Automação em Tempo Real
Documentação Etapa I

Gustavo da Silva Gomes
Fernando Ferreira Santos

2022

UFMG, Belo Horizonte

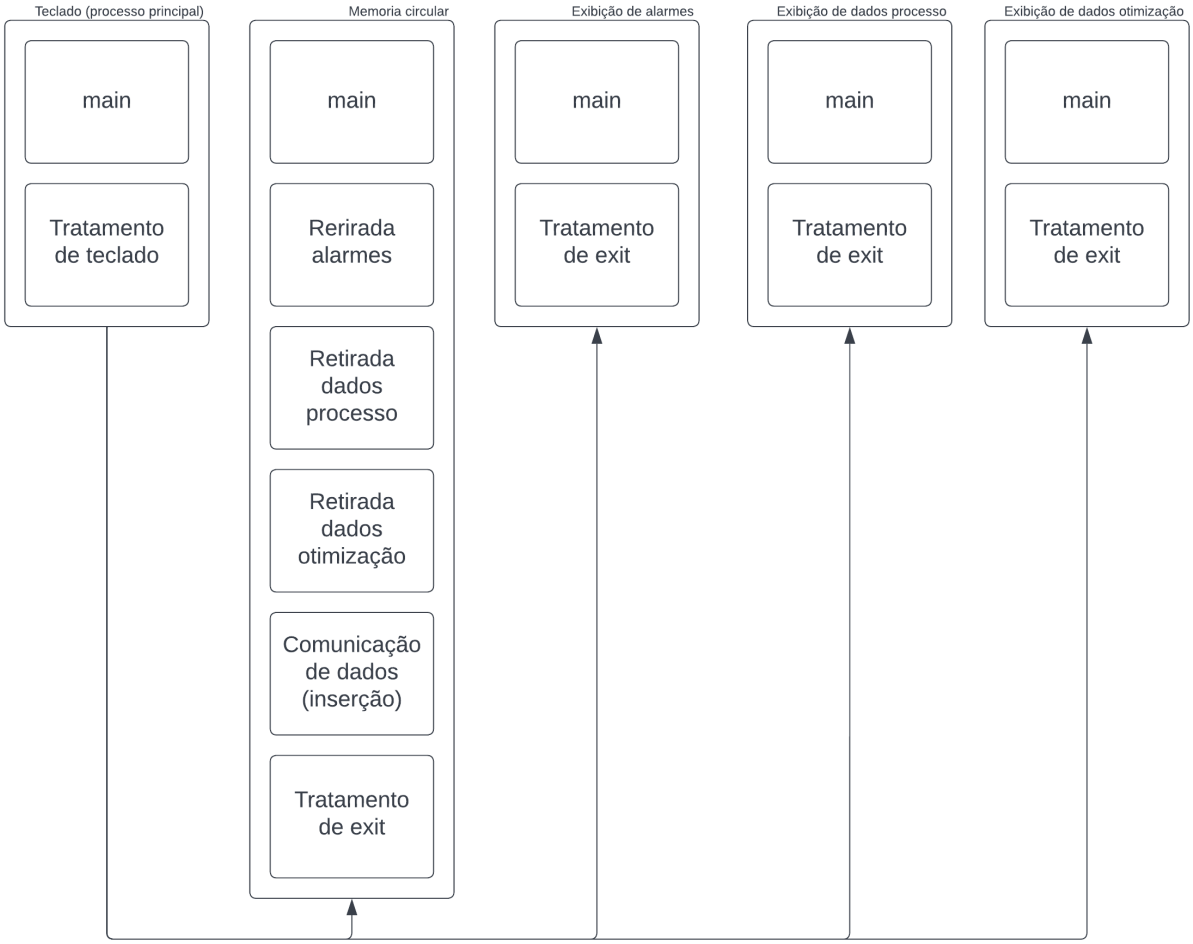
Sumário

1 Introdução

O objetivo deste trabalho é construir uma aplicação de tempo real, que possui geração, remoção e exibição de mensagens dos tipos dados de processo, dados de otimização e alarmes em tarefas e consoles independentes. Essas mensagens devem ser depositadas em uma lista circular em memória e, na etapa II, quando retiradas deverão ser enviadas para as tarefas de exibição por meio de técnicas de comunicação de processos, como pipes e mail-slots. Ao fim do projeto espera-se que os alunos tenham alcançado um maior nível de experiência e entendimento de projetos de automação em tempo real.

