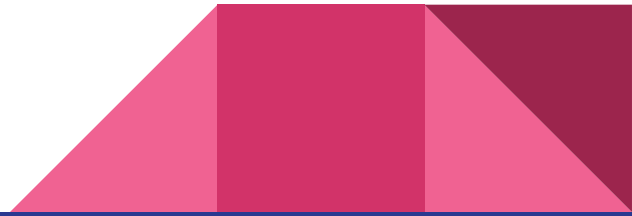
Ticket System

Sistema de atendimento de cinema multithreaded.

Luis Felipe de Azambuja Feyh
Pedro Henrique Gomes Magri

Introdução

Este trabalho tem como objetivo demonstrar o conhecimento de programação paralela dos membros do grupo, utilização de threads, estruturas de sincronização e acesso exclusivo a variáveis compartilhadas e uso de padrões de projeto.



Características não funcionais do trabalho

O trabalho foi desenvolvido na linguagem C, usando a biblioteca pthread.h

Os padrões de projeto utilizados foram: Produtor / Consumidor, Pool de Threads e Suspensão Controlada.

As estruturas de sincronização usadas foram: Mutex e Variáveis de Condição.

Padrões de projeto e estruturas de sincronização

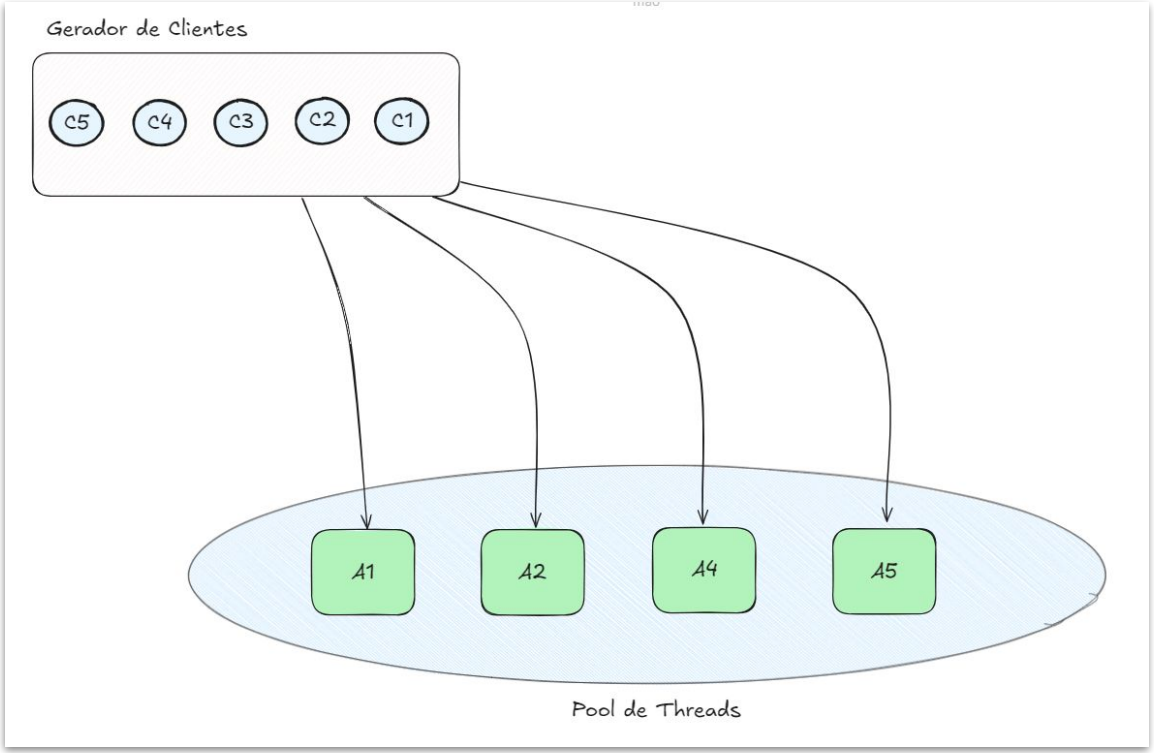
Padrões de projeto

- Pool de Threads
  - Conjunto de atendentes processam pedidos dos clientes paralelamente
- Suspensão Controlada
  - Uso de variáveis de condição para checar estado da fila
- Produtor/Consumidor
  - Novos clientes são gerados e “consumidos” pelas threads de atendentes

Estruturas de sincronização

- Mutex
  - Assentos do cinema e fila de clientes
- Variáveis de condição
  - Suspensão dos atendentes se a fila estiver vazia

