print ("pong --", i)
     i = i+1
     coroutine.yield()
  end
end
create_task (ping)
create_task (pong)
dispatcher()
```

RPC: cenários pouco acoplados

problemas com sincronização causada pelo bloqueio

- escalabilidade de threads de OS
- coleta de lixo: threads a espera de servidores que falham
- sobrecarga de preempção

RPC e assincronismo

- oneway
- chamadas assícronas
  - futuros
  - chamadas assíncronas com callbacks

Futuros

- chamada retorna imediatamente devolvendo um descritor
- descritor usado posteriormente para sincronização
- muito popular atualmente!

Futuros em C++11

```
// a non-optimized way of checking for prime numbers:
bool is_prime (int x) {
  std::cout << "Calculating. Please, wait...\n";
  for (int i=2; i<x; ++i) if (x%i==0) return false;
  return true;
}
int main () {
  // call is_prime(313222313) asynchronously:
  std::future<bool> fut = std::async (is_prime,313222313);
  std::cout << "Checking whether 313222313 is prime.\n";
  // ...
  bool ret = fut.get();      // waits for is_prime to return
  if (ret) std::cout << "It is prime!\n";
  else std::cout << "It is not prime.\n";
  return 0;
}
```

Futuros em ProActive

```
m1 = m0.getBlock (... );
m2 = m0.getBlock (...);

m1 = (Matrix) Javall.turnActive(m1, remoteNode);
m2 = (Matrix) Javall.turnActive(m2, localNode);

// Computes right products
v1 = m1.rightProduct(v0);
v2 = m2.rightProduct(v0);

//Creates result vector
v3 = v1.concat(v2);
```

http://proactive.activeeon.com/programming/

Futuros e avaliação postergada

- objetos retornados por operações assíncronas podem ser passados como argumentos em novas operações
- otimização da transferência de dados

```
v1 = a.foo (...); // chamada assíncrona
v2 = a.bar(...); // chamada assíncrona
...
v1.f(v2)
```

RPC assíncrona com callback

- chamada retorna imediatamente
- retorno dispara execução de *callback*
  - em alguns casos, *callback* especificada na chamada
- casamento com modelo de execução em uso

Chamada assíncrona com callback "em Lua"

```
function collect(val)
  acc = acc + val
  repl = repl + 1
  if (repl==expected) then print ("Current Value: ",
                                  acc/repl)
  end
end
function askvals (peers)
  repl = 0; expected = 0; acc = 0
  for p in pairs (peers) do
    expected = expected + 1
    p:currValue({callback=collect})
  end
end
```

- supondo uma infraestrutura orientada a eventos

Chamada assíncrona com callback "em Lua"

```
function collect(val)
  acc = acc + val
  repl = repl + 1
  if (repl==expected) then print ("Current Value: ",
                                  acc/repl)
  end
end
function askvals (peers)
  repl = 0; expected = 0; acc = 0
  for p in pairs (peers) do
    expected = expected + 1
    p:currValue{callback=collect}
  end
end
```

- estado registrado em globais
- e se novo pedido for realizado antes do primeiro estar completo?

Chamada assíncrona "em Lua" com closures

```
function askvals(peers)
  local acc, repl, expected = 0, 0, 0
---------- escopo lexico
  function collect (val)
    repl = repl+1
    acc = acc + val
    if (repl==expected) then print ("Current Value: ",
                                    acc/repl)
    end
  end
-------------
  for p in pairs (peers) do
    expected = expected + 1
    p:currValue{callback=collect}
  end
end
```

RPC: outras críticas

tolerância a falhas

- propostas de *multicast*

RPC: outras críticas

acoplamento cliente-servidor também espacial

- identificação de servidor que deve tratar a requisição
- endereço bem conhecido funciona bem em ambientes controlados