Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação

Trabalho Final – Simulador de Fila de Banco
Disciplina Programação (CK0226) – Semestre 2022.2

Prof. Miguel Franklin

Desenvolver um projeto em linguagem C representando um sistema de simulação de fila de banco.

O Banco da Cochinchina está precisando de um sistema para organizar as filas de atendimento aos seus caixas. Existem 10 caixas na agência do banco. Em cada dia de serviço, *M* caixas estão operacionais, onde $1 \leq M \leq 10$. Existem 5 classes de clientes que utilizam os serviços desses bancos, cada uma identificada por um número:

1. Cliente Premium
2. Cliente Ouro
3. Cliente Prata
4. Cliente Bronze
5. Cliente Leezu

O banco quer garantir um tratamento diferenciado aos seus clientes mais importantes, que estão listados acima em ordem decrescente de "importância". Como consequência, há 5 filas distintas, dependendo da classe de cliente.

Cada cliente que chega ao banco é caracterizado pelo número de sua conta (inteiro), pela classe da sua conta (no intervalo [1;5], conforme definido acima) e a quantidade (inteira) de operações financeiras que ele deseja efetuar.

A disciplina de atendimento em cada fila individual por classe de conta é o primeiro a chegar é o primeiro a sair. Entre as diferentes filas, há um ESCALONADOR que segue o seguinte procedimento para cada rodada de atendimento:

Passo 1. Atender N_1 cliente(s) Premium, se houver;
Passo 2. Em seguida, atender N_2 cliente(s) Ouro, se houver;
Passo 3. Em seguida, atender N_3 cliente(s) Prata, se houver;
Passo 4. Em seguida, atender N_4 cliente(s) Bronze, se houver;
Passo 5. Em seguida, atender N_5 cliente(s) Leezu, se houver, e ir para o primeiro passo da próxima rodada.

Ao conjunto $\{N_1; N_2; N_3; N_4; N_5\}$, denominamos "**Disciplina de Atendimento**".

Imagine que há, na frente de todas as filas, um painel que chama o próximo cliente, indicando o número de conta do correntista a ser atendido, logo que um caixa está disponível, seguindo-se a Disciplina de Atendimento escolhida. Caso haja mais de um caixa disponível em um determinado momento, um cliente será atribuído ao caixa de menor número disponível.

Para determinar o status de cada caixa, o programa deve ter um *timer* para cada caixa, que cada vez que um cliente que vai fazer *M* operações financeiras segue para ser atendido, o timer é definido para chamar outro cliente dentro de $M \times \Delta T$ min., onde ΔT é o tempo necessário para o caixa processar cada operação financeira. Considere que o valor de ΔT é dado como parâmetro.

Considere que o banco tenha uma forma peculiar de atender os seus clientes: primeiramente as filas são formadas, e só depois o serviço dos caixas se inicia. Não há chegada de novos clientes depois que os caixas começam a trabalhar. O relógio começa a contar no momento em que o primeiro cliente começa a ser atendido.

O programa de organização de filas deve ser capaz de:

1. Executar a simulação do atendimento de todos os clientes de todas as filas, seguindo uma Disciplina de Atendimento predefinida e adotando-se uma quantidade de caixas predefinida (menor ou igual a 10, e maior ou igual a 1);
2. Calcular o tempo de espera de todos os clientes, que será identificado pelo seu número de conta. O tempo de espera é o tempo decorrido desde o início do atendimento dos caixas até a momento que o cliente é chamado no painel;
3. Controlar a ocupação dos caixas através de *timers* individuais por caixa;
4. Determinar o tempo decorrido desde o início do atendimento do primeiro cliente até o final do atendimento do último cliente.

O programa deve mostrar, para cada cliente atendido:

- O caixa que deverá atender o cliente;
- O número de conta do cliente;
- A categoria de conta do cliente;
- O tempo que o cliente levará para ser atendido em sua quantidade de operações financeiras desejadas;
- O tempo que ele esperou até ser atendido.

No final, o programa deve mostrar o tempo total de atendimento, que é o tempo decorrido desde o início do atendimento do primeiro cliente até o final do atendimento do último cliente.

Considere que a unidade básica de tempo é o minuto. Ignore os tempos de enfileiramento (formação das filas), de chamada, e quaisquer outros tempos que não sejam os de atendimento pelo caixa. Considere que não haverá clientes com números de conta duplicados.

As partes abaixo devem ser implementadas e (possivelmente) utilizadas no programa. Note que o programa final (simulador de filas) pode até não necessitar de todas as funções especificadas abaixo. No entanto, todas elas devem ser definidas e implementadas, mesmo que não venham a ser utilizadas no programa.

