Trabalho Prático (T)

Identificação da Disciplina

Código da Disciplina	Nome da Disciplina	Professor	Período
CIC_5MA e EGS_5MA	Programação Concorrente	Jeremias Moreira Gomes	2025/2

1. Objetivo

O objetivo deste trabalho é capacitar o aluno a estudar e compreender conceitos envolvendo programação concorrente. Utilizando o conhecimento adquirido nas aulas, o aluno deverá desenvolver um sistema cujo o gerenciamento e execução de tarefas envolvam a utilização de recursos que são utilizados por múltiplas *threads*.

Ao cumprir os objetivos do trabalho, o aluno terá adquirido: (*i*) uma compreensão prática dos conceitos e características-chave de programação concorrente; (*ii*) maior aprimoramento das suas habilidades de escrita técnica, organização de informações e comunicação eficaz, uma vez que a documentação de implementação exigirá uma apresentação clara e estruturada das principais características do sistema; e (*iii*) essa experiência proporcionará maior confiança e capacidade para explorar esses conceitos na solução de problemas computacionais.

2. Enunciado

A *Intelligent Data Parallelism* (IDP) é uma empresa que tem como objetivo criar e desenvolver softwares recreacionais que estimulam o raciocínio rápido, a cooperação e a comunicação entre pessoas que gostam de jogar jogos cooperativos.

Observando a dinamicidade do mercado de tecnologia, o IDP resolveu investir na criação de diferentes tipos de jogos, ao mesmo tempo em que investe na evolução de times competitivos no cenário de *e-sports*. Para isso, foram criados diversos *squads* responsáveis por avançar em cada uma dessas vertentes, desenvolvendo jogos ou criando simulações de jogos com o objetivo de melhorar o desempenho estratégico dos jogadores.

Um dos fenômenos recentes no mercado de jogos é *Keep Solving and Nobody Explodes*, onde um jogador deve dar instruções para outro jogador, que não pode ver a tela, para desarmar uma bomba. O jogo visa testar a habilidade dos jogadores em se comunicar e colaborar com seus colegas de equipe.

O objetivo do jogo é bem simples, os jogadores recebem instruções específicas para módulos de uma bomba que precisa se desarmada. Cada módulo possui uma característica específica, e o jogador que está desarmado a bomba deve seguir as instruções do jogador que está vendo a tela, para desarmar a bomba. O jogo termina quando a bomba é desarmada ou quando o tempo acaba.

Assim, visando aprimorar a experiência dos jogadores, o seu *squad* foi designado para criar uma simulação do jogo, onde os jogadores poderão treinar individualmente suas habilidades e estratégias, com a possibilidade de ajustar as configurações do jogo para simular diferentes cenários.

3. Keep Solving and Nobody Explodes - Versão de Treino

Na versão de treino/simulação do jogo, há diversas modificações em relação ao jogo original. A primeira delas é que o jogo é jogado por apenas um jogador, que deve desarmar a bomba sozinho, designando módulos para serem desarmados por tedax[1] virtuais.

Ao aparecer um novo módulo na tela, o jogador deve designar esse módulo para um dos tedax disponíveis. Esse tedax, deverá receber, além da designação do módulo, instruções específicas para desarmar o módulo. Ao receber o módulo, o tedax ocupa uma bancada de desativação pelo tempo específico do módulo, e somente após esse tempo, é que o módulo é desarmado, caso as instruções tenham sido corretamente seguidas. Caso contrário, o módulo não é desarmado e retorna para a lista de módulos pendentes. Por último, o número de tedax e bancadas disponíveis é limitado, e o jogador deve gerenciar a designação dos módulos por bancadas e tedax disponíveis.

A figura 1 apresenta um esquema de organização do jogo.

