

Simulação de Atendimento Bancário

Índice

1. Introdução
2. Descrição do Cenário de Simulação
3. Lógica da Implementação
 - 3.1. Inicialização
 - 3.2. Simulação de Eventos
 - 3.3. Processamento de Clientes
 - 3.4. Fim do Expediente
4. Algoritmo de Simulação
 - 4.1. Terminou o Expediente?
 - 4.2. Chegou um Cliente?
 - 4.3. Cliente Entra na Fila
 - 4.4. Inicia a Transação
 - 4.5. Finalização do Expediente
5. Código Fonte
 - 5.1. Inclusão de Bibliotecas
 - 5.2. Definição de Constantes
 - 5.3. Estrutura de Dados para Clientes e para Fila
 - 5.4. Inicialização da Fila e Verificação se a Fila está Vazia
 - 5.5. Inserção na Fila
 - 5.6. Remoção da Fila
 - 5.7. Função para Limpar a Fila
 - 5.8. Função Principal
 - 5.9. Loop Principal
 - 5.10. Atendimento do Cliente no Guichê
 - 5.11. Atualização do Tempo Disponível nos Guichês
 - 5.12. Atendimento dos Clientes Restantes Após o Expediente
 - 5.13. Estatísticas Finais
6. Relatório Técnico
 - 6.1. Objetivo
 - 6.2. Metodologia
 - 6.3. Resultados
 - 6.4. Discussão
 - 6.5. Conclusão
7. Bibliografia

1. Introdução

A simulação de atendimento bancário é uma aplicação prática de Estruturas de Dados e Algoritmos, usada para modelar e analisar o desempenho de sistemas de atendimento ao cliente. Esta atividade utiliza a linguagem de programação C para simular o tempo médio de espera de clientes em uma agência bancária com três guichês de atendimento. A simulação oferece uma oportunidade de entender como filas e tempos de serviço podem afetar a experiência do cliente e a eficiência operacional de uma agência bancária.

2. Descrição do Cenário de Simulação

O cenário abrange uma agência bancária com três guichês e uma fila única de clientes. A cada segundo, um cliente pode chegar à fila com uma probabilidade determinada por um número aleatório. Cada guichê realiza uma transação de duração variável, sendo esta transação escolhida aleatoriamente entre saque, depósito ou pagamento. As durações médias dessas transações são 60 segundos para saques, 90 segundos para depósitos e 120 segundos para pagamentos.

2.1 Entidades Envolvidas

- Guichês: Responsáveis pelo atendimento dos clientes, têm estados de ocupado ou livre e tempos de ocupação variáveis conforme a transação realizada.
- Clientes: Entram na fila e aguardam atendimento. Cada cliente tem o horário de chegada registrado para calcular o tempo de espera.

3. Lógica da Implementação

A lógica de implementação é dividida em quatro partes principais: inicialização, simulação de eventos, processamento de clientes e fim do expediente.

3.1 Inicialização

Na fase de inicialização, o cronômetro é configurado para começar em zero, os guichês são marcados como livres e a fila de clientes é criada vazia. Esta configuração garante que a simulação comece em um estado limpo e controlado.

3.2 Simulação de Eventos

Durante a simulação, a cada segundo, os seguintes passos são executados:

1. **Chegada de Clientes:** Verifica-se se um novo cliente chega à fila, usando um gerador de números aleatórios.
2. **Atualização dos Guichês:** Verifica-se o estado dos guichês, liberando os que concluíram suas transações.
3. **Movimentação de Clientes:** Clientes são movidos da fila para guichês livres, e o tempo de espera é registrado.

3.3 Processamento de Clientes

Para cada cliente que chega, registramos o horário de chegada. Quando um guichê é liberado, o cliente no início da fila é processado, calculamos o tempo de espera e atualizamos o tempo total de espera.

3.4 Fim do Expediente

Quando o expediente termina (após 21600 segundos), todos os clientes restantes na fila continuam sendo atendidos até que não haja mais clientes na fila. Após todos serem atendidos, calculamos e imprimimos as estatísticas finais, como o tempo médio de espera, número total de clientes atendidos, tipos de transações realizadas e tempo extra de expediente necessário.

4. Algoritmo de Simulação

A simulação é baseada em eventos que ocorrem a cada segundo, com foco na chegada de clientes e na liberação de guichês. A seguir, detalhamos os componentes e a lógica do algoritmo.

