

DETECÇÃO DE ERROS EM SISTEMA OPERACIONAL DE TEMPO REAL

Universidade do Vale do Itajaí (UNIVALI)

Escola Politécnica - Ciência da computação

Aluno: Marcos Augusto Fehlauer Pereira

Orientador: Felipe Viel

Introdução

- Sistemas embarcados estão presentes diversas áreas, tipicamente utilizando de um sistema operacional de tempo real
- É provável que a adoção destes sistemas, particularmente sistemas COTS continue a crescer

Problematização

- Existem diversas técnicas para tornar um sistema tolerante à falhas, mas seus tradeoffs nem sempre são claros.
- Pode ser vantajoso de um ponto de vista competitivo e social, que estes sistemas apresentem melhor dependabilidade.
- Portanto, é necessário conhecer os tradeoffs de performance em relação ao seu ganho de tolerância.

Solução Proposta

- Implementar técnicas de tolerância à falhas próximas do escalonador do sistema operacional, analisar o impacto de performance causado e criar uma interface para o uso das técnicas.

Objetivos

Geral

Explorar o uso de técnicas de escalonamento de tempo real com detecção de erros.

Específicos

- Selecionar técnicas de detecção de falhas em nível de software
- Aplicar como prova de conceito em um RTOS as técnicas selecionadas
- Avaliar por meio de métricas a técnica durante a execução em um RTOS
- Avaliar por meio de métricas a técnica uso de memória em um RTOS

Definições Principais

Dependabilidade: Propriedade do sistema que pode ser sumarizada pelos critérios RAMS

- **Reliability** (Confiabilidade): Probabilidade de um sistema executar corretamente em um período
- **Availability** (Disponibilidade): Razão entre o tempo em que o sistema não consegue prover seu serviço (downtime) e o seu tempo total de operação
- **Maintainability** (Capacidade de Manutenção): Probabilidade de que um sistema em um estado quebrado consiga ser reparado com sucesso, antes de um tempo t .
- **Safety** (Segurança): Probabilidade do sistema funcione ou não sem causar danos à integridade humana ou à outros patrimônios

*Tolerar falhas influencia positivamente nos critérios **R** e **A**, melhorando a dependabilidade.*

Definições Principais

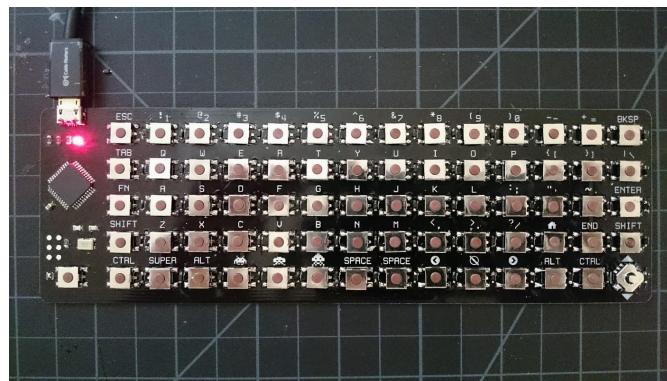
Definições em português segundo a IEEE:

- **Erro (Error)**: A diferença entre um valor esperado e um valor obtido.
- **Defeito (Fault)**: Estado irregular do sistema, que pode provocar (ou não) erros que levam à falhas
- **Falha (Failure)**: Incapacidade observável do sistema de cumprir sua função designada, constituindo uma degradação total ou parcial de sua qualidade de serviço.

Para os propósitos deste trabalho, o termo “Falha” será utilizado como um termo mais abrangente, representando um estado ou evento no sistema que causa uma degradação da qualidade de serviço.

Sistemas Embarcados

- Família vasta de sistemas computacionais que capacitam um dispositivo maior.
- Características comuns: Especificidade, Limitação de Recursos, Critério temporal (Soft ou Hard real time)