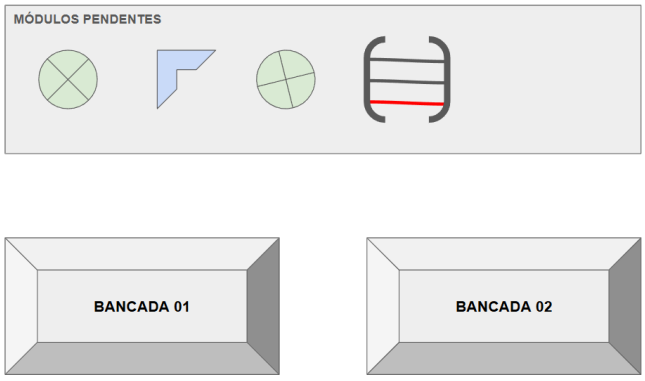

Figura 1. Esquema de Organização do Jogo

É possível perceber que o número de tedax, módulos e bancadas é essencial para a jogabilidade e o treinamento do jogador, e estes são os elementos cruciais, que serão detalhados na parte de implementação.

O jogo termina quando todos os módulos forem desarmados, ou quando o tempo de jogo acabar por acúmulo de módulos não desarmados e a bomba explodir.

[1] Técnico Especialista em Desativação de Artefatos Explosivos

4. Implementação

A implementação do trabalho deve ser feita em C. A organização dos arquivos fica a cargo do aluno/grupo, o qual deverá conter instruções de compilação, bem como um arquivo `Makefile` para facilitar a compilação do código. Além disso, os testes e validação do trabalho serão feitos em um ambiente Linux, similar ao disponibilizado nos laboratórios da disciplina.

4.1. Elementos do Jogo

Como o objetivo do trabalho é gerar uma experiência com programação concorrente, a implementação deverá conter, no mínimo, os seguintes elementos, que funcionarão como *threads*:

- **Mural de Módulos Pendentes**
- **Exibição de Informações**
- **Tedax (es)**
- **Coordenador (jogador)**

4.1.1. Mural de Módulos Pendentes

Essa entidade, que é uma *thread*, é responsável por gerar novos módulos durante a partida. Uma partida contém um tempo limitado de jogo, e durante esse tempo, de maneira regular ou não, o mural irá gerar novos módulos a serem desarmados. Cada módulo possui uma característica específica de solução, e um tempo específico para ser desarmado. Caso o módulo não seja desarmado dentro do tempo limite, ele retorna para o mural de módulos, que fica disponível para o jogador designar para um novo tedax.

4.1.2. Exibição de Informações

Essa entidade é responsável por exibir as informações do jogo para o jogador. Exemplos: tela inicial, lista de módulos acumulados, bancada de desarme livre ou ocupada, lista de tedax disponíveis ou ocupados, tempo restante da partida, etc.

Essa entidade é uma *thread* e o principal cuidado que deve ser tomado é a concordância entre as informações exibidas e o estado atual do jogo, que demanda uma sincronização entre as *threads* (ter acesso exclusivo aos recursos compartilhados, sem que estes sejam alterados durante a atualização da informação).

Para a exibição das informações, é recomendado a utilização da biblioteca `ncurses`, que permite a criação de interfaces de texto em terminais de forma mais amigável e interativa. O apêndice 7 contém um exemplo de código utilizando a biblioteca, bem como um guia rápido de funções que podem ser utilizadas.

4.1.3. Tedax (es)

Um tedax é responsável por desarmar um módulo. Cada tedax está disponível ou ocupado resolvendo um módulo. Quando ocupado, gasta um tempo X para para desarmar um módulo, e após esse tempo, o módulo é desarmado, o qual completa uma tarefa em uma partida. Um tedax só pode desarmar um módulo por vez

e, para isso, ele precisa ocupar uma bancada livre durante esse tempo. Nesse caso, bancadas são recursos compartilhados entre os tedax, e um tedax só pode ocupar uma bancada por vez.

