**Nome dos integrantes:**

Lucas Tourinho - 2221133

Tiago de Jesus Chehab – 2211194

# Relatório Técnico - Trabalho Programação Concorrente

## Introdução

Para o trabalho final da disciplina, foi requerido a criação de um sistema que simule o jogo “Keep Solving and Nobody Explodes”, nesta simulação é necessário apenas um jogador que precisará desarmar uma bomba. O documento detalha a implementação da estrutura do jogo usando a concorrência e foi feita na linguagem de programação C. Na execução utiliza-se de threads, seções críticas e sincronização para garantir o funcionamento correto do jogo.

## Explicação e mecânica do jogo

O jogo simula uma situação de emergência onde se deve desarmar uma bomba antes do término do tempo, ela contém diferentes tipos de módulos explosivos e um número limitado de bancadas. É preciso um grande nível de coordenação para que o jogador consiga cumprir com o objetivo do jogo, desarmar o explosivo.

O número de tedax e módulos é fixo, sendo definido como 3 tedax e 10 módulos disponíveis. No decorrer do jogo, múltiplos tedax podem aguardar na fila para utilizar as bancadas, sempre respeitando a regra que cada tedax só pode interagir com um módulo por vez. O jogo termina quando todos os 10 módulos são desarmados com sucesso. Caso contrário, ele continuará por 120 segundos (2 minutos) até que o jogo seja finalizado e uma mensagem de derrota irá aparecer.

### Fluxo do jogo

1. **Geração de módulos:**
    a. Os módulos são criados periodicamente pela thread “**module_board_func**”.
    b. Cada módulo tem um identificador único, um tipo (**x**, **c**, ou **t**) e um número de interações necessárias para ser desarmado:
        i. **x (Botões)**: Exige pressionar uma quantidade de botões.
        ii. **c (Sequência)**: Exige inserir uma sequência de caracteres.
        iii. **t (Temporizador)**: Requer aguardar um tempo específico.
2. **Coordenação:**

    a. A thread “**coordinator_func**” permite ao jogador designar manualmente um módulo específico para um tedax, uma bancada específica onde o módulo será desarmado.
3. **Desarmamento de módulos:**
    a. Cada tedax, controlado pela thread “**tedax_func**”, monitora os módulos disponíveis.
    b. Quando um módulo é atribuído:
        i. O tedax ocupa uma bancada (usando um semáforo para sincronização).
        ii. Realiza as interações necessárias para desarmar o módulo.
    c. Após concluir o desarmamento:
        i. O módulo é marcado como desarmado.
        ii. A bancada é liberada para outro tedax.
4. **Interface:**
    a. A thread “**display_func**” atualiza continuamente a interface, exibindo os módulos pendentes, em progresso e desarmados, status do tedax e quando há a conclusão do jogo, exibe uma mensagem de vitória.
5. **Temporizador:**
    a. A “**`timer_func`**” é responsável por diminuir o tempo disponível para desarmar os módulos de forma contínua. A cada segundo, a thread reduz o tempo restante e sinaliza quando o tempo acabou, impactando diretamente a continuidade do jogo. Caso o tempo expire antes de o jogador desarmar todos os módulos, o jogo é encerrado, sinalizando derrota.

## Threads

- **module_board_func**: Gera novos módulos explosivos em intervalos fixos e os adiciona à fila de módulos.
- **display_func**: Atualiza a interface do jogo em tempo real, exibindo informações sobre os módulos, tedax e bancadas.
- **coordinator_func**: Coordena manualmente a designação de módulos aos tedax e bancadas.
- **tedax_func**: Representa cada tedax, que espera por um módulo disponível e interage com ele até desarmá-lo.
- **timer_func**: Responsável por gerenciar o tempo restante para o fim do jogo. Ela pode ser usada para implementar a contagem regressiva do tempo, e quando o tempo acabar, ela sinaliza que o jogo terminou.
- **Principal (main)**: Responsável por iniciar e finalizar as threads e gerenciar a configuração inicial.

## Seções críticas

- **Fila de Módulos**: Protegida por um mutex (“**module_queue_lock**”) para evitar condições de corrida durante a leitura e escrita.
- **Bancadas**: Controladas por um semáforo (“**benches**”), que regula o acesso às bancadas disponíveis e gerencia a fila de tedax esperando para utilizá-las.

## Sincronização

- **Mutexes**: Garantem consistência na fila de módulos e no estado de cada módulo.
- **Semáforos**: Gerenciam o uso das bancadas, permitindo que múltiplos tedax aguardem e acessem bancadas de forma organizada.

## Compilação, execução e requisitos

1. **Comandos disponíveis para compilar o projeto:**
    - bash make | O binário será gerado em bin/game. Executar o Jogo:
    - bash ./bin/game | Limpar Arquivos de Compilação:
    - bash make clean | Remove todos os arquivos objetos (.o) e o binário gerado. Limpeza Completa:
    - bash make distclean | Remove todos os diretórios gerados (obj e bin). Ajuda:
    - bash make help | Exibe os comandos disponíveis no Makefile. 3. Requisitos Compilador GCC: Certifique-se de que o GCC está instalado:
    - bash gcc --version Bibliotecas ncurses: | Instale as dependências necessárias:
    - bash sudo apt update sudo apt install libncurses5-dev libncursesw5-dev 4. | Testando o Projeto Compile:
    - bash make | Execute:
    - bash ./bin/game | Limpe e Compile Novamente:
    - bash make clean make | Remova Todos os Diretórios Gerados:
    - bash make distclean

## Conclusão

O projeto demonstra de forma prática os conceitos que foram aprendidos durante a matéria de programação concorrente no segundo semestre de 2024. A utilização de threads, mutex, semáforos, temporizador, permitindo a coordenação eficiente dos tedax e módulos com os recursos limitados disponíveis. A interface do ncurses foi essencial para a representação visual mais clara do estado do jogo, tornando uma experiência interativa e compreensível.
Tudo que foi desenvolvido pelos alunos pode ser acessado através do repositório feito no GitHub através do link: https://github.com/touro3/Keep-Threading-Nobody-Explodes