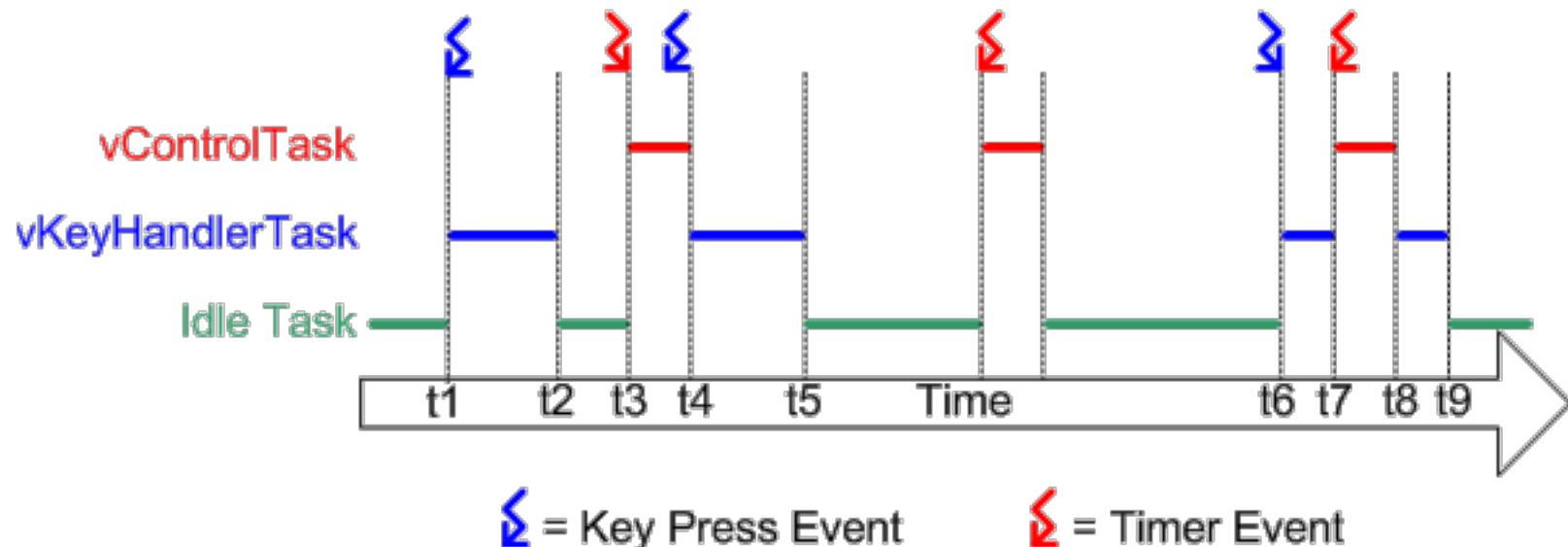


Sistemas Operacionais de Tempo Real

Sistemas comumente usados para diversos tipos de sistemas embarcados, possuem escalonadores totalmente preemptivos. Tipicamente possuem poucas features, dependendo apenas de uma HAL (*Hardware Abstraction Layer*)

Escalonador

Componente do Sistema Operacional responsável por gerenciar o tempo da CPU entre das tarefas.



Falhas

Falhas podem ser classificadas em 3 grupos de acordo com seu padrão de ocorrência:

- Falhas **Transientes**: Ocorrem aleatoriamente e possuem um impacto temporário.
- Falhas **Intermitentes**: Assim como as transientes possuem impacto temporário, porém re-ocorrem periodicamente.
- Falhas **Permanentes**: Causam uma degradação permanente no sistema da qual não pode ser recuperada, potencialmente necessitando de intervenção externa.

Este trabalho focará na tolerância à falhas transientes.