Se a equipe achar necessário, poderá implementar outras estruturas de dados e funções para outros fins.

O programa deverá necessariamente ler um arquivo texto contendo os parâmetros de execução da simulação.

O padrão de formatação do arquivo de entrada é fixo e deve ser seguido à risca, assim como o padrão do arquivo de saída a ser produzido. O programa deve receber, como parâmetro na linha de comando, um número inteiro *X* qualquer, no intervalo [1;9999]. O programa deve, então, ler o arquivo de entrada chamado “entrada-X.txt”, onde X deve ter preenchimento de zeros à esquerda para ficar com 4 dígitos (Ex. 0005, 0043, 0523, 8423, etc.). O programa deve, então, colocar a saída da sua execução no arquivo de nome “saida-X.txt”, onde X deve ter preenchimento de zeros à esquerda para ficar com 4 dígitos, assim como exemplificado acima.

ATENÇÃO: O não atendimento a esses padrões implicará na atribuição de NOTA ZERO ao trabalho da equipe. Isto é, os casos de teste disponibilizados devem ter a saída de execução exatamente igual à saída esperada disponibilizada.

Serão disponibilizados 100 (cem) casos de teste, que são compostos de um arquivo de entrada e da saída esperada para a respectiva entrada. Não deverá haver diferença de nenhum caractere dos arquivos de saída produzido e esperado.

DICA: utilize a ferramenta “diff” do Linux para comparar os arquivos de saída:

obtido e esperado. O comando diff, no caso, deve retornar que não há diferença entre os arquivos.

Parte I: Estrutura de Dados Fila FIFO

Deverá ser desenvolvida uma estrutura de dados utilizando ponteiros e alocação dinâmica de memória (nos mesmos moldes das estruturas de dados desenvolvidas em laboratório) para uma fila com disciplina FIFO (*First In, First Out*), onde um elemento que chega primeiro na fila é também o primeiro elemento a ser retirado. Esta estrutura de dados deve ser desenvolvida nos arquivos fila_fifo.c e fila_fifo.h. Esta fila terá como informação a ser armazenada um valor inteiro. A chave de indexação desta informação também é uma informação inteira. Não deverá haver chaves duplicadas em uma fila. As funções dessa API não devem realizar nenhuma saída de informação na tela (printf). As seguintes funções devem ser implementadas:

void f_inicializar (Fila_FIFO **f);
Inicializa a fila.

int f_inserir (Fila_FIFO **f, int chave, int valor);
Insere um determinado valor inteiro indexado por um valor de chave na fila. Retorna 1 se a inserção for bem sucedida e 0 se houver algum problema (duplicação de chave ou falta de memória).

int f_obter_proxima_chave (Fila_FIFO **f);
Retorna o número de chave do próximo elemento da fila, retirando-o da fila. Retorna -1 se a fila estiver vazia.

int f_consultar_proxima_chave (Fila_FIFO **f);
Retorna a chave do elemento que está na cabeça da fila, sem retirá-lo da fila.

int f_consultar_proximo_valor (Fila_FIFO **f);
Retorna o valor armazenado no elemento que está na cabeça da fila, sem retirá-lo da fila.

int f_num_elementos (Fila_FIFO **f);
Retorna o número de elementos presentes na fila.

int f_consultar_chave_por_posicao (Fila_FIFO **f, int posicao);
Retorna a chave do posicao-ésimo elemento da fila. Caso não exista a posição desejada, retornar -1. A posição se inicia em 1.

int f_consultar_valor_por_posicao (Fila_FIFO **f, int posicao);
Retorna o valor do posicao-ésimo elemento da fila. Caso não exista a posição desejada, retornar -1. A posição se inicia em 1.

Parte II: Registrador

Desenvolver um registrador dos tempos de espera dos clientes, tendo por trás uma estrutura de dados de Árvore Binária de Busca (ABB). Os arquivos a serem gerados são “logtree.c” e “logtree.h”. As funções que deverão ser implementadas são:

void log_inicializar(Log **l);
Inicializa o registrador l.

void log_registrar(Log **l, int conta, int classe, int timer, int caixa);

Inclui um registro de tempo de atendimento da conta de número “conta”, que faz parte da classe de número “classe”, que esperou “timer” minutos para ser atendido pelo caixa de número “caixa”.

float log_media_por_classe(Log **l, int classe);

Retorna o tempo médio de espera, em minutos, para a classe de número “classe”.

int log_obter_soma_por_classe(Log **l, int classe);

Retorna a soma dos tempos de espera de todos os clientes cujas contas fazem parte da classe de número “classe”.

int log_obter_contagem_por_classe(Log **l, int classe);

Retorna a quantidade de clientes atendidos cujas contas são da categoria de número “classe”.

