Exercícios de Modelagem: Semáforos e Monitores

Introdução

Esta apostila contém 10 questões práticas de modelagem divididas em duas seções. Cada seção tem 5 questões com dificuldade progressiva, começando com problemas básicos e evoluindo para sistemas complexos. Use as metodologias dos guias anteriores para resolver cada problema.

Lembre-se:

- **Semáforos**: Foque em recursos limitados, contagem, e sincronização de eventos
- **Monitores**: Foque em dados encapsulados, operações complexas, e múltiplas condições de espera

---

SEÇÃO 1: MODELAGEM COM SEMÁFOROS

Questão 1 - Nível Básico: Laboratório de Computação

Você precisa modelar o controle de acesso a um laboratório de computação que possui 15 computadores. Estudantes chegam, usam um computador por um tempo, e depois saem. Se não há computadores disponíveis, os estudantes devem aguardar na fila.

Implemente usando semáforos POSIX:

- A função `usar_computador(int estudante_id)` que simula um estudante usando o lab
- A função deve imprimir quando o estudante entra, usa o computador, e sai
- Teste com 25 estudantes tentando usar o lab simultaneamente

Considere:

- Quantos semáforos você precisa?
- Qual o valor inicial de cada semáforo?
- Onde colocar sem_wait() and sem_post()?

Questão 2 - Nível Básico-Intermediário: Sistema de Pedidos Online

Um restaurante online processa pedidos em 3 etapas sequenciais:

1. **Recebimento**: Cliente faz pedido
2. **Preparação**: Cozinha prepara o pedido
3. **Entrega**: Entregador leva o pedido

Cada etapa só pode começar depois que a anterior terminou. O sistema deve coordenar essas etapas para múltiplos pedidos simultâneos.

Implemente usando semáforos:

- Três threads: `receber_pedido()`, `preparar_pedido()`, `entregar_pedido()`
- Cada thread processa múltiplos pedidos em loop
- Garanta que a ordem seja respeitada: recepção → preparação → entrega
- Imprima o progresso de cada pedido

Considere:

- Como sincronizar a sequência de etapas?
- Quantos semáforos para coordenar as três etapas?
- Como garantir que não haja deadlock entre as etapas?

Questão 3 - Nível Intermediário: Ponte Estreita Bidirecional

Uma ponte estreita só permite passagem de carros em uma direção por vez. Carros chegam de ambos os lados (Norte e Sul) e querem atravessar para o outro lado. Para evitar acidentes:

- Apenas carros de uma direção podem estar na ponte simultaneamente
- No máximo 3 carros podem estar na ponte ao mesmo tempo (peso máximo)
- Carros devem alternar direções para evitar starvation (após 3 carros de uma direção, a ponte deve permitir carros da direção oposta)

Implemente usando semáforos:

- Funções `atravessar_norte_sul(int carro_id)` e `atravessar_sul_norte(int carro_id)`
- Controle de alternância de direções
- Limite de 3 carros simultâneos na ponte
- Teste com 10 carros de cada direção

Considere:

- Como controlar a direção atual da ponte?
- Como contar carros na ponte?
- Como implementar alternância justa?
- Quantos semáforos são necessários e para que serve cada um?

Questão 4 - Nível Intermediário-Avançado: Sistema de Backup Distribuído

Um sistema de backup tem 3 servidores de armazenamento, cada um com capacidades diferentes:

- **Servidor A**: Aceita até 2 backups simultâneos (rápido, para arquivos pequenos)
- **Servidor B**: Aceita até 1 backup simultâneo (médio, para arquivos médios)
- **Servidor C**: Aceita até 3 backups simultâneos (lento, para arquivos grandes)

Clientes fazem backup especificando o tipo de arquivo. O sistema deve:

- Direcionar arquivos pequenos preferencialmente para servidor A
- Se A estiver ocupado, pequenos podem ir para B ou C
- Arquivos médios vão para B, mas podem usar C se B ocupado
- Arquivos grandes só podem usar servidor C
- Implementar política de fallback inteligente

Implemente usando semáforos:

- Função `fazer_backup(int cliente_id, char tipo_arquivo)` onde tipo é 'P', 'M', ou 'G'
- Sistema de fallback: se servidor preferido ocupado, tenta alternativas
- Relatório de qual servidor foi usado para cada backup
- Teste com 15 clientes fazendo backups aleatórios

Considere:

- Como modelar as capacidades diferentes dos servidores?
- Como implementar a lógica de fallback?
- Como evitar que fallback cause starvation?
- Como garantir que as políticas de direcionamento sejam respeitadas?

