



RELATÓRIO TÉCNICO — Implementação Concorrente do Jogo

Keep Solving and Nobody Explodes

Autor: Rafael Severo e Pedro Fontosa

Instituição: Instituto Brasileiro de Ensino, Desenvolvimento e Pesquisa (IDP)

Disciplina: Sistemas Operacionais (Ubuntu 24.04 – pthreads / ncurses)

Ano: 2025

Resumo

Este relatório descreve a implementação concorrente e o funcionamento completo do jogo *Keep Solving and Nobody Explodes*, desenvolvido em linguagem C com uso extensivo de *POSIX Threads*, mutexes, variáveis de condição, semáforos e interface ncurses.

O sistema simula múltiplos módulos explosivos que surgem ao longo do tempo, devendo ser gerenciados por técnicos (Tedax) e processados em bancadas limitadas. O documento apresenta:

1. **A arquitetura concorrente (implementação técnica)**
2. **As regras do jogo, objetivos e forma de jogar (manual do usuário)**
3. **Um guia detalhado de instalação e execução**

O objetivo é demonstrar não apenas a robustez da implementação, mas também o funcionamento lógico e lúdico do simulador, garantindo entendimento completo pelo avaliador.

1. Introdução

O projeto tem como finalidade integrar conceitos de programação concorrente estudados na disciplina a um sistema interativo em tempo real.

O jogo simula uma central de desarmamento onde o jogador atua como coordenador: não desarma bombas diretamente, mas **gerencia recursos, prioriza módulos e opera a fila de trabalho**, sob pressão de tempo e recursos escassos.

A ideia central é construir um ambiente multithread estável, livre de condições de corrida, capaz de manter interface e lógica funcionando simultaneamente, mesmo durante picos de carga (modo Insano).

2. Arquitetura do Sistema e Organização das Threads

A aplicação foi projetada em modelo **Produtor-Consumidor** com múltiplos agentes paralelos.

As threads são:

2.1 Thread Principal (main)

Inicializa estruturas, aloca o pool de Tedax, carrega parâmetros de dificuldade e inicia as demais threads.

Exibe o menu (Figura 1), permitindo ao jogador definir:

- Velocidade de geração de módulos
- Número de Tedax
- Número de Bancadas
- Tempo limite por módulo
- Duração total do jogo

```
=====
Keep Solving and Nobody Explodes
=====

Escolha a dificuldade (tecle o numero e Enter):

[1] FACIL    (1 Tedax, 1 bancada, spawn lento)
[2] MEDIO    (2 Tedax, 2 bancadas) (padrao)
[3] DIFICIL  (3 Tedax, 3 bancadas, spawn rapido)
[4] INSANO   (3 Tedax, 2 bancadas, spawn muito rapido)

Escolha: 4
Dificuldade escolhida: 4 -> TEDAX=3 BANCADAS=2 GEN_MS=720 TIMEOUT=18s
Pressione ENTER para iniciar o jogo...
```

2.2 Gerador (generator_fn)

Cria novos módulos periodicamente.

Cada módulo contém:

- ID
- tipo (Botão, Fios ou Senhas)
- tempo limite
- solução interna
- timestamp de criação

2.3 Watcher (watcher_fn)

Decrementa o tempo restante dos módulos a cada ciclo.

Quando o tempo chega a zero:

- o módulo **explode** (timeout)
- é devolvido ao Mural com novo tempo e penalidade
- o jogador perde tempo/controla

2.4 Interface (ui_thread_fn)

Responsável por entrada e saída.

Utiliza ncurses e apresenta a tela (Figura 2).

Por segurança, nunca acessa o Mural sem:

- `mural_lock_access()`
- `mural_unlock_access()`

A UI funciona apenas como **produtora de comandos**, não executa lógica.

2.5 Coordenador (coordinator_fn)

Consome comandos da fila.