Parte III: Escalonador

Desenvolver um escalonador de filas que trata de forma diferenciada 5 filas, com as características de disciplina citadas no enunciado no trabalho. Os arquivos no projeto para o escalonador deverão ser: “escalonador.c” e “escalonador.h”.

O escalonador deve ter as seguintes funções:

void e_inicializar (Escalonador *e, int caixas, int delta_t, int n_1, int n_2, int n_3, int n_4, int n_5);

Inicializa o escalonador, alocando e inicializando as 5 filas, que serão atendidas por “caixas” caixas, onde cada operação é tratada em “delta_t” minutos por um caixa, e o escalonamento utiliza a Disciplina de Atendimento representada por {n_1; n_2; n_2; n_4; n_5}, nos termos do que foi definido na página 1 deste enunciado.

int e_inserir_por_fila (Escalonador *e, int classe, int num_conta, int qtde_operacoes);

Insere na fila “classe” o cliente de número “num_conta”, que pretende realizar “qtde_operacoes” operações bancárias.

int e_obter_prox_num_conta(Escalonador *e);

Retorna o número da conta do próximo cliente a ser atendido de acordo com a Disciplina de Atendimento, retirando-o da sua respectiva fila.

int e_consultar_prox_num_conta (Escalonador *e);

Retorna o número da conta do próximo cliente a ser atendido de acordo com a Disciplina de Atendimento, sem retirá-lo da sua respectiva fila.

int e_consultar_prox_qtde_oper (Escalonador *e);

Retorna a quantidade de operações bancárias que o próximo cliente das filas pretende realizar com o caixa, sem retirá-lo da sua respectiva fila.

int e_consultar_prox_fila (Escalonador *e);

Retorna a próxima fila que será atendida de acordo com a Disciplina de Atendimento.

int e_consultar_qtde_clientes (Escalonador *e);

Retorna a quantidade total (soma) de clientes esperando atendimento em todas as filas.

int e_consultar_tempo_prox_cliente (Escalonador *e);

Retorna o tempo necessário para que o próximo cliente a ser atendido realize todas as operações financeiras que deseja, sem retirá-lo da sua respectiva fila. Retornar -1 caso não tenha nenhum cliente em todas as filas.

int e_conf_por_arquivo (Escalonador *e, char *nome_arq_conf);

Realiza a configuração de inicialização do escalonador através da leitura do arquivo de configuração de nome "nome_arq_conf", retornando 1 em caso de sucesso e 0 caso contrário.

void e_rodar (Escalonador *e, char *nome_arq_in, char *nome_arq_out);

Executar a simulação do atendimento, lendo o arquivo de configuração de nome "nome_arq_in" e escrevendo o resultado do processamento para arquivo de nome "nome_arq_out".

Critérios de Avaliação

A avaliação será realizada em duas fases:

1. Análise do código-fonte;
2. Análise do atendimento dos requisitos do enunciado;
3. Análise funcional automatizada da execução do programa (teste).

O código-fonte será avaliado de acordo com os seguintes critérios qualitativos:

i. Eficácia do programa em suprir todos os requisitos;
ii. Eficiência do programa (otimização);
iii. Organização do código (uso racional de subprogramas, estruturas, etc.);
iv. Legibilidade do código (uso de endentação e semântica dos identificadores de variáveis);
v. Documentação (comentários dentro do código fonte).

Obviamente, funcionalidades adicionais às que foram solicitadas neste documento são bem vindas e serão gratificadas na nota (na medida do possível). O código-fonte deve conter, em comentário no início, os nomes e matrículas dos alunos que compõem o grupo. Os códigos-fonte devem ser submetidos na data fixada através da Google Classroom. **NÃO ENTREGAR NENHUM ARQUIVO COMPILADO (executável ou .o).**

Lembramos que todos os programas serão submetidos a análise léxica automática, que pode evidenciar cópia de código. Caso seja identificada "similaridade" entre códigos distintos, mesmo que parcial, as equipes envolvidas serão penalizadas.

Os trabalhos serão corrigidos no **Linux**. Portanto, certifique-se que o trabalho feito no Windows também compila e roda no Linux.

Prazo de Entrega: 9 de dezembro de 2022, até 23:55.

Upload através do Google Classroom

Não serão aceitas entregas por e-mail.