Questão 5 - Nível Avançado: Sistema de Reuniões Virtuais Complexo

Uma empresa tem salas de reunião virtuais com características específicas:

- **3 salas pequenas**: máximo 4 participantes cada, equipadas com projetor básico
- **2 salas médias**: máximo 8 participantes cada, equipadas com projetor + quadro digital
- **1 sala grande**: máximo 15 participantes, equipamentos completos + gravação

Regras de negócio complexas:

- Reuniões são agendadas com antecedência especificando: número de participantes, duração, equipamentos necessários

- Se a sala ideal estiver ocupada, o sistema pode sugerir divisão de grupos ou upgrade de sala
- VIPs têm prioridade e podem "expulsar" reuniões menos importantes (com 5 min de aviso)
- Salas têm tempo de limpeza de 10 minutos entre reuniões
- Sistema deve implementar fila de espera inteligente que considera horários de término

Implemente usando semáforos:

- Função `agendar_reuniao(int reuniao_id, int participantes, int duracao_min, char* equipamentos, bool eh_vip)`
- Função `terminar_reuniao(int reuniao_id)`
- Sistema de fila de espera que reordena baseado em prioridades e horários
- Função de limpeza automática das salas
- Relatório de ocupação e filas de espera

Considere:

- Como modelar salas com características diferentes?
- Como implementar sistema de prioridades com VIPs?
- Como gerenciar tempo de limpeza entre reuniões?
- Como coordenar múltiplas filas de espera para diferentes tipos de sala?
- Como implementar "expulsão" de reuniões menos importantes?
- Quantos semáforos são necessários e como eles interagem?

SEÇÃO 2: MODELAGEM COM MONITORES

Questão 6 - Nível Básico: Conta Bancária Compartilhada

Uma família possui uma conta bancária compartilhada que pode ser acessada por múltiplos membros da família simultaneamente. A conta tem as seguintes regras:

- Saldo inicial de R$ 5.000
- Múltiplos saques podem ocorrer simultaneamente, desde que o saldo não fique negativo
- Depósitos podem ocorrer a qualquer momento
- O sistema deve manter histórico das últimas 10 transações
- Deve existir operação para consultar saldo atual

Implemente um monitor ContaBancaria com:

- Método `sacar(int valor, string membro_familia)`
- Método `depositar(int valor, string membro_familia)`

- Método `consultar_saldo()`
- Método `obter_historico()`
- Teste com 5 membros da família fazendo transações aleatórias

Considere:

- Que dados precisam ser protegidos juntos?
- Quando uma operação de saque deve esperar?
- Como manter consistência entre saldo e histórico?

Questão 7 - Nível Básico-Intermediário: Sistema de Reservas de Cinema

Um cinema possui 3 salas com lotações diferentes:

- **Sala 1**: 50 lugares
- **Sala 2**: 80 lugares
- **Sala 3**: 120 lugares

Cada sala tem sessões em horários diferentes. Clientes podem:

- Consultar disponibilidade de lugares em uma sessão específica
- Reservar lugares (especificando quantidade)
- Cancelar reservas
- Listar todas as reservas de um cliente

Implemente um monitor Cinema com:

- Método `consultar_disponibilidade(int sala, string horario)`
- Método `reservar_lugares(int cliente_id, int sala, string horario, int quantidade)`
- Método `cancelar_reserva(int cliente_id, int sala, string horario)`
- Método `listar_reservas_cliente(int cliente_id)`
- Estrutura para armazenar sessões e suas ocupações

Considere:

- Como organizar dados de múltiplas salas e sessões?
- Que condições de espera são necessárias?
- Como garantir atomicidade nas reservas múltiplas?
- Como tratar cancelamentos e disponibilizar lugares novamente?

Questão 8 - Nível Intermediário: Sistema de Entrega com Rastreamento

Uma empresa de delivery precisa coordenar pedidos, entregadores e clientes. O sistema deve gerenciar:

- **Pedidos**: com status (preparando, pronto, saiu_entrega, entregue)
- **Entregadores**: disponíveis ou ocupados, cada um com localização atual
- **Clientes**: podem acompanhar status do pedido em tempo real

Regras de negócio:

- Pedidos só podem ser atribuídos a entregadores disponíveis
- Entregadores próximos à origem do pedido têm prioridade
- Clientes podem cancelar pedidos apenas se ainda não saíram para entrega
- Sistema deve notificar clientes sobre mudanças de status

Implemente um monitor SistemaEntrega com:

- Método `criar_pedido(int cliente_id, string endereco)`
- Método `marcar_pedido_pronto(int pedido_id)`
- Método `atribuir_entregador(int pedido_id, int entregador_id)`
- Método `atualizar_status_entrega(int pedido_id, string novo_status)`
- Método `cancelar_pedido(int cliente_id, int pedido_id)`
- Método `consultar_status_pedido(int cliente_id, int pedido_id)`
- Sistema de notificações para clientes

Considere:

- Como modelar relacionamentos entre pedidos, entregadores e clientes?
- Que condições de espera existem para cada tipo de operação?
- Como implementar sistema de notificações thread-safe?
- Como tratar cancelamentos em diferentes estados do pedido?