Executa:

- Auto-Assign
- Designação manual
- Validação de sintaxe
- Seleção de Tedax livre
- Entrega do módulo ao trabalhador correspondente

2.6 Tedax (tedax_thread_fn)

Cada Tedax:

1. aguarda tarefas via variável de condição
 2. disputa uma bancada livre (semáforo)
 3. processa o módulo (tempo variável)
 4. libera bancada
 5. retorna ao estado “Disponível”
-

3. Sincronização e Seções Críticas

A arquitetura utiliza mecanismos POSIX para garantir estabilidade:

3.1 Proteção do Mural

Um mutex (`mural_lock`) protege contra acessos simultâneos.

UI, Gerador, Watcher e Coordenador disputam acesso.

Implementação de **leitura segura** na UI evita *segfault* durante renderização.

3.2 Fila de Comandos (produtor–consumidor)

A UI produz comandos
O Coordenador consome

Proteção via:

`Q_mut`

`Q_cond`

`pthread_cond_wait()` (evita polling e desperdício de CPU)

3.3 Bancadas (semáforos)

`benches_sem` limita acesso simultâneo.

`bench_mutex` garante integridade na tabela de ocupação.

3.4 Tedax Locks

Cada técnico tem mutex próprio e variável de condição.

Corrigido o problema de *double-check locking* presente no código original.

4. Funcionalidades, Interface e Fluxo do Jogo (Regras)

Esta seção explica **como jogar, regras, objetivos, condições de vitória/derrota, interpretar a UI, e como funcionam os comandos.**

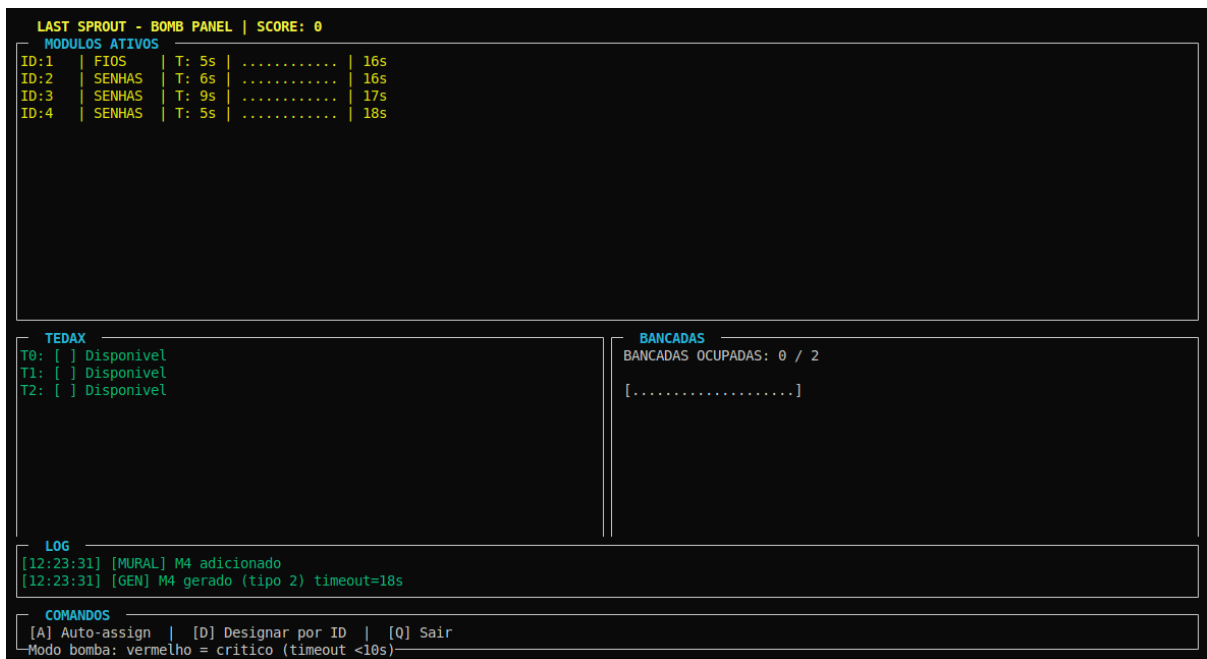


Figura 2 — Interface principal do jogo em execução

4.1 Objetivo do Jogo

Você é o **Coordenador**.

Seu objetivo é **manter o sistema funcionando até o cronômetro principal zerar**.

