Algoritmo Mutex usando Multicast e Relógios Lógicos

Exclusão mútua (mutex) baseado em multicast

Idéia básica:

Os processos que solicitam a entrada em uma seção crítica fazem multicast da mensagem de pedido.

A entrada é aceita quando todos processos responderem a mensagem.

Um processo só ganha o RC quando todos os outros processos aceitarem seu pedido

Definições

Exclusão mútua (mutex): técnica usada em programação concorrente para **evitar que dois processos** ou threads tenham **acesso simultaneamente** a um **recurso compartilhado** (seção crítica).

Multicast: transmissão de informação para múltiplos destinatários simultaneamente.

Definições

Exclusão mútua (mutex): técnica usada em programação concorrente para **evitar que dois processos** ou threads tenham **acesso simultaneamente** a um **recurso compartilhado** (seção crítica).

Multicast: transmissão de informação para múltiplos destinatários simultaneamente.

Na inicialização

state := RELEASED;

Para entrar na seção

state := WANTED;
Envia o *pedido* por *multicast* para todos os processos;
T := indicação de tempo do pedido;
Espera até (número de respostas recebidas = (*N* – 1));
state := HELD;

} *Processamento do pedido referido aqui*

No recebimento de um pedido <Ti, pi> em p_j ($i \neq j$)

if (*state* = HELD *or* (*state* = WANTED *and* (*T, pj*) < (*Ti, pi*)))

then

enfileira *pedido* de p_i sem responder;

else

responde imediatamente para p_i;

end if

Para sair da seção crítica

state := RELEASED;

responde a todos os pedidos enfileirados;

Problemas da apresentação 01

- Estava tentando fazer cada processo ter sua própria Região Crítica.

- Solução: Diminuir o problema para termos apenas uma região crítica para todos concorrerem.

Novas estrutra de dados

Comecei o problema novamente com uma nova estrutura de dados a fim de simplificar tanto a estrutura quanto a legibilidade

Processo

```
public class MyProcess implements ProcessActions{

    Integer pid;
    Integer lamportClock;
    ArrayList<Message> msgList;
    ArrayList<Boolean> mutexList;
    ProcessState state;
    TextView textView;
    RelativeLayout relativeLayout;
    Activity activity;
    int color;
```

Funções executadas pelo Processo

```
public interface ProcessActions {

    public void inicializa(/*ArrayList<MyProcess> processList*/);
    public void solicitaRegiaoCritica();
    public void recebeSolicitacaoRegiaoCritica(Message msg);
    public void acessaRegiaoCritica(Activity activity,  RelativeLayout relativeLayoutPar);
    public void liberaRegiaoCritica(Activity activity);
    public void enviaMensagem(Message msg);
    public void enfileira(Message msg);
    public void desenfileira();
}
```

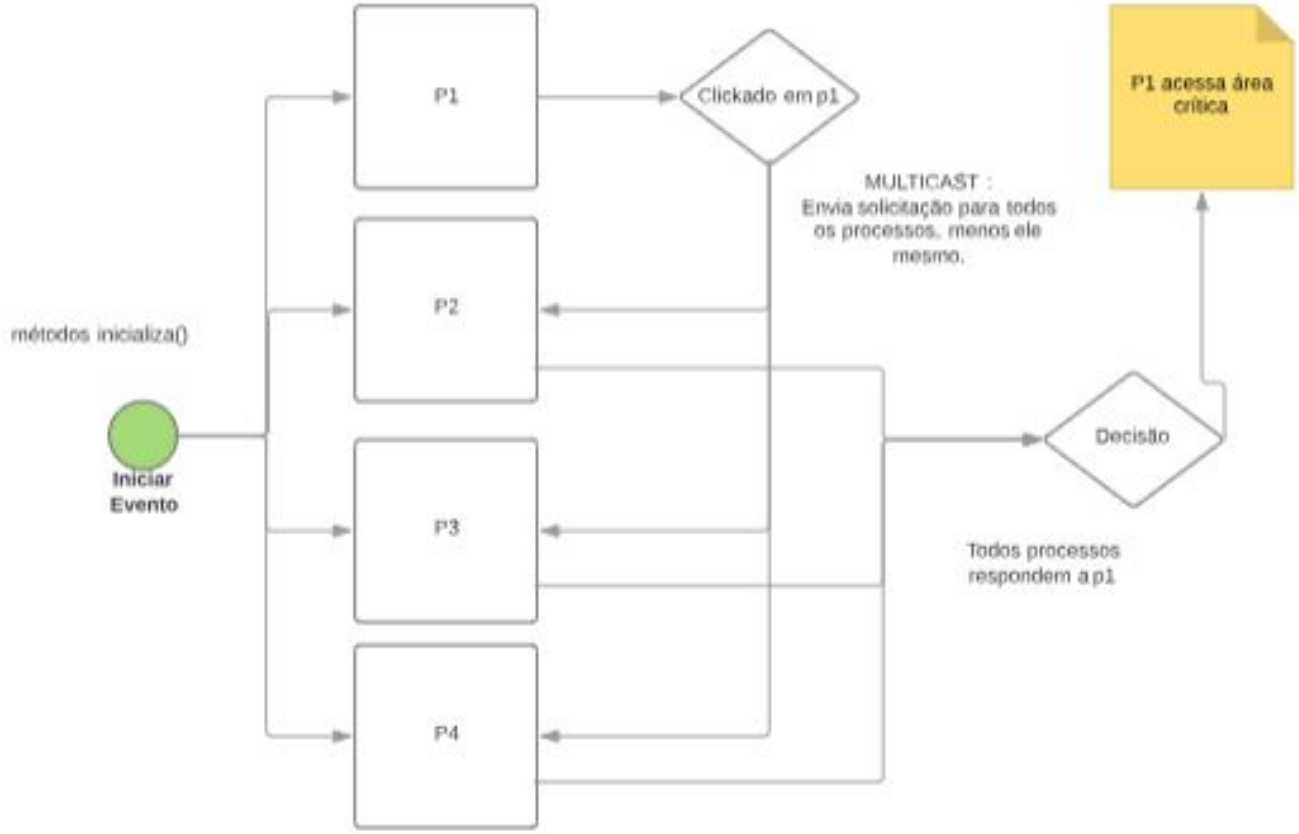
Mensagem

```
public class Message {

    MyProcess senderProcess;
    MyProcess receiverProcess;
    int senderLamportClock;

    public Message(MyProcess senderProcess, MyProcess receiverProcess, int senderLamportClock) {
        this.senderProcess = senderProcess;
        this.receiverProcess = receiverProcess;
        this.senderLamportClock = senderLamportClock;
    }
```