Cada módulo demanda uma ação específica para ser desarmado, além do tempo necessário para desarmá-lo. O tedax deve seguir as instruções do jogador para desarmar o módulo, e caso as instruções sejam corretamente seguidas, o módulo é desarmado, caso contrário, o módulo retorna para o mural de módulos.

Há pelo menos duas formas simples de organizar bancadas durante a implementação: (*i*) cada tedax possui uma bancada própria, a qual esta serve para resolver qualquer tipo de módulo; ou (*ii*) há bancadas específicas para cada tipo de módulo, e o tedax deve ocupar a bancada específica para resolver esse módulo. Além destas, há outras formas de organizar as bancadas, e fica a critério do aluno/grupo escolher e implementar a que achar mais adequada (não esquecendo de documentar).

Dependendo das escolhas do aluno/grupo, o número de tedax disponíveis é limitado, e o jogador deve gerenciar a designação dos módulos por bancadas e tedax disponíveis. Além disso, ao terminar corretamente a desativação de um módulo, o módulo é retirado da lista de módulos pendentes, e o tedax é liberado para desarmar um novo módulo.

Recomenda-se iniciar a implementação com um único tedax, e uma única bancada de desarme, para somente depois estender a implementação para múltiplos tedax e bancadas, após entendimento e validação do funcionamento do jogo.

4.1.4. Coordenador (jogador)

O coordenador é responsável por assinalar os módulos para os tedax. Essa entidade (*thread*) é quem irá coletar os *inputs* do usuário. Com a lista de módulos pendentes diferente de vazia, o coordenador pode assinalar um módulo para um tedax, caso ele esteja disponível e uma bancada de desarme também a esteja.

Para coletar os *inputs* do usuário, caso esteja utilizando a biblioteca `ncurses`, é recomendado a utilização da função `getch()`, que lê um caractere do teclado, sem a necessidade de pressionar a tecla *Enter*. Uma sugestão é manter em *buffer* as últimas teclas pressionadas pelo usuário, e caso estas sejam um comando válido, realizar a ação desejada. Exemplo: se o usuário pressionar as teclas `1`, em seguida `p` e por último `2`, o coordenador está dando a ordem de "O tedax 1 resolver o módulo p, na bancada 2" [2].

4.2. Sumário

Ao final do trabalho, o aluno/grupo deverá entregar a implementação do jogo/simulador.

5. Metodologia

Este trabalho será dividido em duas partes: (*i*) Implementação do Jogo; e (*ii*) Relatório Técnico.

[2]Esta é apenas uma sugestão de implementação. O importante é o usuário entender de maneira amigável como jogar o jogo de maneira simplificada, já que não é necessário fazer movimentações bidimensionais como no jogo original. O tipo de input está diretamente relacionado com a organização das bancadas e tedax.

5.1. Implementação do Jogo (80 pts)

Nesta etapa, os alunos deverão implementar jogo proposto, que precisa utilizar elementos de programação concorrente. A implementação deverá ser feita em C, e deverá conter os elementos do jogo descritos na seção de implementação.

O jogo deverá suportar múltiplos tedax e múltiplas bancadas; Essa quantidade poderá estar atrelada a uma configuração inicial do jogo, que pode ser algo relacionado à sua dificuldade. O mínimo exigido é a possibilidade de ser ter de um a três tedax, com o mesmo número de bancadas.

Os módulos ficam a cargo do aluno decidir o que utilizar e como implementar. Não é necessário ser algo complexo, mas que seja suficiente para que o jogador necessite de um pouco de destreza para resolver. Por exemplo, um módulo que exige ao tedax pressionar um botão X vezes para resolver o módulo, e a sequência de botões para o coordenador seria algo como `3x2pp`, onde o 3 é o número do tedax, `x` é o tipo de módulo, 2 é a bancada disponível, e `pp` é a quantidade de vezes que o tedax deve pressionar um botão para resolver o módulo (lembre-que a complexidade de um módulo, está diretamente relacionada com a complexidade de reconhecer a sequência de instruções correta digitada pelo coordenador).

