Relatório Trabalho Prático Sistemas Distribuídos

Nomes: Gabriel Petry, Guilherme Kuhn, João Pedro Demari

Introdução:

Como nos foi passado em atividade, o intuito desse trabalho é implementar o algoritmo DIMEX, que propõe uma solução para exclusão mútua em sistemas distribuídos. O algoritmo garante que processos distribuídos possam coordenar o acesso à seção crítica de forma eficiente e segura, evitando conflitos e garantindo a integridade do sistema.

O algoritmo DIMEX:

Em nosso código, seguimos a descrição da implementação apresentada em aula presente nos slides. O algoritmo funciona de maneira que um processo que quer entrar na seção crítica, por meio de abstração, envia uma mensagem para todos os outros processos pedindo permissão. À princípio, se os outros processos não estiverem em contato com a seção crítica, eles iram enviar uma mensagem de permissão para que o processo com intenção de interagir com a seção crítica possa agir de forma segura.

Dessa forma, o DIMEX orquestra os processos para que, ao serem identificadas várias necessidades de entrada na seção crítica, possa existir prioridade entre os processos para decidir quem entrará primeiro. Por ser um sistema distribuído, não há a garantia de que os relógios de cada processo estejam sincronizados. Sendo assim, a definição de prioridade acontece pelo relógio lógico, funcionando como uma fila de senhas que são atualizadas a cada requisição para entrada na seção crítica.

Propriedades do DIMEX:

DMX1: (não-postergação e não bloqueio) se um processo solicita Entry, decorrido um tempo ele entregará resp:

De acordo com essa propriedade do DIMEX, se um processo solicita a entrada na seção crítica, decorrido um tempo, o DIMEX entregará uma resp. Isso é comprovado em nosso código por alguns passos:

1. Quando o processo p1 solicita entry, o DIMEX manda uma mensagem para todos os outros processos, pedindo permissão para entrar.
    a. No melhor caso, todos os processos estarão no estado de não querer entrar em contato com a seção crítica. Assim, vão enviar a resposta Ok e, assim, o algoritmo DIMEX vai entregar a resp.

```
func (module *DIMEX_Module) handleUponDeliverReqEntry(msgOutro PP2PLink.PP2PLink_Ind_Message) {
	// outro processo quer entrar na SC
	//quando pl entregar msg [ pl, Deliver | q, [ reqEntry, rid, rts ]

	reqTs := module.reqTs
	id := module.id
	rts, rid := stringToReqTsAndId(msgOutro.Message)
	module.outDbg("reqTs: " + strconv.Itoa(reqTs) + " id: " + strconv.Itoa(id) + " rts: " + strconv.Itoa(rts) + " rid: " + strconv.Itoa(rid)
	if (module.st == noMX) || (module.st == wantMX && after(id, reqTs, rid, rts)) {
		//	then  gera evento [ pl, Send | q , [ respOk ]  ]
		rsAddress := module.processes[rid]
		module.outDbg("		<<<---- responde OK! " + msgOutro.Message)
		module.sendToLink(rsAddress, "respOK", strconv.Itoa(module.id))
	} else {
		module.waiting[rid] = true
		// if   (st == inMX) OR  (st == wantMX AND [rts,rid]> [reqTs,id])
		// then   waiting := waiting + [ q ]     else  // empty
	}
	module.lcl = max(module.lcl, rts)

}
```

    b. No pior caso, todos os processos estarão no estado de também querer entrar na seção crítica. Assim o processo p1 vai entrar em waiting em todos os outros processos, neste exemplo em que o processo p1 ficou com o úitimo relógio lógico.
    Por mais que entre em waiting, quando todos os outros processos enviarem Exit, irão mandar o OK para o p1, assim ele poderá enviar a resp quando tiver o OK de todos os outros processos

```
func (module *DIMEX_Module) handleUponReqExit() {

	// quando aplicação avisa [ dmx, Exit   ]  faça
	for qIndex := range module.waiting {
		if module.waiting[qIndex] {
			q := module.processes[qIndex]
			// manda msg[ pl, Send | q , [ respOk ]  ]
			module.sendToLink(q, "respOK", strconv.Itoa(module.id))
			module.waiting[qIndex] = false
		}
	}

	module.st = noMX

}
```

Nos dois casos extremos, implementamos nosso algoritmo de forma que entregue o ok, seja um outro processo que não quer entrar na seção crítica e enviou o ok ou que já enviou o exit da seção crítica e enviou o ok.

DMX2: (mutex) Se um processo p entregou resp, nenhum outro processo entregará resp antes que p sinalize solicite Exit:

Nosso código garante a propriedade de exclusão mútua (mutex) porque, para um processo entrar na seção crítica, ele precisa receber o *respOK* de todos os outros processos. Um outro processo só envia o *respOK* se não quiser entrar na seção crítica ou se quiser entrar, mas o processo que enviou a solicitação tiver prioridade sobre o processo que recebeu a mensagem (seja pelo relógio lógico ou pelo ID). Dessa forma, se nenhum outro processo quiser entrar, ou se o processo em análise tiver prioridade, não haverá interferência, permitindo que ele entre sozinho na seção crítica.

Por outro lado, se mais de um processo quiser entrar na seção crítica, os critérios de desempate do método que é executado ao receber uma mensagem de solicitação garantem que não haverá dois processos na seção crítica ao mesmo tempo. Mesmo que dois processos enviem mensagens indicando que querem entrar na seção crítica, a prioridade é dada ao processo que solicitou primeiro, de acordo com o relógio lógico. E se o relógio lógico for igual? O desempate é feito pelo ID dos processos.

Além disso, o *respOK* só será enviado quando o DIMEX enviar *Exit*, ou seja, quando o processo já tiver saído da seção crítica.

Esses são os únicos momentos em que o *respOK* pode ser enviado de volta ao processo que quer entrar na seção crítica. Assim, o processo P1 que deseja entrar na seção crítica só receberá o *respOK* se o outro processo P2 já tiver terminado sua execução na seção crítica, ou se o processo P1 tiver prioridade (seja pelo relógio lógico ou pelo ID) sobre P2.