Fluxograma UML

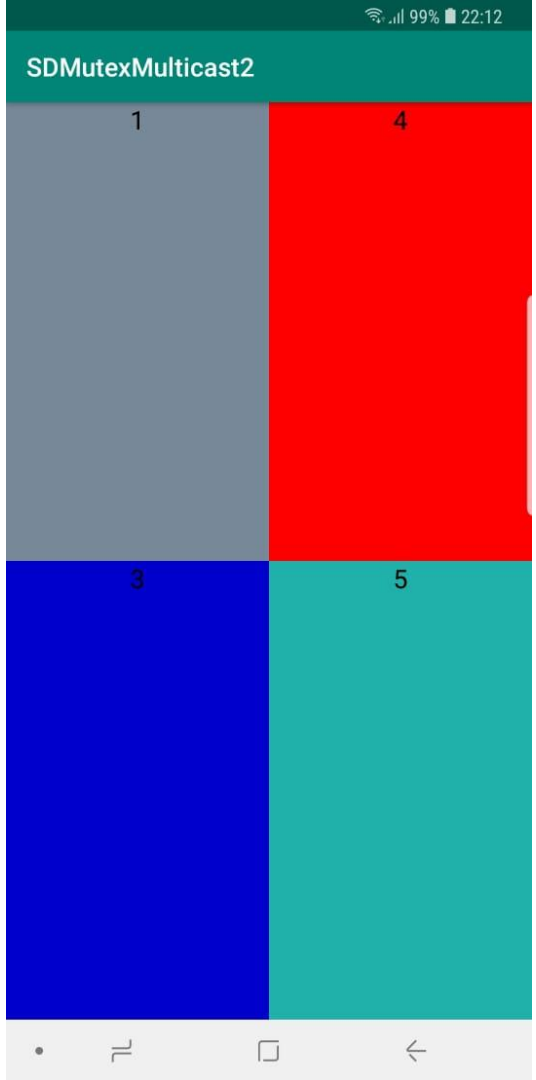


Como funciona o aplicativo

A cada click no layout, é solicitado o acesso à Seção Crítica.

A Seção Crítica é o recurso do SO Android de Câmera (flash), pois esse recurso só pode ser acessado unicamente por um processo (seja ele do sistema, qualquer aplicação ou thread de aplicação).

O relógio de cada processo é mostrado em tela o valor atualizado em tempo real



P0

P1

P2

P3

Problemas que tive:

A solução que eu queria entregar era distribuir cada processo em um dispositivo.

Tive problema em enviar os dados entre dispositivos.

Foi feito implementação via bluetooth, mantendo os dispositivos pareados sendo cliente e servidor, porém a complexidade dessa implementação estava ficando maior que o próprio problema.

Foi feita implementação via firebase, cuja a solução era a comunicação via push notification entre dispositivos. A falha dessa solução foi eu não conseguir implementar o envio de N dispositivos para N-1 pelo fato de não conseguir implementar um servidor local para executar isso o tratamento.

Aplicação finalizada

Foram criados na main 4 processos isolados.

Cada processo comunica com a main (fazendo o papel de distribuidora do serviço) a fim de aplicar o multicast.

As solução da aplicação segue o algoritmo com a descrição do problema/solução.

A requisição da SC é só pode ser tomada exclusivamente por um processo (feature do SO), ou retornaria erro.

Em vídeo existem 2 testes de demonstração, porém, caso queira executar o aplicativo (.apk) se encontra no git.