- Não existe explosão única que encerra a partida.
- Um módulo estoura → volta para o Mural com penalidade.
- Se muitos módulos se acumulam, você perde o controle e não consegue mais estabilizar.

O desafio é **gerenciar a fila** com eficiência.

4.2 A Interface (como interpretar)

1. Mural (Esquerda)

Lista todos os módulos ativos:

- **ID**
- **Tipo** (BOTÃO, FIOS, SENHAS)
- **T:** tempo restante

- **Tempo total decorrido**

Cores indicam risco:

- **Verde:** seguro
- **Amarelo:** moderado
- **Vermelho:** crítico (<10s)

2. Painel de Tedax (Centro)

Mostra cada trabalhador:

- [] Disponível
- [0] Ocupado (com barra de progresso)

Cada Tedax é uma thread real.

3. Bancadas (Direita)

Representam recursos físicos.

Mesmo com Tedax livres, **não é possível processar módulos sem bancadas livres.**

- . = bancada livre
- # = bancada ocupada

4. Log (Inferior)

Todas as ações do sistema e do jogador são registradas:

- módulo gerado
- atribuição
- falhas
- sucessos
- desbloqueios
- tempo restante

4.3 Como Jogar — Comandos

A — Auto-Assign (opção rápida)

Pega o módulo mais antigo e envia ao primeiro Tedax livre.

Vantagem: rápido

Desvantagem: não prioriza módulos críticos

D — Designar Manualmente (modo cirúrgico)

Permite:

1. escolher um módulo crítico
2. fornecer a instrução de desarmamento

Fluxo:

1. Pressione **D**
2. Digite o ID
3. Escolha:
 - **A** — Assign
 - **S** — Solve
4. Digite a instrução correta

4.4 Sintaxe dos Comandos de Desarmamento

Tipo de Módulo	Formato do Comando	Exemplos (O que digitar)
✂ FIOS	CUT <número>	CUT 1 CUT 2

		CUT 3
 BOTAO	<COR> <AÇÃO>	RED PRESS BLUE HOLD GREEN DOUBLE YELLOW PRESS
 SENHAS	WORD <PALAVRA>	WORD FIRE WORD WATER WORD EARTH WORD WIND

		WORD VOID
--	--	-----------

4.5 Condições de Vitória e Derrota

Vitória:

O cronômetro superior chega a **zero** sem travamentos.

Derrota (implícita):

O sistema fica sobrecarregado por:

- módulos demais
- Tedax saturados
- bancadas insuficientes
- tempo perdido
- reaparecimento constante de módulos estourados

O jogo **não exibe GAME OVER**, mas a interface ficará saturada e você não conseguirá recuperar o controle.

4.6 Estratégia Recomendada

1. Priorize módulos vermelhos.
2. Use **A** quando a fila está tranquila.
3. Use **D** para resgatar módulos críticos.
4. Evite deixar Tedax inativos.

5. Mantenha sempre alguma bancada livre para emergências.
-

5. Guia de Instalação e Execução

5.1 Dependências

No Ubuntu 24.04:

```
sudo apt update  
sudo apt install build-essential libncurses5-dev
```

5.2 Compilação

```
make clean  
make
```

Gera o binário **ksne**.

5.3 Execução

```
./ksne
```

6. Manual de Utilização

O jogador controla o Coordenador. A interface apresenta:

- **Mural:** módulos ativos, tipo e tempo restante
- **Tedax:** estado das threads
- **Bancadas:** controle de recursos
- **Log:** histórico de eventos
- **Comandos:**

- **A** — Auto-Assign
- **D** — Designação manual
 - CUT n
 - <COR> <AÇÃO>
 - WORD <palavra>

Dificuldades alteram:

- taxa de geração
- tempo limite dos módulos
- tempo total do jogo

7. Conclusão

A implementação final integra conceitos de programação concorrente com uma mecânica de jogo funcional. Todas as threads operam de maneira segura e sincronizada, eliminando condições de corrida e travamentos do código original.

O jogo demonstra na prática:

- coordenação entre múltiplas threads
 - uso adequado de semáforos e mutexes
 - comunicação através de filas sincronizadas
 - interface ncurses estável
 - gerenciamento real de recursos limitados
-