4.1 Terminou o Expediente?

O término do expediente é controlado pelo cronômetro, que marca o tempo em segundos. O período de atendimento da agência é de 6 horas, equivalentes a 21600 segundos. Após esse período, o expediente é considerado encerrado.

4.2 Chegou um Cliente?

A cada segundo, um valor aleatório entre 0 e 29 é gerado. Se o valor for 0, indica que um cliente chegou à agência e entrou na fila. Caso contrário, nenhum cliente chega naquele segundo.

4.3 Cliente Entra na Fila

Ao chegar, um cliente é representado pelo horário de chegada, que é o valor corrente do cronômetro. Esse valor é inserido na fila de clientes, registrando assim o momento de entrada.

4.4 Inicia a Transação

Quando um guichê fica livre, ele atende o próximo cliente da fila. O tipo de transação que será realizada é determinado por um número aleatório entre 0 e 2:

- Saque (0): O tempo de ocupação do guichê é de 60 segundos.
- Depósito (1): O tempo de ocupação do guichê é de 90 segundos.
- Pagamento (2): O tempo de ocupação do guichê é de 120 segundos.

O cliente é removido da fila e o tempo de espera é calculado com base no horário de chegada e o horário atual do cronômetro.

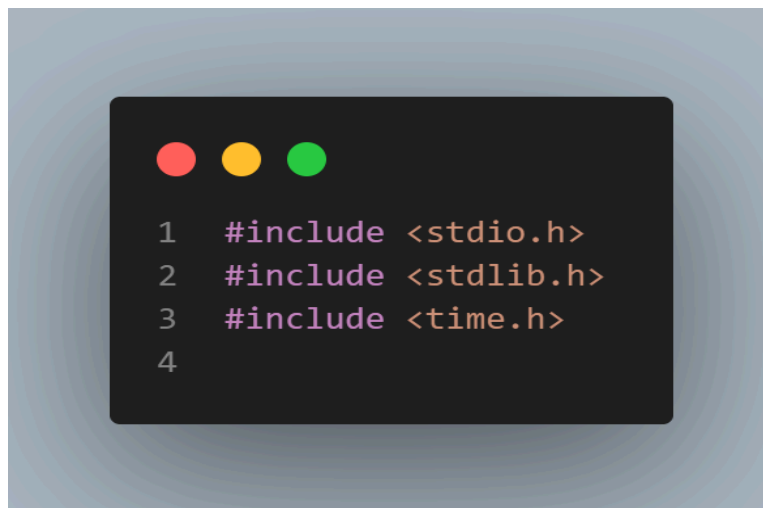
4.5 Finalização do Expediente

Após o término do expediente (21600 segundos), todos os clientes ainda na fila continuam sendo atendidos até que a fila esteja vazia. As seguintes informações são impressas:

1. Número total de clientes atendidos.
2. Número de clientes que fizeram saque, depósito e pagamento.
3. Tempo médio de espera na fila.
4. Tempo extra de expediente necessário para atender todos os clientes.

5. Código Fonte em C

5.1 Inclusão de Bibliotecas



- **#include <stdio.h>**: Inclui a biblioteca padrão de entrada e saída, necessária para usar funções como 'printf'.
- **#include <stdlib.h>**: Inclui a biblioteca padrão de funções utilitárias, como alocação de memória (malloc) e geração de números aleatórios ('rand').

- **#include <time.h>**: Inclui a biblioteca para manipulação de tempo, utilizada para inicializar a semente do gerador de números aleatórios ('srand').

5.2 Definição de Constantes

```
1 // Definições para tipos de operações
2 #define SAQUE 0
3 #define DEPOSITO 1
4 #define PAGAMENTO 2
5
6 // Tempos de duração das operações em segundos
7 #define TEMPO_SAQUE 60
8 #define TEMPO_DEPOSITO 90
9 #define TEMPO_PAGAMENTO 120
10
11 #define EXPEDIENTE 21600
12
```

- **#define SAQUE 0, #define DEPOSITO 1, #define PAGAMENTO 2**: Definem constantes para representar os tipos de transações.
- **#define TEMPO_SAQUE 60, #define TEMPO_DEPOSITO 90, #define TEMPO_PAGAMENTO 120**: Definem os tempos de cada tipo de transação em segundos.
- **#define EXPEDIENTE 21600**: Define a duração do expediente em segundos (6 horas).