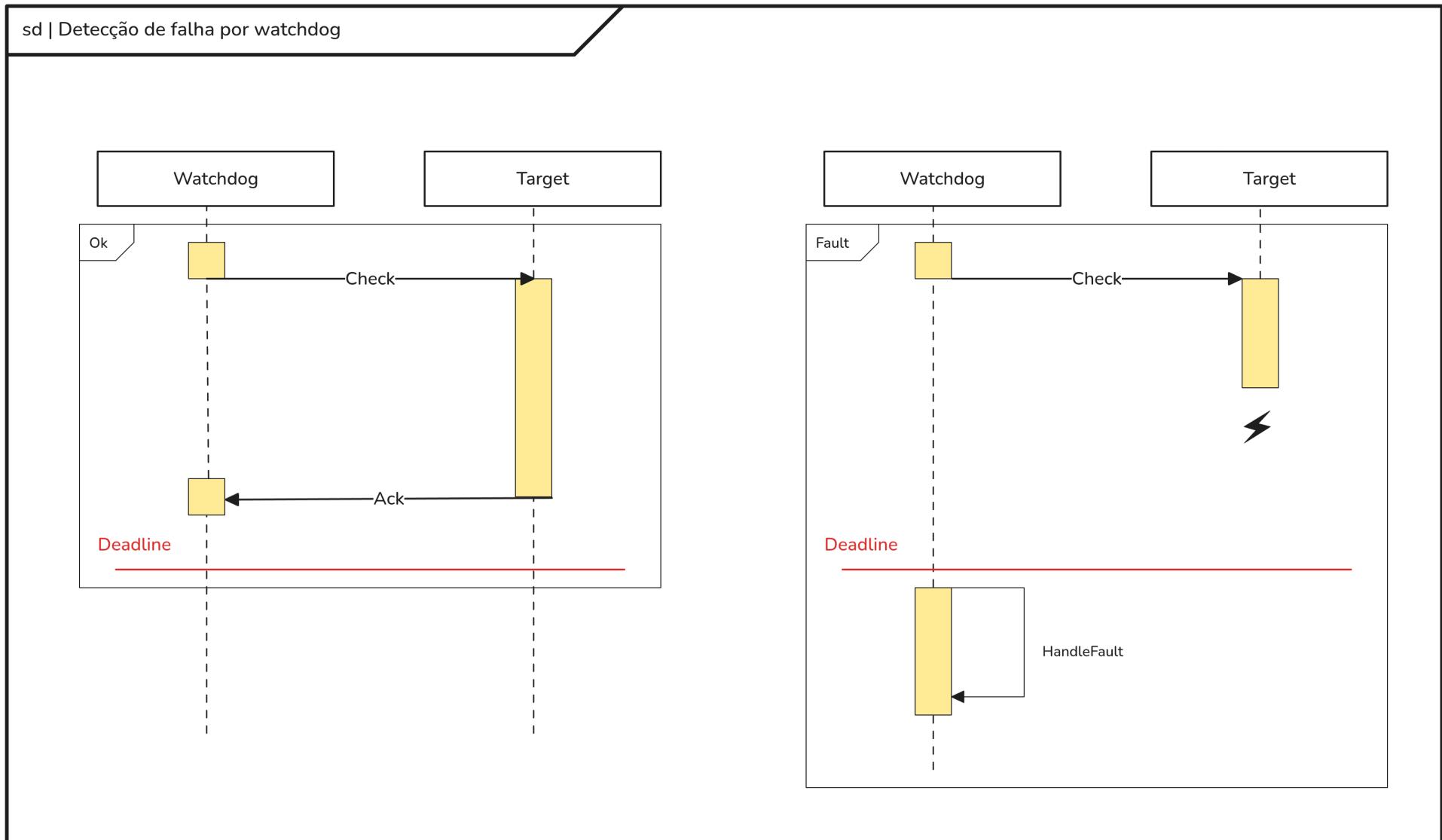
Mecanismos de Detecção

- CRC (Cyclic Redundancy Check): Um valor de checagem é criado com base em um polinômio gerador e verificado, utilizado primariamente para verificar integridade de pacotes ou mensagens.
- Asserts: Checagem de uma condição invariante que dispara uma falha, simples e muito flexível, pode ser automaticamente inserido como pós e pré condição na chamada de funções

```
void assert(bool predicate, string message){  
    [[unlikely]]  
    if(!predicate){  
        log_error(message); // Opcional: imprimir uma mensagem de erro  
        trap(); // Emitir exceção  
    }  
}
```

