 # Trabalho Prático de Programação Paralela e Concorrente - Samir Avelino Carvalho
Trabalho usando semáforos para controlar a concorrência e a sincronização entre as threads.

Algoritmo Escolhido: C++

Forma de compilação e execução:

Na pasta do winrar, com o arquivo digitar o seguinte comando no terminal:

clang++-7 -pthread -std=c++17 -o main main.cpp src/barman.cpp src/cliente.cpp src/comanda.cpp src/garcom.cpp src/gerente.cpp src/pedido.cpp src/threadedClass.cpp

aperte enter e logo após digite o comando:

./main

Funcionamento do algoritmo:

O algoritmo possui o main e as pastas include e src.

A pasta include contém os arquivos de extensão hpp. São arquivos de cabeçalho para as classes do programa contendo as sub-rotinas, as variáveis, as declarações para a frente de classes.

Na pasta src, contém os arquivos cpp de código fonte.

Os arquivos no include e src estão separados em ("barman, cliente, comanda, garcom, gerente, pedido, threadedClass").

- Barman: A classe barman foi criada para controlar a capacidade de atendimento dos garçons. No barman.hpp possui como protected: um contador de instâncias da classe, uma função para ser colocada como thread, uma função para preparar mais bebida e atualizar o valor de forma atômica. Como public possui uma função que cria a thread.
  Já na barman.cpp: Cria a thread e executa a função run, passando como argumento uma referencia da instancia, possui tb um id para contar o numero de instancias no momento de criação.
  Então se executa o comportamento da classe e se recupera o gerente, acessando as variaveis globais e semáforos. barman.wait() aguarda a liberação do lock. Depois, enquanto o bar está aberto o barman espera um pedido chegar. Um comando para NULL caso algum erro de comanda nula chegar. Caso contrário o barman pega a comanda. Enquanto prepara ele verifica a disponibilidade dos ingredientes, se acabou ele repõe. O barman então prepara a bebida, e devolve o pedido para a fila de comanda.

- Cliente: No cliente.hpp, como privado possui uma função para thread, uma função para decidir qual pedido fazer. Como publico possui um contador de instancias, um pedido para fazer, um construtor padrão, e um semaforo unico

do cliente que está aguardando o pedido. void start -> herança que cria a thread.

Na cliente.cpp possui um id para o numero de instancias no momento de criação, e possui sem_init(&sem_pedido_chegou,0,0) para o pedido que ainda não chegou. pthread_create(&thread, NULL, run, this) para startar a thread. A lógica da função cliente run: Se o bar está sem mesas disponiveis o cliente aguarda na fila por uma mesa if (sem_trywait(&g.sem_mesas) != 0). Caso contrario o cliente pegou uma mesa no bar, cliente->decidir() para escolher o pedido, g.cliente_para_fila(cliente) para entrar na fila de atendimento. Então espera o pedido voltar e logo após pega sua bebida e começa a beber, depois de beber libera a mesa sem_post(&g.sem_mesas). No decidir() o cliente escolhe sua bebida de forma random, podendo escolher entre 3 bebidas diferentes.

- Comanda: Na comanda possui uma abstração que vincula um pedido a um cliente. Pedido* p é para o pedido que o cliente fez. Cliente* c é o cliente que fez o pedido. Comanda(Cliente* c,Pedido* p) construtor padrão.

- Garcom: No garcom.hpp, classe que representa um garcom, possui uma herança para função que cria a thread. void atender() recebe um pedido de um cliente e void servir() leva uma bebida a um cliente
  No garcom.cpp,O garçom espera por um cliente para atender e recebe uma chamada de atendimento. O garçom atende o cliente, prepara a comanda e envia para a fila. garcom.atender() atendimento é pedido pronto e levar pedido para cliente garcom.servir(). Ao final o garçom volta se o bar ainda está aberto

- Gerente:

No gerente.hpp possui, variáveis globais (número de clientes, números de garçons, números de barman), possui uma quantidade de mesas que serve para limitar o número de atendimento no bar, por exemplo, se houver 10 clientes e 6 mesas, 4 clientes ficarão na fila de espera até que os clientes na mesa terminam sua bebida e liberam a mesa. Número de mesas no bar definido como 5.
Possui N_Bebidas 3 para quantidade de Bebidas diferentes disponíveis no bar para os clientes escolherem. MAX_BEBIDAS 2 para um limite de bebidas para a preparação ao mesmo tempo, definido como podendo ser preparadas duas bebidas ao mesmo tempo.
A classe possui como privado, um construtor privado para usar singleton, ou seja, para que apenas uma instância da classe possa existir e ser utilizada por toda aplicação. Como publico, variáveis globais "prioridade" para controle de prioridade nos pedidos novos. setN_Cliente, setN_Garcom, setN_Barman, para receber como