5.2. Documentação (20 pts)

Nesta etapa, deve-se documentar a implementação do jogo, descrevendo, principalmente, a organização das *threads*, seções críticas e o funcionamento da sincronização entre elas. A documentação deve conter também um guia de utilização do jogo, com as instruções de como instalar e executar o jogo, que será realizado em um ambiente Linux similar ao disponibilizado nos laboratórios da disciplina (Ubuntu 24.04).

A documentação deverá ser concisa e objetiva, não deixando de ser clara, contendo no máximo **três páginas**, seguindo o modelo de artigos da Sociedade Brasileira de Computação (link).

5.3. Assimetria de Tedax e Bancadas (Bônus de 10 pts)

Uma maneira simples de implementar o jogo é assinalar apenas um tedax por bancada, onde não é necessário nem escolher o número da bancada, pois cada tedax tem a sua própria.

Assim, se a quantidade de tedax e bancadas puderem ser diferentes, onde múltiplos tedax podem se enfileirar na espera pelo uso de diferentes bancadas, este receberá uma pontuação adicional de até 10 pontos, pela complexidade das possibilidades durante uma partida.

6. Detalhes Acerca do Trabalho

6.1. Restrições deste Trabalho

Esta atividade poderá ser realizada em até **dois integrantes**, que deverão ser informados ao professor da disciplina até 29/11/2025 (sábado). Caso a dupla não seja informada, será considerado que o aluno está realizando o trabalho de maneira individual.

6.2. Datas Importantes

Esse trabalho poderá ser submetido a partir de 26/11/2025 (quarta-feira) - 12:00h até 12/12/2025 (sexta-feira) - 23:55h.

6.3. Por onde será a entrega o Trabalho?

O trabalho deverá ser entregue via Ambiente Virtual (https://ambientevirtual.idp.edu.br/), na disciplina de Programação Concorrente. Na disciplina haverá uma atividade chamada "Trabalho Prático (T)", para submissão do trabalho.

6.4. O que deverá ser entregue, referente ao trabalho?

Deverá ser entregue a documentação e os códigos produzidos pelos alunos, relacionado a resolução do trabalho.

6.5. Como entregar o trabalho?

A submissão das atividades deverá ser feita em um arquivo único comprimido do tipo zip (`Zip archive data, at least v1.0 to extract`), contendo documentação e um diretório com os códigos elaborados[3]. Além disso, para garantir a integridade do conteúdo entregue, o nome do arquivo comprimido deverá possuir duas informações (além da extensão `.zip`):

- As matrículas dos alunos, separada por hífen.
- O *hash* md5 do arquivo e `.zip`.

Exemplo: `20001101-20010203-3b1b448e9a49cd6a91a1ad044e67fff2.zip`

Para gerar o md5 do arquivo comprimido, utilize o comando `md5sum` do Linux e em seguida faça o renomeamento utilizando o *hash* coletado.

Segue um exemplo de geração do arquivo final a ser submetido no moodle, para um aluno:

```
[13930] j3r3mias@tardis:trabalho-01 > ls
trabalho-01-apc-jeremias.c
[13931] j3r3mias@tardis:trabalho-01 > zip -r aaa.zip trabalho-01-apc-jeremias.c
  adding: trabalho-01-apc-jeremias.c (deflated 99%)
[13932] j3r3mias@tardis:trabalho-01 > ls
aaa.zip  trabalho-01-apc-jeremias.c
[13933] j3r3mias@tardis:trabalho-01 > ls -lha aaa.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 335 out 15 16:54 aaa.zip
[13934] j3r3mias@tardis:trabalho-01 > md5sum aaa.zip
44cf46f9237cb506ebde3e9e1cd044f9  aaa.zip
[13935] j3r3mias@tardis:trabalho-01 > mv aaa.zip 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
[13936] j3r3mias@tardis:trabalho-01 > ls -lha 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 335 out 15 16:54 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
[13937] j3r3mias@tardis:trabalho-01 > md5sum 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
44cf46f9237cb506ebde3e9e1cd044f9  160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
[13938] j3r3mias@tardis:trabalho-01 >
```