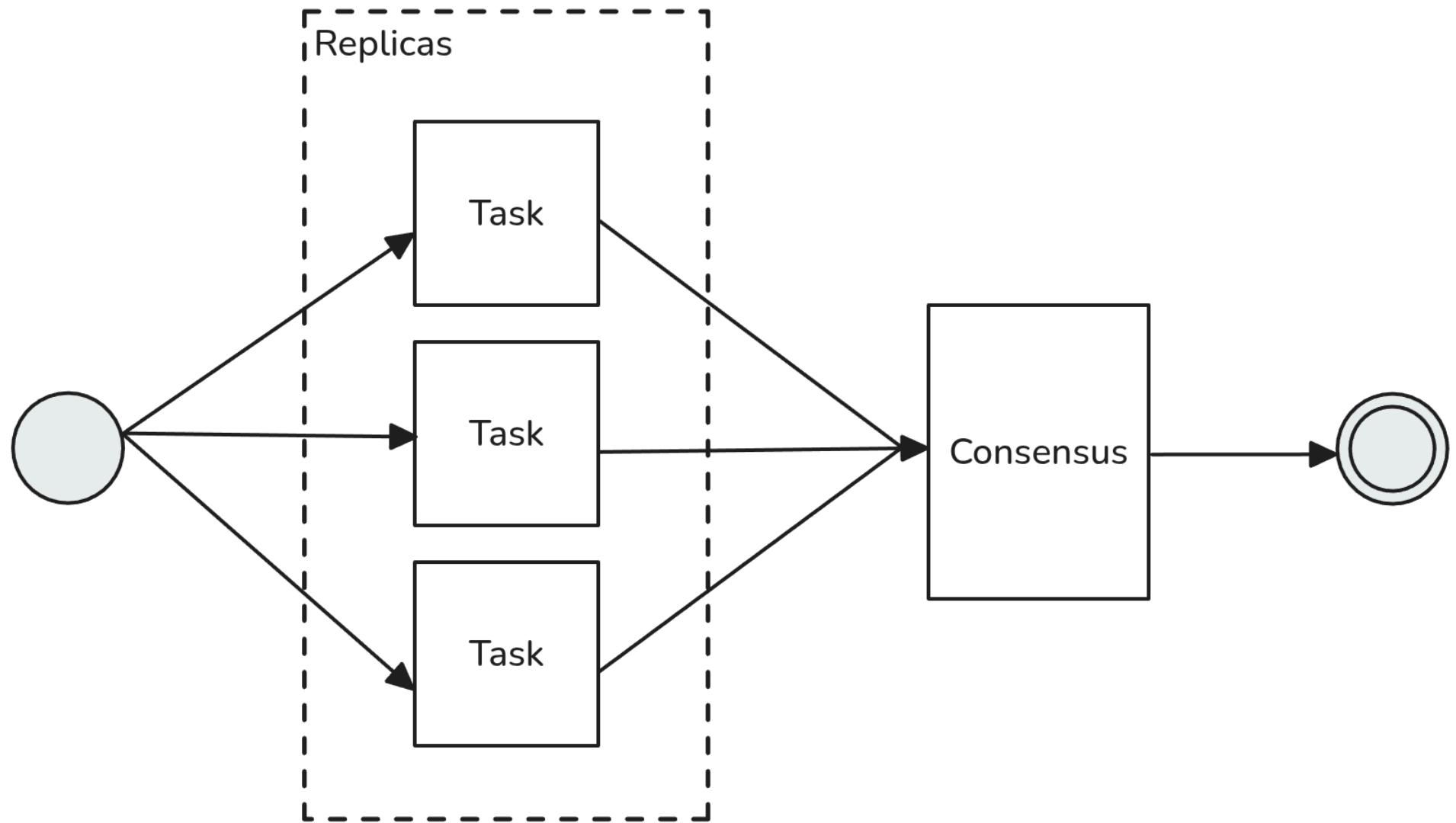
Mecanismos de Detecção

- Heartbeat signal: Sinal de Checagem, tipicamente baseado em uma deadline



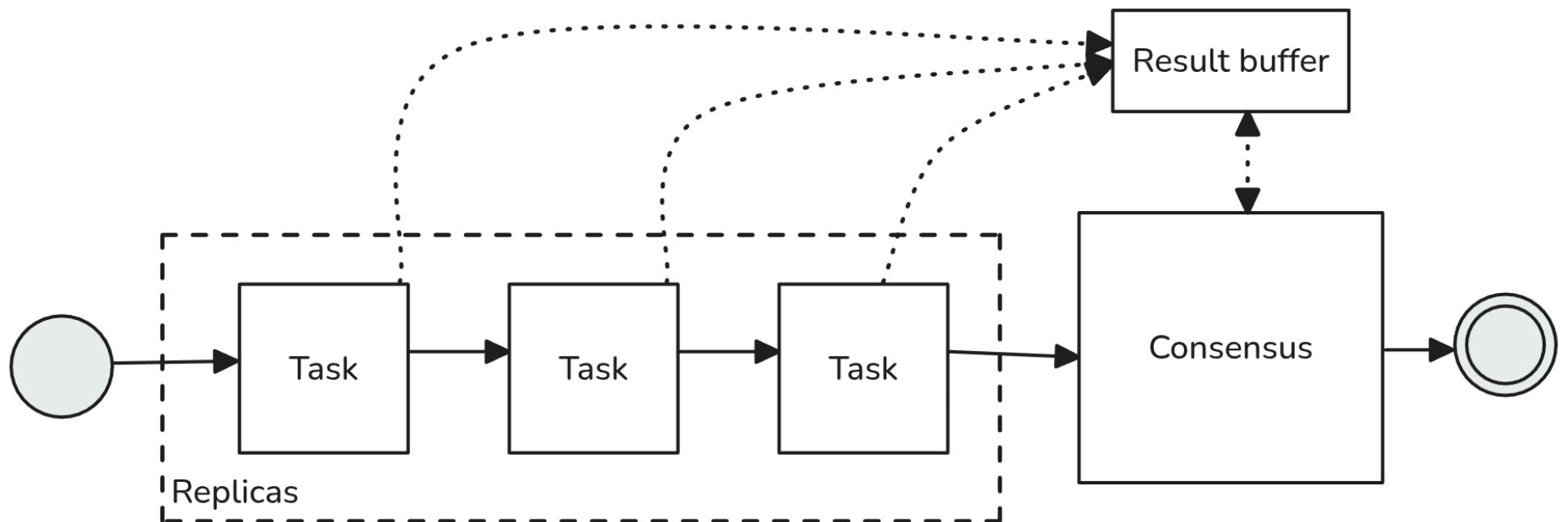
Mecanismos de Tratamento

Redundância: Pode ser inserida em diversos estágios, depende do fato que é menos provável que uma falha ocorra em N lugares dentro de um período t .



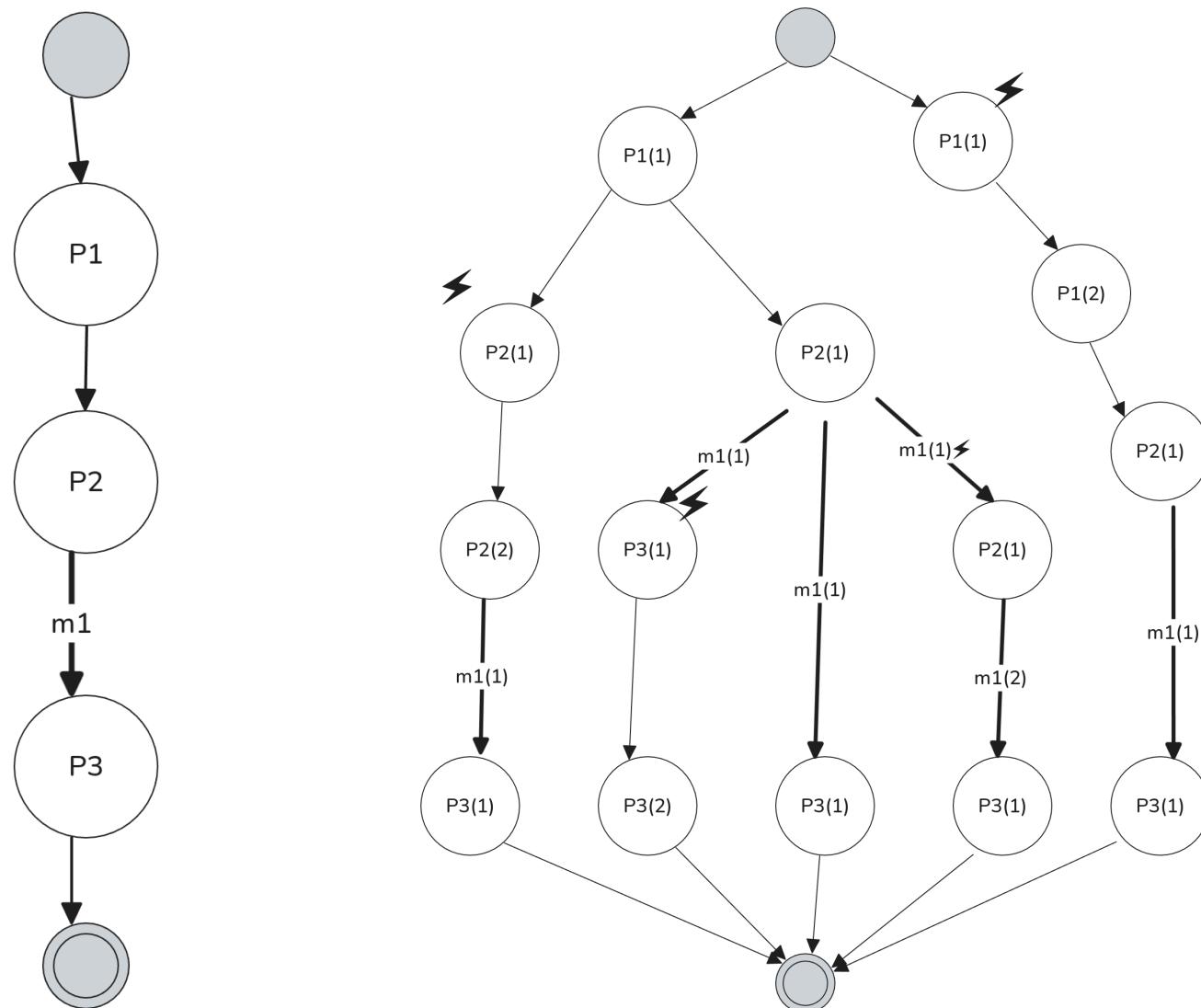
Mecanismos de Tratamento

Reexecução: Um tipo de redundância temporal, pode ser utilizado para condições de transparência, depende do fato que é improvável que uma falha transiente ocorra N vezes em sucessão.