Representação visual



Produtor/Consumidor

```c
void *gerarClientes(void *args)
{
  int tempo_bilheteria = *(int *)args;
  int id_cliente = 1;
  unsigned int seed = time(NULL) ^ pthread_self();
  time_t inicio = time(NULL);

  while (time(NULL) - inicio < tempo_bilheteria)
  {
    Cliente novo_cliente;
    novo_cliente.id = id_cliente++;
    novo_cliente.assento_desejado = rand_r(&seed) % num_assentos;

    pthread_mutex_lock(&mutexFila);
    if (qntd_clientes_na_fila < 64)
    {
      fila[qntd_clientes_na_fila++] = novo_cliente;
      printf("[CLIENTE %d] Entrou na fila e deseja o assento %d. \n", novo_cliente.id, novo_cliente.assento_desejado);
      pthread_cond_signal(&condFila);
    }
    else
    {
      printf("[CLIENTE %d] Fila cheia! O cliente não conseguiu entrar. \n", novo_cliente.id);
    }
    pthread_mutex_unlock(&mutexFila);

    int tempo_aleatorio = (rand_r(&seed) % 5) + 1;
    sleep(tempo_aleatorio);
  }

  return NULL;
}
```

Produtor/gerador de clientes

1. Gera novo cliente com ID sequencial e assento desejado aleatório
2. Adquire tranca da fila
3. Inclui cliente na fila se o número de clientes é menor que o limite estabelecido
4. Libera tranca da fila
5. Aguarda tempo randômico para executar novamente

Pool de threads e Suspensão Controlada

```c
void *inicializarAtendente(void *args)
{
    int atendente_id = *(int *)args;

    free(args);

    printf("[ATENDENTE %d] Bilheteria aberta!\n", atendente_id);

    while (1)
    {

        Cliente cliente;

        pthread_mutex_lock(&mutexFila);
        while (qntd_clientes_na_fila == 0 && sistema_ativo)
        {
            printf("[ATENDENTE %d] Aguardando clientes.\n", atendente_id);
            pthread_cond_wait(&condFila, &mutexFila);
        }

        if (!sistema_ativo)
        {
            pthread_mutex_unlock(&mutexFila);
            printf("[ATENDENTE %d] Encerrando atividades.\n", atendente_id);
            return NULL;
        }

        // Chama o primeiro cliente da fila
        cliente = fila[0];

        // Faz a fila "andar"
        for (int i = 0; i < qntd_clientes_na_fila - 1; i++)
        {
            fila[i] = fila[i + 1];
        }
        qntd_clientes_na_fila--;
        pthread_mutex_unlock(&mutexFila);

        // Atende o cliente
        atenderCliente(&cliente, atendente_id);
    }

    return NULL;
}
```

Pool de threads consumidoras (Atendentes)

1. Adquire bloqueio da fila
2. Verifica por variável de condição o estado da fila
    a. Se a fila estiver vazia, aguarda clientes
3. Se houver clientes, captura valor do primeiro cliente e atualiza fila
4. Libera tranca da fila
5. Atende o cliente

Arquivo Main

```c
int main(int argc, char *argv[])
{
  srand(time(NULL));
  pthread_t atendentes[NUM_ATENDENTES];
  pthread_t geradorClientes;
  pthread_t threadEntrada;
  int tempo_bilheteria = 60;

  inicializarAssentos();

  for (int i = 0; i < NUM_ATENDENTES; i++)
  {
    usleep(10000);
    int *id = malloc(sizeof(int)); // aloca memória separada
    if (id == NULL)
    {
      perror("malloc");
      exit(1);
    }
    *id = i + 1;

    pthread_create(&atendentes[i], NULL, inicializarAtendente, id);
  }

  pthread_create(&threadEntrada, NULL, entradaUsuario, &tempo_bilheteria);
  pthread_create(&geradorClientes, NULL, gerarClientes, &tempo_bilheteria);

  pthread_join(geradorClientes, NULL);
  pthread_join(threadEntrada, NULL);
  printf("[SISTEMA] Horário da bilheteria encerrado.\n");
  encerrarSistema();

  for (int i = 0; i < NUM_ATENDENTES; i++)
  {
    pthread_join(atendentes[i], NULL);
  }

  pthread_mutex_destroy(&mutexFila);
  pthread_cond_destroy(&condFila);

  return 0;
}
```

Inicialização do programa

1. Define todas as threads e o tempo que a bilheteria ficará aberta.
2. Preenche o array cinema de assentos
3. Inicializa as threads atendentes, cada uma com um id único, depois inicializa as outras threads.
4. Espera o join das threads e espera o tempo acabar
5. Destrói os mutex e variáveis de condição.