Figura 2. Exemplo de geração de arquivo para submissão

6.6. Quem deverá entregar o trabalho?

O Ambiente Virtual disponibiliza uma opção de trabalhos em grupo, onde um membro do grupo realiza a submissão e o arquivo é disponibilizado para ambos. Caso esta opção não esteja disponível, todos os alunos deverão submeter o trabalho no Ambiente Virtual onde, **somente um aluno deverá gerar a versão final do trabalho (arquivo zip)**, para que o *hash* do arquivo não corra o risco de ficar diferente prejudicando a avaliação da entrega.

[3] Evite enviar junto, arquivos desncessários como diretórios `.git/`, por exemplo

6.7. Observações importantes

- **Não deixe para fazer o trabalho de última hora**.
- Não serão aceitos trabalhos com atraso.
- **Não deixe para fazer o trabalho de última hora**.
- Cópias de trabalho receberão zero (além disso, plágio é crime).
- **Não deixe para fazer o trabalho de última hora**.

6.8. Dicas de Implementação

Para facilitar a interação entre usuário e software, é recomendado a utilização da biblioteca `ncurses` para a implementação do jogo. A biblioteca `ncurses` é uma biblioteca de programação que permite a criação de interfaces de texto em terminais de forma mais amigável e interativa. A biblioteca é amplamente utilizada em sistemas Unix e Linux, e é uma excelente ferramenta para a criação de jogos de texto. O apêndice contém um exemplo de uso da biblioteca, bem como um guia rápido de utilização.

Para a implementação do jogo, é recomendado que se planeje bem a estrutura de implementação, antes de começar a codificar. Além disso, é recomendado ter uma implementação sem a utilização de *threads* para garantir que o proposto funciona corretamente. Seguir por esses passos irá facilitar a implementação do sistema de concorrências, para que o acúmulo de problemas de depuração não seja um empecilho.

7. Bibliografia

1. TANENBAUM, Andrew S. Sistemas Operacionais Modernos. Terceira edição. 2010.

A. Ncurses: Guia Simplificado

A.1. Introdução

Ncurses é uma biblioteca de programação que permite a criação de interfaces gráficas em modo texto.

A.2. Instruções de Compilação

Utilize a flag -lncurses para compilar programas que utilizam a biblioteca ncurses. Exemplo:

```
gcc -o programa codigo.c -lncurses
```

A.3. Funções e Constante

- **`LINES`**: número de linhas da tela
- **`COLS`**: número de colunas da tela
- **`initscr()`**: inicializa o terminal em modo curses
- **`endwin()`**: finaliza o terminal que está em modo curses
- **`clear()`**: limpa a tela
- **`echo()`**: habilita a exibição do que é digitado
- **`noecho()`**: desabilita a exibição do que é digitado
- **`keypad(stdsrc, TRUE)`**: habilita o uso de teclas especiais
- **`keypad(stdsrc, FALSE)`**: desabilita o uso de teclas especiais
- **`raw()`**: desativa o buffer de quebra de linha (CTRL+Z e CTRL+C continuam funcionando)
- **`noraw()`**: ativa o buffer de quebra de linha
- **`noraw()`**: desabilita o modo raw
- **`start_color()`**: inicializa o uso de cores
  - **`COLOR_BLACK`**
  - **`COLOR_RED`**
  - **`COLOR_GREEN`**
  - **`COLOR_YELLOW`**
  - **`COLOR_BLUE`**
  - **`COLOR_MAGENTA`**
  - **`COLOR_CYAN`**
  - **`COLOR_WHITE`**
- **`init_pair(número, cor_texto, cor_fundo)`**: inicializa um par de cores
- **`attron(atributos)`**: liga um atributo
- **`attroff(atributos)`**: desliga um atributo
  - **`A_NORMAL`**: exibe o texto normalmente
  - **`A_STANDOUT`**: destaca o texto
  - **`A_BOLD`**: exibe o texto em negrito
  - **`A_DIM`**: exibe o texto com menos brilho
  - **`A_UNDERLINE`**: sublinha o texto
  - **`A_REVERSE`**: inverte as cores do texto e do fundo
  - **`A_BLINK`**: exibe o texto piscando
  - **`A_ALTCHARSET`**: conjunto de caracteres alternativos para os mesmos caracteres
- **`printw("texto")`**: exibe um texto na tela (utilizado no lugar de **`printf`**)