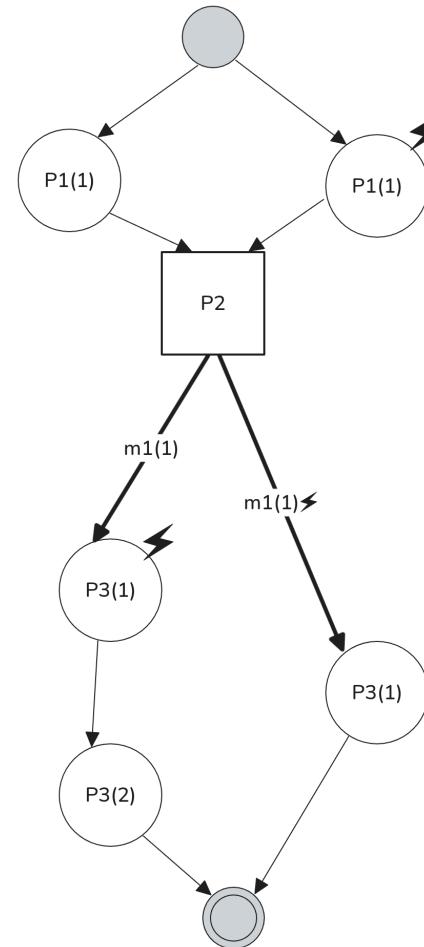
Escalonamento Tolerante à Falhas

Grafo tolerante à falhas de um programa simples (3 processos, 1 mensagem) e sua versão que tolera até uma falha transiente.



Escalonamento Tolerante à Falhas

Com condição de transparência (através de reexecução) inserida.



Inserir condições de transparência pode drasticamente reduzir a complexidade do grafo de execução.

Injeção de Falhas

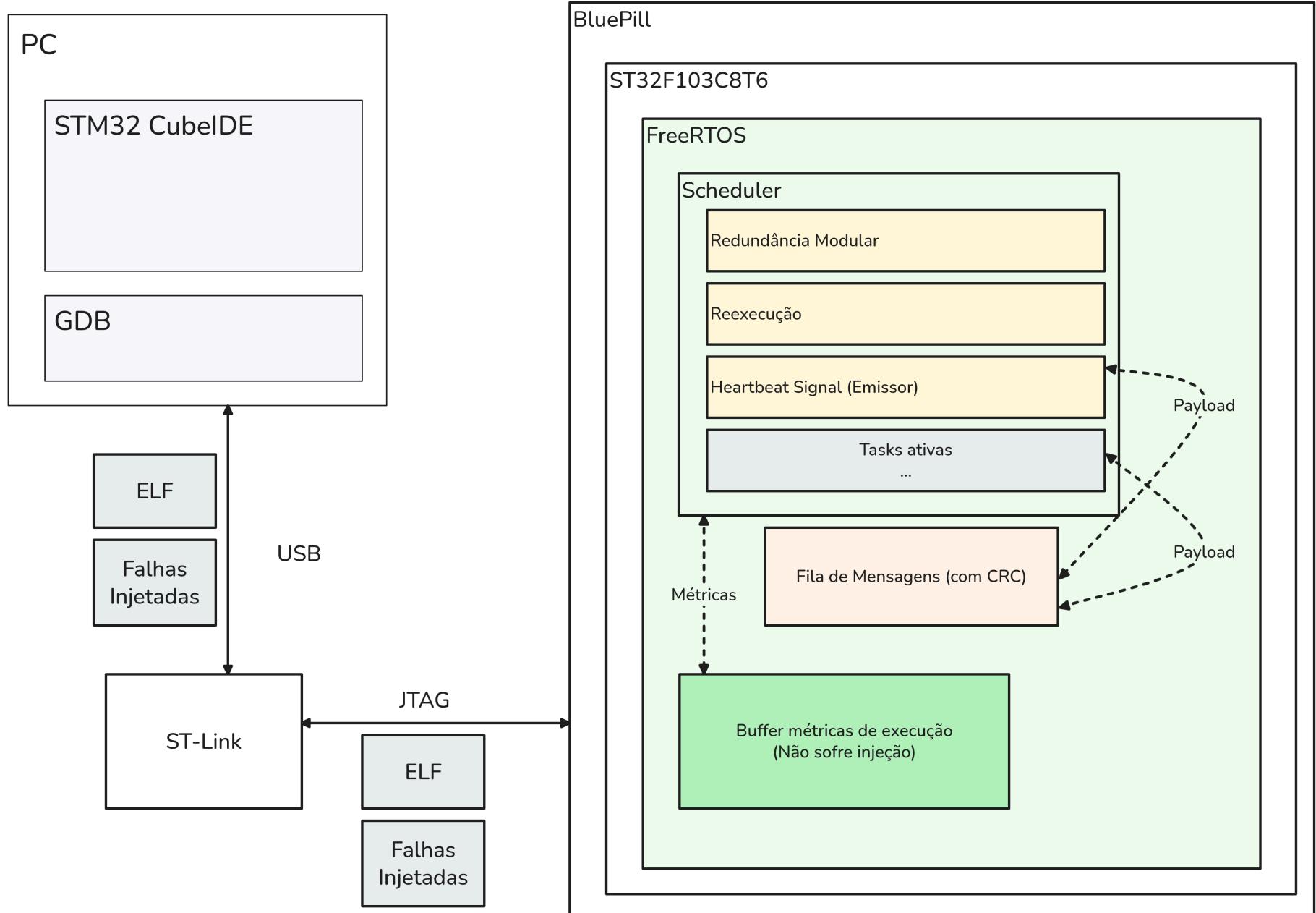
Tipos de injeção e suas desvantagens (Mamone, 2018)

