Trabalho Prático (T)

Identificação da Disciplina

Código da Disciplina	Nome da Disciplina	Professor	Período
CIC_5MA e EGS_5MA	Programação Concorrente	Jeremias Moreira Gomes	2025/1

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 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 crescente demanda por experiências digitais que desafiem habilidades técnicas e cognitivas, a IDP expandiu seu portfólio de simulações com foco em ambientes caóticos e altamente dinâmicos. Para isso, diversos *squads* foram montados com o objetivo de desenvolver simulações que permitam aos jogadores treinar sua capacidade de tomar decisões rápidas, coordenar tarefas concorrentes e manter o controle em situações de pressão.

Um dos jogos mais promissores dentro dessa proposta é o *Fora no Espaço*, um jogo cooperativo ambientado em uma nave espacial compartilhada por um grupo de jogadores. O desafio do jogo está em gerenciar os ambientes, de maneira funcional, habitável e sustentável, enquanto enfrenta desafios alienígenas e conflitos logísticos.

O jogador controla personagens com múltiplas responsabilidades. No entanto, para fins didáticos, o foco da simulação será especificamente nas tarefas de envolvendo a alimentação dos tripulantes da nave, ou seja a preparação de alimentos e bebidas, que possuem alta restrição de tempo e recursos, quando se vive em um ambiente espacial. Essas tarefas, embora pareçam simples, envolvem alto grau de coordenação entre os jogadores, já que os recursos são limitados, os equipamentos são compartilhados, e as ações precisam ser bem sincronizadas para garantir que todos sejam alimentados antes que a energia dos personagens se esgote.

Assim, o seu *squad* foi designado para desenvolver uma simulação do jogo *Fora no Espaço*, com ênfase no tratamento de tarefas concorrentes, onde múltiplos jogadores (ou agentes simulados) executam ações em paralelo, disputando por acesso a recursos compartilhados e otimizando o fluxo de trabalho na cozinha da nave.

3. Fora no Espaço

Na versão de treino/simulação do jogo, há diversas modificações em relação ao jogo original. A primeira delas é que, ao invés de combinar ingredientes, o tripulante prepara o pedido como um todo. Assim, há um painel da parte superior do jogo que apresenta apenas os nomes dos pedidos que estão pendentes de preparo, ou estão em preparação naquele momento.

Ao aparecer um novo pedido no painel, o responsável pela alimentação (chefe da cozianha) assinala o pedido para algum dos tripulantes disponíveis. O tripulante então ocupa uma bancada de preparo de ingredientes por um período específico de tempo, de acordo com a complexidade do prato. Por exemplo, o tempo de preparo dos ingredientes de um *hamburguer* é de 5 segundos, enquanto o tempo de preparo dos ingredientes de um suco é de 2 segundos. Após o tripulante terminar o preparo dos ingredientes, ele irá levar esse conteúdo para a cozinha de preparo do prato (fogão espacial (no nosso jogo existe), forno, etc). Este preparo também tem um tempo específico, e somente após esse término, é que o prato é entregue e o pedido pode sair do painel de pedidos. Por último, esse tripulante encontra-se novamente disponível para receber um novo pedido do chefe.

A figura 1 apresenta um esquema de organização do jogo.

É possível perceber que o número de tripulantes, bancadas e cozinhas 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 pedidos tiverem sido atendidos, ou quando o tempo de jogo acabar por acúmulo de pedidos não atendidos.

4. Implementação

A implementação do trabalho deve ser feita em **C**. A organização dos arquivos fica a cargo do aluno/dupla (spoiler), o qual deverá conter instruções de compilação, bem como um arquivo `Makefile` para facilitar

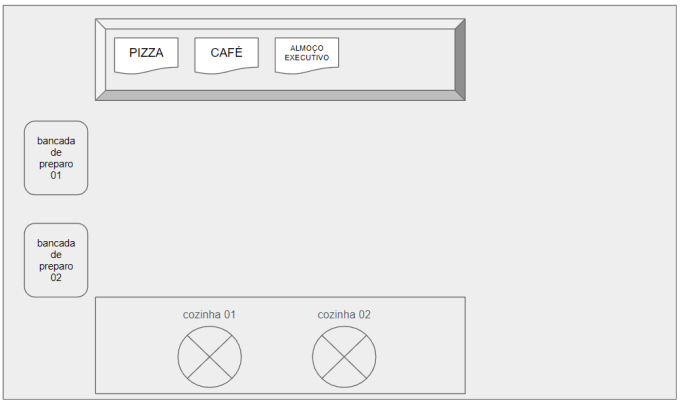

Figura 1. Esquema de Organização do Jogo

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 Pedidos da Nave**
- **Exibição de Informações**
- **Chefe da Cozinha**
- **Tripulantes (s)**

4.1.1. Mural de Pedidos da Nave

Essa entidade é responsável por gerar novos pedidos para o jogo. De tempos em tempos, estipulado pela sua organização no jogo (exemplo: pela dificuldade do jogo), essa *thread* irá gerar um novo pedido, que será inserido em uma estrutura, tanto para a saída do jogo (tela a ser utilizada por outra *thread*) quanto para a organização do chefe da cozinha de assinalar o pedido para um tripulante. Um exemplo de estrutura que pode ser utilizada para representar um pedido é uma lista encadeada, para que seja possível adicionar novos pedidos e retirar pedidos já atendidos, mesmo fora de ordem[1].