- **`move(linha, coluna)`**: move o cursor para a posição especificada
- **`mvprintw(linha, coluna, "texto")`**: exibe um texto na posição especificada (combinação de **`move`** e **`printw`**)
- **`getch()`**: lê um caractere digitado
- **`getstr(string)`**: lê uma string digitada
- **`addch(caractere)`**: exibe um caractere na tela na posição atual

A.4. Material de Apoio

O link https://terminalroot.com.br/ncurses/ contém um Guia de Utilização da biblioteca ncurses, com exemplos de código e explicações mais detalhadas sobre várias funções.

A.5. Exemplo Simples de Código

```c
// gcc -o programa exemplo-ncurses.c -lncurses
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ncurses.h>

void imprime_conteudo(int tipo)
{
    attron(COLOR_PAIR(7));
    move(3, 0);
    printw("Par de cores %02d", tipo);
    move(4, 0);
    attron(COLOR_PAIR(tipo));
    for (int i = '0'; i <= 'z'; i++) {
        printw("%c", i);
    }
    attron(A_REVERSE);
    move(5, 0);
    attron(A_ALTCHARSET);
    for (int i = '0'; i <= 'z'; i++) {
        printw("%c", i);
    }
    attroff(A_ALTCHARSET);
    attroff(A_REVERSE);
    refresh();
}

int main()
{
    int tecla;
    initscr(); // Inicializa a tela (posição atual é (0, 0))
    start_color();
    raw();     // Não precisa esperar uma quebra de linha
    noecho(); // O que for digitado não aparece na tela
    keypad(stdscr, TRUE); // Teclas especiais como F1, F2, etc..

    // Cria as cores que serão utilizadas (mas ainda não usa)
    init_pair(7, COLOR_BLACK, COLOR_WHITE); // padrão
    init_pair(1, COLOR_BLUE, COLOR_WHITE);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(5, COLOR_RED, COLOR_BLACK);

    printw("Iniciando...");
    refresh(); // Realiza todos os comandos pendentes atualizando na tela
    sleep(1);

    attron(COLOR_PAIR(5));
    mvprintw(LINES - 1, 0, "(%d - %d) - Pressione 'q' para sair...", LINES, COLS);
    attroff(COLOR_PAIR(5));

    move(0, 0);
    printw(" - Digite 1, 2, q ou F4: ");
    do {
        tecla = getch();
        switch (tecla) {
            case '1':
                imprime_conteudo(1);
                break;
            case '2':
                imprime_conteudo(2);
                break;
            case KEY_F(4):
                attron(COLOR_PAIR(7));
                move(8, 0);
                printw("Tecla F4 foi pressionada.. saindo");
```

```
66                  refresh();
67                  sleep(1);
68              case 'q':
69              default:
70                  break;
71          }
72      } while (tecla != 'q' && tecla != KEY_F(4));
73
74      // Desabilitando tudo que foi habilitado antes de sair, para não deixar o
75      // terminal em estado diferente do anterior à execução
76      keypad(stdscr, FALSE);
77      noraw();
78      echo();
79
80      endwin();
81
82      printf("Termino da execucao\n");
83
84      return 0;
85  }
```