Técnica	Vantagens	Desvantagens
Física	<ul style="list-style-type: none">• Alta fidelidade à falhas reais• Possível injetar em partes específicas do chip	<ul style="list-style-type: none">• Alto custo• Menos controle sobre o tipo particular de falha• Especialistas para lidar com equipamentos
Lógica em Hardware	<ul style="list-style-type: none">• Boa aproximação de falhas reais• Altamente precisa e oferece controle sobre dados injetados• Overhead pequeno no tempo de execução	<ul style="list-style-type: none">• Necessita de uma unidade extra(depurador/injetor)• Necessita de um sistema de comunicação• Pode necessitar de pinos ou modificações adicionais no sistema alvo
Lógica em Software	<ul style="list-style-type: none">• Baixo custo• Flexível e precisa• Altamente portável	<ul style="list-style-type: none">• Overhead no tamanho do código e no tempo de execução
Simulada	<ul style="list-style-type: none">• Zero intrusividade• Hardware alvo não necessário• Flexível e precisa	<ul style="list-style-type: none">• Custo de ferramenta de simulação pode ser alto• Nem sempre uma descrição HDL do sistema está acessível

Comparação dos Trabalhos Relacionados

N	Trabalho	Sistema	Hardware	Injeção	Técnicas
1	Reliability Assessment of Arm Cortex-M Processors under Heavy Ions and Emulated Fault Injection	Bare Metal, FreeRTOS	CY8CKIT-059	Física & Lógica em Software	Redundância de Registradores, Deadlines, Redução de Registradores, Asserts
2	Application-Level Fault Tolerance in Real-Time Embedded System	BOSS	Máquinas PowerPC 823 e um PC x86_643 não especificado	Simulada em Software	Redundância Modular, Deadlines, Rollback/Retry
3	A Software Implemented Comprehensive Soft Error Detection Method for Embedded Systems	MicroC/OS-ii	MPC555 Evaluation Board	Lógica em Hardware	Análise de fluxo de controle e de dados com sensibilidade à deadlines
-	Este Trabalho	FreeRTOS	STM32 Bluepill	Lógica em Software e Hardware	Deadlines, Heartbeat, Asserts, Reexecução e Redundância de Tarefas

Visão Geral



Premissas

- Registradores de controle (Stack Pointer, Return Address, Program Counter, Thread Pointer) serão isentos de falhas diretas.
- Será assumido que testes sintéticos possam ao menos aproximar a medição de um cenário com falhas físicas.
- Não será utilizado RTTI ou exceções baseadas em stack unwinding.

Métodos

- Técnicas de detecção e tolerância baseadas em software.
- Injeção lógica em software durante desenvolvimento.
- Injeção lógica em hardware para teste final com depurador de hardware.
- Métricas coletadas com o profiler do FreeRTOS e mecanismos de código (contadores)
- Interface de tarefa com fortificação em sua V-Table
- Arquitetura orientada à passagem de mensagens
- Serão desenvolvidos dois programas de teste simples, para servir de exemplo para a coleta das métricas.

Materiais

- STLink: Depurador de hardware
- STM32F103C8T6 “Bluepill”: Microcontrolador para a execução do código
- GCC: Compilador para a linguagem C++ (Versão 14 ou acima)
- STCubeIDE, QEMU: IDE e ferramenta de virtualização para auxílio



Requisitos

- Implementar os Algoritmos e Técnicas propostos
- Implementar os programas de teste
- Implementar interface de tarefa com TF
- Executar e coletar métricas com injeção lógica em software
- Executar e coletar métricas com injeção lógica em hardware

