Exercícios sobre Processos em Elixir

prof. André Rauber Du Bois

Universidade Federal de Pelotas
dubois@inf.ufpel.edu.br

1 Questionário

1. Escreva um processo que calcula a área de diferentes formas geométricas. Exemplos de mensagens que ele poderia tratar:
   - `{:quadrado, lado}`: calcula a área de um quadrado
   - `{:retangulo, base, altura}`: calcula a área de um retângulo
   - `{:circulo, raio}`: calcula a área de um círculo
   - `:die`: encerra o processo
2. Fazer um script Elixir com três processos que se comunicam conforme a Figura 1. Ao iniciar, o Processo 1 sabe apenas o pid do Processo 2, e o Processo 2 sabe apenas o pid do Processo 3. O objetivo do programa é que o pid do Processo 1 chegue até o Processo 3 através do Processo 2. Por último, o Processo 3 deve enviar uma mensagem para o Processo 1.

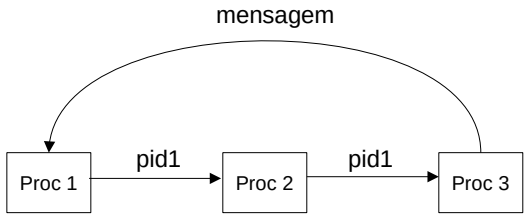

Figure1. Comunicação entre processos que não se conhecem

3. O fibonnaci de um número `n` é sempre a soma dos dois números anteriores na sequência:

   ```
   fib(n) = fib(n-1) + fib(n-2)
   ```

   Implemente uma função que calcula um número da sequência fibonaci usando dois processos para o cálculo dos números anteriores da sequência, ou seja, se a função deve calcular o fibonacci de `n`, ela cria um processo para calcular o fibonacci `n-1` e outro para calcular o de `n-2`. A função devolve

como resposta a soma dos resultados enviados pelos dois processos. Os processos devem calcular o fibonacci usando alguma das versões sequenciais vistas em aula. Teste essa função. Será que existe algum número da sequência que o seu cálculo é mais rápido com dois processos do que usando o cálculo sequencial?

4. Uma maneira de se simular uma memória compartilhada em Elixir é através do uso de um processo que armazena e compartilha o estado corrente da memória. O objetivo deste exercício é implementar um processo que simula variáveis compartilhadas. O processo guarda uma lista de tuplas do tipo `{var,value}`, que contém o nome de uma variável (a string `var`) e o estado corrente dessa variável (`value`). Esse processo deve saber tratar as seguintes mensagens:
   - `{:new, str_var, initial_value}`: a mensagem `:new` contém uma string com o nome da nova variável (`str_var`) e o estado inicial da variável(`initial_value`). Caso a variável já esteja na lista de variáveis, o processo responde com `:ja_existe`, caso contrário, a nova variável é adicionada na lista de variáveis e o processo responde com `:ok`.
   - `{:read, str_var}`: a mensagem `:read` contém o nome da variável a ser lida (`str_var`). Caso a variável exista na lista, o processo responde com `{:ok, conteudo_atual}`, caso contrário, responde com `:var_nao_existe`
   - `{:write, str_var, new_value }`: a mensagem `:write` contém o nome da variável a ser escrita (`str_var`) e seu novo valor (`new_value`). Caso a variável exista na lista, o processo faz um update na lista de variáveis e responde com `:ok`, caso contrário, responde com `:var_nao_existe`.
   - `:die`: o processo encerra

   Implementar também um cliente que use a memória comparitlhada.