5.3 Estrutura de Dados para Clientes, Guichês e Fila

```

1  typedef struct Cliente {
2      int tempo_chegada;
3      int tipo_operacao; // Tipo de operação: SAQUE, DEPOSITO ou PAGAMENTO
4      int tempo_atendimento;
5      struct Cliente* proximo;
6  } Cliente;
7
8  // Estrutura dos guichês
9  typedef struct Guiche {
10     int tempo_disponivel; // Tempo restante de atendimento no guichê
11     int atendendo_cliente;
12 } Guiche;
13
14 // Estrutura para representar uma fila de clientes
15 typedef struct Fila {
16     Cliente* inicio;
17     Cliente* fim;
18     int tamanho;
19 } Fila;

```

- **typedef struct Cliente:** Define um novo tipo de dado **Cliente** que representa um cliente.
- **int tempo_chegada:** Armazena o tempo de chegada do cliente.
- **int tipo_operacao:** Armazena o tipo de operação do cliente (**SAQUE, DEPOSITO, PAGAMENTO**).
- **int tempo_atendimento:** Armazena o tempo necessário para o atendimento do cliente
- **struct Cliente* proximo:** Ponteiro para o próximo cliente na fila, criando uma lista encadeada.
- **typedef struct Guiche:** Define um novo tipo de dado **Guiche** que representa um guichê de atendimento.
- **int tempo_disponivel:** Armazena o tempo restante de atendimento no guichê.
- **int atendendo_cliente:** Indica se o guichê está ocupado (1) ou livre (0).

- **typedef struct Fila:** Define um novo tipo de dado **Fila** que representa uma fila de clientes.
- **Cliente* inicio:** Ponteiro para o primeiro cliente da fila.
- **Cliente* fim:** Ponteiro para o último cliente da fila.
- **int tamanho:** Número de clientes na fila.

5.4 Inicialização da Fila e Verificação se a Fila está Vazia

```

1 void inicializaFila(Fila* fila) {
2     fila->inicio = NULL;
3     fila->fim = NULL;
4     fila->tamanho = 0;
5 }
6
7 int filaVazia(Fila* fila) {
8     return fila->tamanho == 0;
9 }
10

```

- **void inicializaFila(Fila* fila):** Função para inicializar a fila, definindo os ponteiros **inicio** e **fim** como **NULL** e o tamanho como **0**.
- **int filaVazia(Fila* fila):** Função que retorna **1** (true) se a fila está vazia e **0** (false) caso contrário, verificando se o tamanho é **0**.

5.5 Inserção na Fila

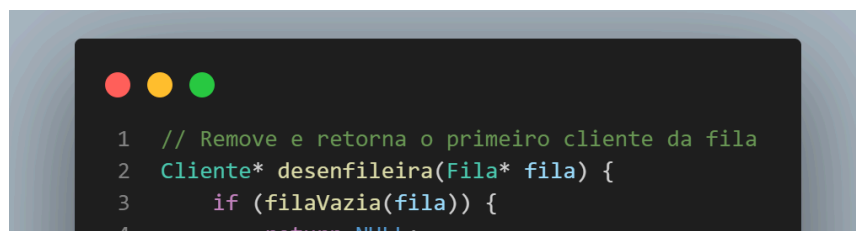
```

1 // Adiciona um novo cliente na fila
2 void enfileira(Fila* fila, int tempo_chegada, int tipo_operacao) {
3     Cliente* novo = (Cliente*)malloc(sizeof(Cliente));

```



- **void enfileira(Fila* fila, int tempo_chegada):** Função para inserir um cliente na fila.
- **Cliente* novo = (Cliente*)malloc(sizeof(Cliente)):** Aloca memória para um novo cliente.
- **novo->tempo_chegada = tempo_chegada:** Define o tempo de chegada do novo cliente.
- **novo->tipo_operacao = tipo_operacao:** Define o tipo de operação do cliente.
- **novo->tempo_atendimento = 0:** Inicializa o tempo de atendimento do cliente como **0**.
- **novo->proximo = NULL:** Define o ponteiro para o próximo cliente como **NULL**.
- **if (filaVazia(fila)):** Se a fila estiver vazia, define o novo cliente como **inicio** e **fim** da fila.
- **else:** Caso contrário, adiciona o novo cliente no final da fila e atualiza o ponteiro **fim**.
- **fila->fim = novo:** Atualiza o ponteiro **fim** para apontar para o novo cliente.
- **fila->tamanho++:** Incrementa o tamanho da fila.