Algoritmos e Técnicas

- CRC
- Heartbeat Signal
- Redundância Modular
- Replicação Temporal
- Asserts

Interface

Exemplo ilustrativo de uma mensagem

```
// Deve ser o suficiente para conter o valor de um timer monotônico
using Time_Point = size_t;

using Task_Id = unsigned int;

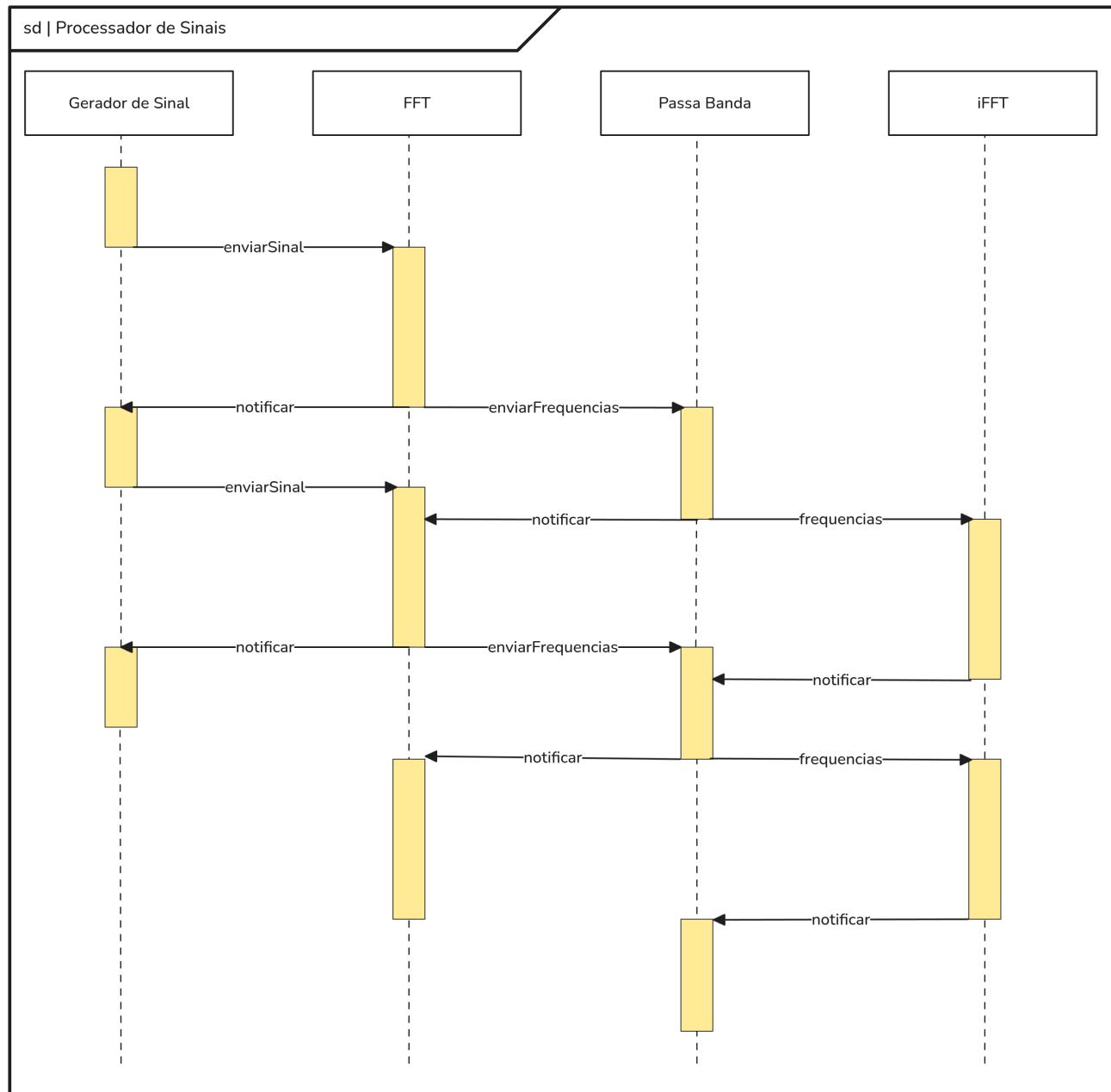
template<typename Payload>
struct FT_Message {
    uint32_t    check_value;
    Task_Id     sender;
    Task_Id     receiver;
    Time_Point  sent_at;
    Time_Point  deadline; // 0 - Sem deadline de entrega
    Payload     payload;
};
```

