4º ATIVIDADE de CES-27 / 2018

CTA - ITA - IEC

Prof Juliana e Prof Vitor

Objetivo: Trabalhar com algoritmo de exclusão mútua para sistemas distribuídos.

Entregar (através do TIDIA): Códigos dos exercícios (arquivos .go) e relatório. O relatório deve apresentar o código, explicar detalhes particulares/críticos do código, apresentar testes realizados e comentar os resultados.

Tarefa: Implemente o Algoritmo de Ricart-Agrawala para exclusão mútua.

Obs: CS = *critical section*

Requisitos:

1) Vamos rodar o sistema como abaixo descrito. Neste exemplo, temos 3 processos e um recurso compartilhado:

- Terminal A: `SharedResource`
- Terminal B: `Process 1 :10002 :10003 :10004`
- Terminal C: `Process 2 :10002 :10003 :10004`
- Terminal D: `Process 3 :10002 :10003 :10004`

Obs: Temos sempre a mesma sequencia de portas. Cada processo tem seu *id*. De acordo com o *id*, o processo sabe sua porta e a dos colegas. Ex: o processo 1 usa tem porta 10002, o processo 2 tem porta 10003, e o processo 4 tem porta 10004.

Obs: O algoritmo de Ricart-Agrawala é implementado no processo.

2) O recurso compartilhado será representado por um processo também, chamado `SharedResource.go`. Basicamente o `SharedResource` fica esperando alguém mandar uma mensagem para ele, através de uma porta fixa (ex: 10001). A mensagem recebida indica o processo (*id*) que a enviou, o relógio lógico desse processo (no momento da requisição), e um texto simples (ex: "CS sugou"). O código do `SharedResource` é simples como indicado abaixo.

```
func main() {
        Address, err := net.ResolveUDPAddr("udp", ":10001")
        CheckError(err)
        Connection, err := net.ListenUDP("udp", Address)
        CheckError(err)
        defer Connection.Close()
        for {
            //Loop infinito para receber mensagem e escrever todo
            //conteúdo (processo que enviou, seu relógio e texto)
            //na tela
           //FALTA FAZER
        }
}
```

3) O processo deve "escutar" o teclado (terminal).

   3.1) Caso receba a mensagem "x", o processo deve solicitar acesso à CS, em seguida usar a CS e liberar a CS.
   Obs: Caso o processo receba "x" indevido, ele deve imprimir na tela "x ignorado". O "x"é indevido quando o processo já está na CS ou esperando para obter a CS.

   3.2) Caso receba o seu *id*, o processo executa uma ação interna (apenas um incremento do seu relógio lógico).
   Obs: Isso já foi feito na "Atividade 2". Aqui serve apenas para testarmos o algoritmo depois com relógios distintos para os processos.

4) "Usar a CS" significa simplesmente enviar uma mensagem para o `SharedResource` e dormir um pouco.
   Relembrando: A mensagem deve ter o processo (*id*) que a enviou, o relógio lógico desse processo (no momento da requisição), e um texto simples (ex: "CS sugou").
   Obs: Antes de "usar a CS", o processo deve escrever na tela "Entrei na CS".
   Obs: Depois de "usar a CS", o processo deve escrever na tela "Sai da CS".

5) O processo deve sempre ser capaz de receber mensagens dos outros processos.
   Obs: Assim o processo pode estar esperando a CS, mas receber mensagem de outro solicitando a CS.
   Obs: Provavelmente você terá que usar `goroutine` para a função `doServerJob`. E ainda essa função terá um loop infinito para receber mensagens dos outros processos.

Para o relatório ⇨Teste o sistema assim com três processos.

- Elabore um caso com um processo solicitando a CS e, depois que ele liberar, outro processo solicita a CS. Mostre o esquema (figura) do resultado esperado e apresente o resultado obtido (telas). Explique/comente.

Para o relatório ⇨Teste o sistema assim com cinco (ou mais) processos.

- Elabore um caso com processos solicitando a CS "simultaneamente". Mostre o esquema (figura) do resultado esperado e apresente o resultado obtido (telas). Explique/comente.

Bom trabalho!