5.6 Remoção na Fila



- **int desenfileira(Fila* fila):** Função para remover um cliente da fila e retornar seu tempo de chegada.
- **if (filaVazia(fila)):** Se a fila estiver vazia, retorna -1.
- **Cliente* temp = fila->inicio:** Armazena o ponteiro para o cliente no início da fila.
- **fila->inicio = fila->inicio->proximo:** Move o ponteiro **inicio** para o próximo cliente.
- **if (fila->inicio == NULL) {:** Se a fila ficou vazia, atualiza o ponteiro **fim** para **NULL**.
- **fila->tamanho--:** Decrementa o tamanho da fila.
- **return temp:** Retorna o tempo de chegada do cliente removido.

5.7 Função para Limpar a Fila



```
1 // Limpa toda a fila
2 void limpaFila(Fila* fila) {
3     while (!filaVazia(fila)) {
```

- **while (!filaVazia(fila))**: Continua enquanto a fila não estiver vazia.
- **Cliente* temp = fila->inicio**: Armazena o ponteiro para o cliente no início da fila.
- **fila->inicio = fila->inicio->proximo**: Move o ponteiro **inicio** para o próximo cliente.
- **free(temp)**: Libera a memória do cliente removido.
- **fila->fim = NULL**: Define o ponteiro **fim** como **NULL**, indicando que a fila está vazia.
- **fila->tamanho = 0**: Redefine o tamanho da fila para **0**.

5.8 Função Principal

```

1  int main() {
2      srand(time(NULL)); // Inicializa a geração de números aleatórios
3      Fila fila;
4      inicializaFila(&fila); // Inicializa a fila de clientes vazia
5
6      Guiche guiches[3] = {{0, 0}, {0, 0}, {0, 0}}; // Inicializa os guichês com tempo disponível zero e sem cliente
7
8      int tempo;
9      int tempo_total_espera = 0;
10     int clientes_atendidos = 0;
11     int num_saques = 0, num_depositos = 0, num_pagamentos = 0;

```

- **int main()**: Função principal onde a simulação ocorre.

- **srand(time(NULL))**: Inicializa a semente para geração de números aleatórios.
- **Fila fila**: Declara a fila de clientes.
- **inicializaFila(&fila)**: Inicializa a fila de clientes.
- **int guiches[3] = {0, 0, 0}**: Declara e inicializa os guichês, indicando que todos estão inicialmente livres.
- **int tempo** : Declara uma variável tempo para controlar o tempo da simulação.
- **int tempo_total_espera = 0**: Inicializa a variável para acumular o tempo total de espera dos clientes.
- **int clientes_atendidos = 0**: Inicializa a variável para contar o número total de clientes atendidos.
- **int num_saques = 0, num_depositos = 0, num_pagamentos = 0**: Inicializa as variáveis para contar o número de cada tipo de transação.

5.9 Loop Principal de Simulação

```
1 // Loop que simula o expediente de atendimento
2 for (tempo = 0; tempo < EXPEDIENTE; tempo++) {
3     if (rand() % 30 == 0) {
4         int tipo_operacao = rand() % 3;
5         enqueue(&fila, tempo, tipo_operacao);
6     }
```

- **for (int tempo = 0; tempo < EXPEDIENTE; tempo++)**: Loop que simula cada segundo do expediente.
- **if (rand() % 30 == 0)**: Se o número aleatório entre 0 e 29 for 0, um cliente chega à fila.
- **int tipo_operacao = rand() % 3** : Escolhe um tipo de operação aleatoriamente (0, 1 ou 2).
- **enfileira(&fila, tempo)**: Insere o cliente na fila com o tempo de chegada atual