Interface

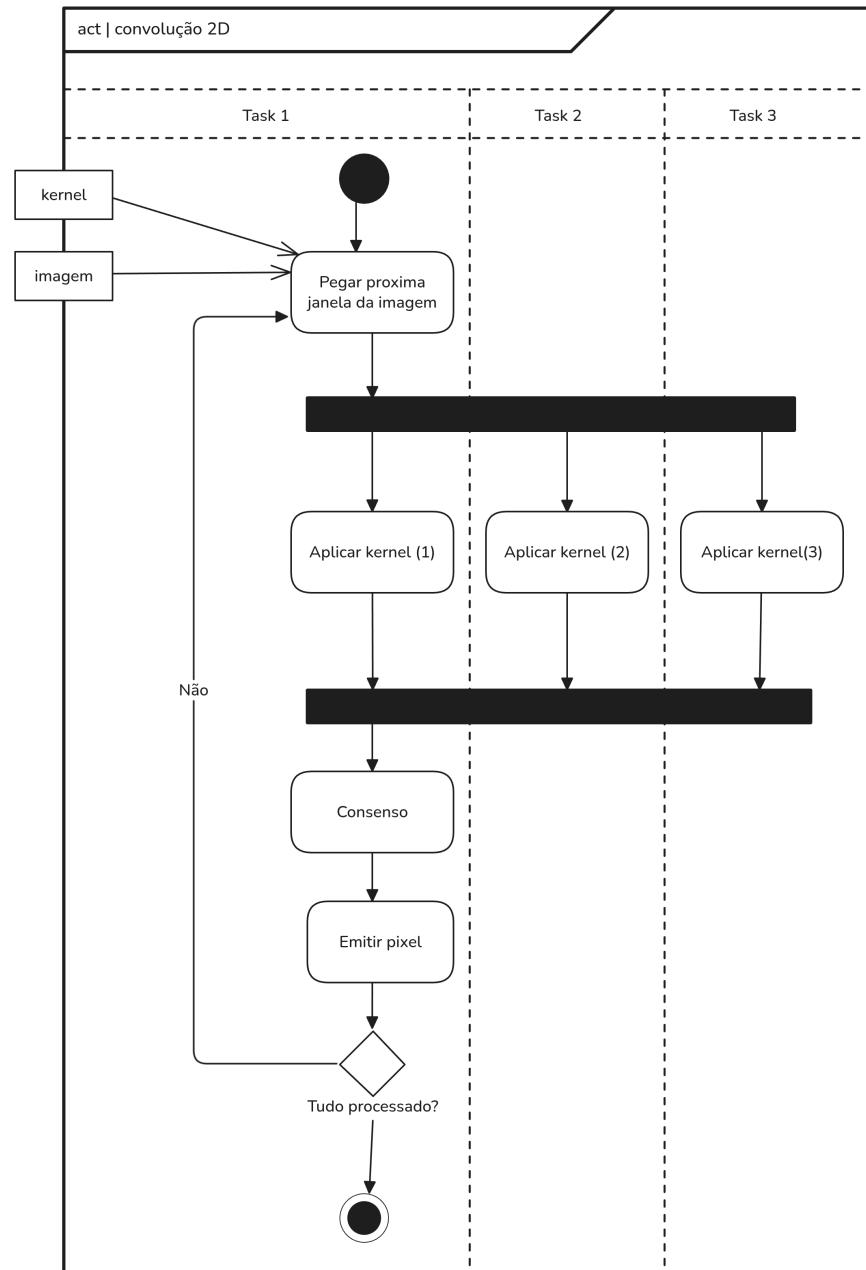
Interface básica de uma tarefa

```
using FT_Handler = bool (*)(FT_Task*);  
  
struct FT_Task_Info {  
    Task_Id id;  
    FT_Handler handler;  
    Time_Point started_at;  
    Time_Point deadline; // 0 - Sem deadline  
};  
  
struct FT_Task {  
    virtual Task_Id execute(void* param) = 0;  
    virtual void attach_heartbeat_watchdog(uint32_t* sequence_addr,  
Time_Point interval) = 0;  
    virtual FT_Task_Info info() = 0;  
};
```

Programa Teste 1: Processador de Sinal digital



Programa Teste 2: Convolução Bidimensional



Plano de Verificação

- Testes unitários para:
 - ▶ CRC
 - ▶ Heartbeat Signal
 - ▶ Redundância Modular
 - ▶ Replicação temporal
 - ▶ Fila de Mensagens (MPMC bounded queue)
 - ▶ FFT, iFFT, Passa-Banda (Programa exemplo 1)
 - ▶ Convolução (Programa exemplo 2)
- Testes de integração e ponta-a-ponta para os programas de exemplo
- Teste com injeção baseada em software para garantir funcionamento preliminar
- Teste e análise final com injeção lógica em hardware
- É possível reutilizar uma boa parte da lógica de emissão de falhas em software para o depurador de hardware

Campanha de Injeção de Falhas

- Sujeitos à falhas: Maioria da memória, Registradores de propósito geral
- Falhas injetadas: Bit flips com XOR, números aleatórios. Simulando corrupções de memória
- Método de injeção: Task auxiliar durante testes e desenvolvimento, Comandos via sessão GDB para alterar valores no controlador via STLink

Combinações de técnicas a serem usadas:

Comb.	Reexecução	Redundância modular	Heartbeat Signal	CRC	Asserts
1	-	-	-	-	-
2	-	-	-	✓	✓
3	✓	-	-	✓	✓
4	✓	-	✓	✓	✓
5	-	✓	-	✓	✓
6	-	✓	✓	✓	✓

Análise de Riscos

Risco	Probabilidade	Impacto	Gatilho	Contingência
Funcionalidades e API do RTOS é incompatível com a interface proposta pelo trabalho.	Baixo	Alto	Implementar interface no RTOS	Utilizar outro RTOS, modificar o FreeRTOS, adaptar a interface
Problemas para injetar falhas com depurador em hardware	Baixa	Alto	Realizar injeção no microcontrolador	Utilizar de outro depurador, depender de falhas lógicas em software como última alternativa
Não conseguir coletar métricas de performance com profiler do FreeRTOS	Baixa	Médio	Teste em microcontrolador ou ambiente virtualizado	Inserir pontos de medição manualmente

Cronograma do TCC3

Atividade	07/2025	08/2025	09/2025	10/2025	11/2025	12/2025
Escrita da Monografia	XXXX	XXXX	XXXX	XXXX	XXXX	X__
Implementação dos Algoritmos	XXXX	XX__	___	___	___	___
Testes dos Algoritmos	XXXX	XX__	___	___	___	___
Implementação dos Programas Exemplo	___	_XX	XXX_	___	___	___
Teste dos Programas Exemplo	___	_XX	XXX_	___	___	___
Implementação da Injeção com Software	___	___	XXXX	___	___	___
Implementação da Injeção com Hardware	___	___	_XX	XXX_	___	___
Execução microcontrolador e coleta das métricas	___	___	___	_XX	XXXX	___
Revisão Textual	___	___	___	___	_XX	XXXX

Considerações Finais

- RTOSes são de grande importância e aumentar sua dependabilidade pode ser benéfico
- Integrar a detecção e tolerância à falhas com o processo de escalonamento através de uma interface permite criar abstrações para melhorias incrementais
- O escopo da campanha de falhas do trabalho serão os erros de memória, causados por evento externo ou erro de design
- Dentre as principais limitações do trabalho:
 - Não será realizado teste físico
 - Não será realizado análise de fluxo para pegar corrupções mais sofisticadas de fluxo.