Questão 9 - Nível Intermediário-Avançado: Sistema de Hospital com Emergências

Um hospital precisa gerenciar atendimento em diferentes setores:

- **Triagem**: classifica pacientes por prioridade (1=crítico, 2=urgente, 3=normal)
- **Emergência**: 3 médicos, atendem apenas prioridade 1 e 2
- **Consulta Geral**: 5 médicos, atendem prioridade 3 e urgências se necessário

- **Exames**: 2 equipamentos, qualquer prioridade pode usar

Regras complexas:

- Pacientes críticos (prioridade 1) têm acesso imediato, mesmo interrompendo consultas normais
- Médicos de consulta geral podem ser "requisitados" para emergências se todos os médicos de emergência estiverem ocupados
- Pacientes aguardam em filas separadas por prioridade
- Sistema deve rastrear tempo de espera e alertar se passar de limites (crítico: 0min, urgente: 30min, normal: 2h)

Implemente um monitor Hospital com:

- Método `registrar_paciente(int paciente_id, int prioridade, string sintomas)`
- Método `atender_emergencia(int medico_id)`
- Método `atender_consulta_geral(int medico_id)`
- Método `liberar_medico(int medico_id, int paciente_id)`
- Método `realizar_exame(int paciente_id)`
- Sistema de alertas por tempo de espera excessivo
- Métricas de ocupação e tempo médio de atendimento

Considere:

- Como modelar filas de prioridade diferentes?
- Como implementar interrupção de consultas para emergências?
- Como gerenciar "empréstimo" de médicos entre setores?
- Como implementar sistema de alertas baseado em tempo?
- Como garantir que pacientes críticos sempre tenham prioridade absoluta?

Questão 10 - Nível Avançado: Sistema de Controle de Tráfego Aéreo

Um aeroporto precisa coordenar pousos e decolagens em 2 pistas com características diferentes:

- **Pista 1**: Pode ser usada para pouso OU decolagem (não simultâneo), aceita aviões de qualquer tamanho
- **Pista 2**: Apenas para decolagens, só aceita aviões pequenos e médios

Regras operacionais complexas:

- Pousos têm prioridade absoluta sobre decolagens (combustível limitado)
- Aviões grandes só podem usar Pista 1

- Condições climáticas podem fechar pistas temporariamente
- Slots de decolagem são agendados, mas podem ser reorganizados por emergências
- Sistema deve manter separação mínima entre operações (3 minutos entre pousos/decolagens na mesma pista)
- Controle de tráfego pode implementar "holding pattern" (aviões ficam aguardando em círculo)

Implemente um monitor ControleTrafegoAereo com:

- Método `solicitar_pouso(int voo_id, char tamanho_aviao, int combustivel_restante)`
- Método `solicitar_decolagem(int voo_id, char tamanho_aviao, int slot_agendado)`
- Método `liberar_pista(int pista_id, int voo_id)`
- Método `fechar_pista(int pista_id, string motivo)`
- Método `abrir_pista(int pista_id)`
- Método `declarar_emergencia(int voo_id)`
- Sistema de holding pattern para aviões aguardando
- Reorganização automática de slots por prioridades
- Controle de separação temporal entre operações

Considere:

- Como modelar pistas com capacidades e restrições diferentes?
- Como implementar sistema de prioridades multinível?
- Como gerenciar separação temporal entre operações?
- Como tratar emergências que reorganizam toda a fila?
- Como implementar holding pattern de forma thread-safe?
- Como coordenar fechamento/abertura de pistas com operações em andamento?
- Como garantir que aviões com pouco combustível tenham prioridade absoluta?

Instruções de Implementação

Para cada questão:

1. **Analise o problema** usando a metodologia apropriada (semáforos ou monitores)
2. **Identifique** todos os recursos, dados compartilhados, e condições de sincronização
3. **Projete** a estrutura de dados e operações necessárias
4. **Implemente** a solução completa em C++
5. **Teste** com cenários diversos, incluindo casos extremos

6. **Documente** suas decisões de design e justifique escolhas técnicas

Critérios de avaliação:

- Corretude da sincronização (sem deadlocks, race conditions)
- Completude da implementação (todos os requisitos atendidos)
- Qualidade do design (código limpo, bem estruturado)
- Tratamento de casos especiais e erros
- Eficiência da solução (performance adequada)

Boa sorte com os exercícios! Lembre-se de aplicar as metodologias dos guias para abordar cada problema de forma sistemática.