5.10 Atendimento dos Clientes nos Guichês

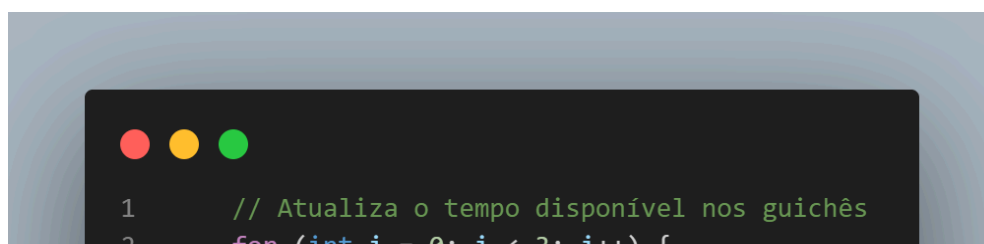
```

1  // Atendimento dos clientes nos guichês
2  for (int i = 0; i < 3; i++) {
3      if (guiches[i].tempo_disponivel == 0 && !filaVazia(&fila) && !guiches[i].atendendo_cliente) {
4          guiches[i].atendendo_cliente = 1;
5          Cliente* cliente = desenfileira(&fila);
6          int tipo_transacao = cliente->tipo_operacao;
7
8          // Inicia o atendimento de acordo com o tipo de operação
9          switch (tipo_transacao) {
10             case SAQUE:
11                 guiches[i].tempo_disponivel = TEMPO_SAQUE;
12                 num_saques++;
13                 break;
14             case DEPOSITO:
15                 guiches[i].tempo_disponivel = TEMPO_DEPOSITO;
16                 num_depositos++;
17                 break;
18             case PAGAMENTO:
19                 guiches[i].tempo_disponivel = TEMPO_PAGAMENTO;
20                 num_pagamentos++;
21                 break;
22             }
23
24             // Calcula o tempo de espera do cliente na fila
25             tempo_total_espera += (tempo - cliente->tempo_chegada);
26             clientes_atendidos++;
27
28             guiches[i].atendendo_cliente = 0; // Marca o guichê como disponível
29             free(cliente); // Libera a memória do cliente
30         }
31     }

```

- **for (int i = 0; i < 3; i++)** : Loop para verificar os três guichês.
- **if (guiches[i].tempo_disponivel == 0 && !filaVazia(&fila) && !guiches[i].atendendo_cliente)** : Verifica se o guichê está disponível, a fila não está vazia e o guichê não está atendendo outro cliente.
- **guiches[i].atendendo_cliente = 1;** Marca o guichê como ocupado.
- **Cliente* cliente = desenfileira(&fila);** Remove o primeiro cliente da fila.
- **int tipo_transacao = cliente->tipo_operacao;** Obtém o tipo de operação do cliente.
- **switch (tipo_transacao)** : Define o tempo de atendimento do guichê de acordo com o tipo de operação:
 - **case SAQUE:** Tempo de atendimento de 60 segundos.
 - **case DEPOSITO:** : Tempo de atendimento de 90 segundos.
 - **Case PAGAMENTO:** Tempo de atendimento de 120 segundos.
- **tempo_total_espera += (tempo - cliente->tempo_chegada);** Calcula o tempo de espera do cliente e acumula.
- **clientes_atendidos++;** Incrementa o contador de clientes atendidos.
- **guiches[i].atendendo_cliente = 0;** Marca o guichê como disponível.
- **free(cliente);** Libera a memória alocada para o cliente.

5.11 Atualização do Tempo Disponível nos Guichês



- **for (int i = 0; i < 3; i++):** Loop para atualizar os três guichês.
- **if (guiches[i].tempo_disponivel > 0):** Se o guichê está atendendo um cliente, decrementa o tempo de atendimento restante.

5.12 Atendimento dos Clientes Restantes Após o Expediente

```

1 // Atendimento dos clientes restantes após o expediente
2 float tempo_extra = 0;
3 while (!filaVazia(&fila)) {
4     for (int i = 0; i < 3; i++) {
5         if (guiches[i].tempo_disponivel == 0 && !filaVazia(&fila) && !guiches[i].atendendo_cliente) {
6             guiches[i].atendendo_cliente = 1;
7             Cliente* cliente = desenfileira(&fila);
8             int tipo_transacao = cliente->tipo_operacao;
9
10            switch (tipo_transacao) {
11                case SAQUE:
12                    guiches[i].tempo_disponivel = TEMPO_SAQUE;
13                    num_saques++;
14                    break;
15                case DEPOSITO:
16                    guiches[i].tempo_disponivel = TEMPO_DEPOSITO;
17                    num_depositos++;
18                    break;
19                case PAGAMENTO:
20                    guiches[i].tempo_disponivel = TEMPO_PAGAMENTO;
21                    num_pagamentos++;
22                    break;
23            }
24
25            tempo_total_espera += (EXPEDIENTE + tempo_extra - cliente->tempo_chegada);
26            clientes_atendidos++;
27
28            guiches[i].atendendo_cliente = 0;
29            free(cliente);
30        }
31    }
32
33    for (int i = 0; i < 3; i++) {
34        if (guiches[i].tempo_disponivel > 0) {
35            guiches[i].tempo_disponivel--;
36        }
37    }
38 }