2 Estrutura de processos e threads

Organizamos o nosso projeto de acordo com o esquema abaixo:



1. **Processo de teclado**:
   O processo de teclado é o processo em que se inicia a aplicação, sendo composto por duas threads:

   (a) **Thread main**:
   É a thread que inicia todo o projeto. Ela tem a responsabilidade de criar todos os processos filhos e a thread de tratamento de teclado.

   (b) **Thread Tratamento de teclado**:
   Responsável por identificar entradas do teclado e setar, resetar ou pulsar um evento conforme o estado atual do sistema.

2. **Processo de memória circular**:
   O processo de memória circular é o processo responsável por todas as ações relacionadas à

memória circular. Apesar de compartilhar o console da thread de tratamento de teclado, preferimos criar um processo ao invés de threads por motivos semânticos. É composto pelas seguintes threads:

(a) **Thread main**:
É a thread que inicia o processo. Ela tem a responsabilidade de criar todos os eventos de sincronização locais e as threads secundárias.

(b) **Thread Retirada de alarmes**:
Responsável por identificar mensagens do tipo alarme e retirá-las da memória circular.

(c) **Thread Retirada de dados de processo**:
Responsável por identificar mensagens do tipo dado de processo e retirá-las da memória circular.

(d) **Thread Retirada de dados de otimização**:
Responsável por identificar mensagens do tipo dado de otimização e retirá-las da memória circular.

(e) **Thread Comunicação de dados**:
Responsável por criar mensagens dos três tipos com uma periodicidade de 1 segundo.

(f) **Thread Tratamento de exit**:
Responsável por identificar a ativação do evento de exit (que é ativado quando é pressionado ESC ou uma tecla não conhecida) e forçar a finalização do processo.

3. **Processo de exibição de alarmes**:
O processo de exibição de alarmes responsável por exibir as mensagens retiradas pela tarefa de remoção de alarmes. Como seu funcionamento ainda não está implementado (será feito na etapa II), enquanto ele estiver ativo imprimirá no console uma mensagem que indique thread desbloqueada com um intervalo de 2s entre cada mensagem. Se o processo estiver inativo, nada será impresso no console. É composto pelas threads:

   (a) **Thread main**:
   É a thread que inicia o processo. Ela tem a responsabilidade de criar todos os eventos de sincronização locais e as thread secundária.

   (b) **Thread Tratamento de exit**:
   Responsável por identificar a ativação do evento de exit (que é ativado quando é pressionado ESC ou uma tecla não conhecida) e forçar a finalização do processo.

4. **Processo de exibição de dados de otimização**:
O processo de exibição de dados de otimização responsável por exibir as mensagens retiradas pela tarefa de remoção de dados de otimização. Como seu funcionamento ainda não está implementado (será feito na etapa II), enquanto ele estiver ativo imprimirá no console uma mensagem que indique thread desbloqueada com um intervalo de 2s entre cada mensagem. Se o processo estiver inativo, nada será impresso no console. É composto pelas threads:

   (a) **Thread main**:
   É a thread que inicia o processo. Ela tem a responsabilidade de criar todos os eventos de sincronização locais e as thread secundária.

   (b) **Thread Tratamento de exit**:
   Responsável por identificar a ativação do evento de exit (que é ativado quando é pressionado ESC ou uma tecla não conhecida) e forçar a finalização do processo.

5. **Processo de exibição de dados de processo**:
   O processo de exibição de dados de processo responsável por exibir as mensagens retiradas pela tarefa de remoção de dados de processo. Como seu funcionamento ainda não está implementado (será feito na etapa II), enquanto ele estiver ativo imprimirá no console uma mensagem que indique thread desbloqueada com um intervalo de 2s entre cada mensagem. Se o processo estiver inativo, nada será impresso no console. É composto pelas threads:

   (a) **Thread main**:
   É a thread que inicia o processo. Ela tem a responsabilidade de criar todos os eventos de sincronização locais e as thread secundária.

   (b) **Thread Tratamento de exit**:
   Responsável por identificar a ativação do evento de exit (que é ativado quando é pressionado ESC ou uma tecla não conhecida) e forçar a finalização do processo.

3 Classes auxiliares

Para isolar os detalhes de implementação da geração de mensagens, criamos a classe MessageGenerator, que tem o papel de criar mensagens dos três tipos gerados pelo sistema.