4.1.2. Exibição de Informações

Essa entidade é responsável por exibir as informações do jogo para o jogador. Exemplos: lista de pedidos acumulados, bancada de preparo de ingredientes disponível ou ocupada por qual tripulante, cozinha de preparo de pratos disponível ou ocupada por qual tripulante, tempo de jogo, tempo restante, ou disponibilidade de tripulantes.

[1]Essa decisão por estruturas, fica a cargo da dupla/do aluno. A lista encadeada é apenas um exemplo.

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 eles 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. Tripulantes (s)

O tripulante é responsável por preparar os pratos. O preparo do prato inicia quando o chefe da cozinha assinala um pedido e uma bancada disponível para o tripulante. Cada prato precisa passar por duas etapas: a preparação dos ingredientes e o preparo do prato.

Cada prato possui um tempo específico de preparo dos ingredientes e um tempo específico de preparo do prato. Do ponto de vista técnico, o tripulante sendo uma *thread*, ele irá ocupar o recurso (bancada de preparo de ingredientes ou cozinha de preparo de pratos) por um tempo específico, e após esse tempo, ele irá liberar o recurso para que outro tripulante possa utilizá-lo. Com um único tripulante, o gerenciamento de recursos de preparo é simplificado, porém deve-se pensar em como será a concorrência entre os tripulantes, para que não haja conflitos de acesso aos recursos.

Ao terminar o preparo do prato, o pedido deve ser retirado da lista de pedidos pendentes, e o tripulante deve ser liberado para receber um novo pedido. Essa retirada pode ser feita tanto pelo chefe quanto pelo tripulante (provavelmente mais fácil ser direto pelo tripulante, mas é decisão de implementação), e deve ser feita de forma segura, para que não haja conflitos de acesso à lista de pedidos.

Recomenda-se iniciar a implementação com um único tripulante, e uma única bancada de preparo de ingredientes e cozinha de preparo de pratos, para somente depois estender a implementação para múltiplos tripulantes e bancadas.

4.1.4. Chefe da Cozinha

O chefe é responsável por assinalar os pedidos para os tripulantes. Essa entidade (*thread*) é quem irá coletar os *inputs* do usuário. Com a lista de pedidos diferente de vazia, o chefe pode assinalar um pedido para um tripulante, caso ele esteja disponível e uma bancada de preparo de ingredientes também o 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 duas teclas pressionadas pelo usuário, e caso estas sejam um comando válido, realizar a ação desejada. Exemplo: se o usuário pressionar as teclas `1` e em seguida `p`, o gerente está dando a ordem de "O tripulante 1 realizar o preparo do pedido Pizza"; nesse caso, o primeiro caractere identifica o tripulante, e o segundo caractere identifica o prato a ser realizado[2].

[2]De novo, 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.

4.2. Sumário

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

5. Metodologia

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

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 tripulantes, e múltiplas bancadas de preparo de ingredientes; 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 tripulantes, com o mesmo número de bancadas de preparo de ingredientes e cozinhas de preparo de pratos[3].

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 22.04).

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 Tripulantes e Bancadas (Bônus de 10 pts)

Se a quantidade de tripulantes e bancadas de preparo puderem ser diferentes, onde múltiplos tripulantes podem se enfileirar na espera pelo uso de bancadas de preparo dos pratos, este receberá uma pontuação adicional de até 10 pontos.

Exemplo: quando um tripulante terminar de utilizar a bancada de preparo e se deslocar para a cozinha de preparo, a bancada de preparo está disponível para outro tripulante, que pode ocupá-la e começar a preparar os ingredientes do prato.

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 13/06/2025 (sexta-feira).

6.2. Datas Importantes

Esse trabalho poderá ser submetido a partir de 05/06/2025 (sexta-feira) - 12:00h até 29/06/2025 (domingo) - 23:55h.

[3] Exemplo: se há dois tripulantes, existem duas bancadas e preparo e duas cozinhas de preparo de pratos.

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[4]. 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: `22001101-22011000-3b1b448e9a49cd6a91a1ad044e67fff2.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.

[4] 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  }
```