parâmetros as variáveis de entrada na main. bebidas[3] contagem de bebidas disponíveis.
O pthread_rwlock_t barrier_lock = PTHREAD_RWLOCK_INITIALIZER se usa um lock RW como barreira para as classes, todas as classes aguardam esse lock ser liberado para leitura para continuar, assim nenhuma thread tenta acessar nada até que tudo esteja preparado.
sem_t sem_mesas para controle de acesso por volume, representa clientes que possuem uma mesa no bar.
sem_t sem_sinal_atendimento para controle de fluxo sinais representa um sinal de atendimento (pedido pronto ou cliente chamou)
sem_t sem_sinal que representa um sinal de preparo para o pedido (pedido chegou)
sem_t sem_sinal_fim_expediente que representa o fim do expediente, todos os clientes foram atendidos
sem_t sem_controle_clientes que opera como um lock para interagir com a fila de clientes
sem_t sem_controle_comandas que opera como um lock para interagir com a fila de comandas
sem_t sem_controle_bebidas que opera como um lock para interagir com a fila de bebidas
sem_t sem_controle_contador que opera como um lock para interagir com a contagem de clientes no estabelecimento (encerramento do programa)
sem_t sem_controle_bebida que opera como um lock para interagir com os recursos disponíveis para fazer a bebida
void abrir_bar() libera as threads
static Gerente& getManager() pega o singleton do gerente.
void cliente_para_fila(Cliente* c) insere cliente na fila de atendimento de forma atômica e emite sinal para garçom
Cliente* atender_cliente() retira e retorna o primeiro cliente da fila de forma atômica.
void comanda_para_fila(Comanda* c) insere comanda na fila de preparo de forma atômica e emite sinal
Comanda* pegar_comanda() retira e retorna a primeira comanda da fila de forma atômica.
void bebida_para_fila(Comanda* p) insere comanda na fila de comandas prontas de forma atômica e emite sinal para garçom
Comanda* pegar_bebida() retira e retorna ao primeiro da fila de forma atômica.

No gerente.cpp possui um construtor e um destrutor.
void Gerente::abrir_bar() libera todas as threads
Gerente::cliente_para_fila enfileira cliente e emite sinal para atendimento (garcom), sem_wait(&sem_controle_clientes) concorrencia no acesso a fila, sem_post(&sem_sinal_atendimento) adiciona na fila e avisa que precisa de atendimento
Cliente* Gerente::atender_cliente() desenfileira e retorna cliente para atendimento

void Gerente::comanda_para_fila(Comanda* c) enfileira comanda e envia sinal
Comanda* Gerente::pegar_comanda() desenfileira e retorna comanda
Comanda* Gerente::pegar_bebida() desenfileira e retorna(representado por uma comanda).
Gerente::Gerente() Inicializa todos os semáforos e recursos
pthread_rwlock_wrlock(&barrier_lock) bar começa fechado
sem_init(&sem_mesas, 0, N_MESAS) quantidade de lugares para os clientes inicialmente;
sem_init(&sem_sinal, 0, 0); nenhum sinal requisitado inicialmente
somente 1 thread pode operar por vez { sem_init(&sem_controle_comandas,0,1); sem_init(&sem_controle_clientes,0,1); sem_init(&sem_controle_bebidas,0,1); sem_init(&sem_controle_contador,0,1);
estoques começam vazios bebidas[i] = 0

Pedido: Temos as opções de bebidas enum Bebida. Mas cada cliente está restringido a um pedido. Possui um contador de instâncias, um id para o id do pedido, Bebida para a bebida e um construtor específico Pedido(int p) e um construtor padrão para alocação estática de atributos de cliente Pedido().

ThreadedClass: A threadClass é uma classe para todas as threads. É necessário implementar uma função estática em cada classe que for derivada desta, para ser o ponteiro
Na classe ThreadedClass possui como protegido, pthread_t thread que é a thread da instancia e void wait() // aguarda liberação do lock para continuar executando. (aguardar barreira/broadcast)
Como publico possui um id, para o id da instancia, virtual void start(void) = 0 onde todas as classes tem implementar para criação de thread

Na threadClass.cpp possui ThreadedClass::join() o join, e pthread_rwlock_rdlock(&(Gerente::barrier_lock)) aguarda para leitura lock que é trancado antes de qualquer thread iniciar