```

- **float tempo_extra = 0:** Inicializa uma variável para acumular o tempo extra após o expediente.
- **while (!filaVazia(&fila)):** Loop que continua enquanto a fila não está vazia.
- **for (int i = 0; i < 3; i++):** Verifica os três guichês para atender os clientes restantes.
- **if (guiches[i].tempo_disponivel == 0 && !filaVazia(&fila) && !guiches[i].atendendo_cliente):** Verifica se o guichê está disponível, a fila não está vazia e o guichê não está atendendo outro cliente.
- **guiches[i].atendendo_cliente = 1:** Marca o guichê como ocupado.
- **Cliente* cliente = desenfileira(&fila):** Remove o primeiro cliente da fila.
- **int tipo_transacao = cliente->tipo_operacao:** Obtém o tipo de operação do cliente.
- **switch (tipo_transacao):** Define o tempo de atendimento do guichê de acordo com o tipo de operação:
 - **case SAQUE:** Tempo de atendimento de 60 segundos.
 - **case DEPOSITO:** Tempo de atendimento de 90 segundos.
 - **case PAGAMENTO:** Tempo de atendimento de 120 segundos.
- **tempo_total_espera += (EXPEDIENTE + tempo_extra - cliente->tempo_chegada):** Calcula o tempo de espera do cliente (considerando o tempo extra) e acumula.
- **clientes_atendidos++:** Incrementa o contador de clientes atendidos.
- **guiches[i].atendendo_cliente = 0:** Marca o guichê como disponível.
- **free(cliente);:** Libera a memória alocada para o cliente.
- **for (int i = 0; i < 3; i++):** Atualiza os três guichês.

- **if (guiches[i].tempo_disponivel > 0):** Se o guichê está atendendo um cliente, decrementa o tempo de atendimento restante.
- **tempo_extra++:** Incrementa o tempo extra.

5.13 Estatísticas Finais

```

1 // Impressão das estatísticas
2 printf("\nEstatísticas:\n");
3 printf("Número total de clientes atendidos: %d\n", clientes_atendidos);
4 printf("Número de saques: %d\n", num_saques);
5 printf("Número de depósitos: %d\n", num_depositos);
6 printf("Número de pagamentos: %d\n", num_pagamentos);
7 printf("Tempo médio de espera na fila: %.2f minutos\n", minutos_espera, segundos_espera);
8 printf("Tempo extra de expediente: %.2f minutos\n", minutos_extra, segundos_extra);
9
10 limpaFila(&fila); // Limpa a fila antes de terminar o programa
11
12 return 0;
13 }
14 float tempo_medio_espera = (float)tempo_total_espera / clientes_atendidos;

```

Calcula o tempo médio de espera dos clientes.

- **printf("Tempo médio de espera: %.2f segundos\n", tempo_medio_espera):** Imprime o tempo médio de espera.
- **printf("Clientes atendidos: %d\n", clientes_atendidos):** Imprime o número total de clientes atendidos.
- **printf("Número de saques: %d\n", num_saques):** Imprime o número de saques realizados.
- **printf("Número de depósitos: %d\n", num_depositos):** Imprime o número de depósitos realizados.
- **printf("Número de pagamentos: %d\n", num_pagamentos):** Imprime o número de pagamentos realizados.
- **limpaFila(&fila):** Limpa a fila, liberando toda a memória alocada.
- **return 0:** Retorna 0, indicando que o programa terminou com sucesso.

6. Relatório Técnico

6.1 Objetivo

Tivemos como objetivo implementar uma simulação de atendimento bancário, no qual podemos determinar o tempo médio que um cliente permanece na fila de uma agência e quantos clientes são atendidos pelo banco, utilizando estruturas de dados e algoritmos adequados em C. Seguimos as informações e atribuições fornecidas pelo enunciado da atividade.

6.2 Metodologia

A simulação foi realizada em um programa que representa uma fila para os clientes, e três guichês de atendimento. A cada segundo passado, eventos são processados para simular se a chegada de clientes acontecerá ou não. Mais um evento ocorrerá para decidir qual operação será realizada pelo cliente (saque, depósito ou pagamento) onde a cada uma delas, uma quantidade de tempo será atribuída, após isso, os guichês serão liberados. Finalmente, após o tempo do expediente ser cumprido, não chegará mais clientes e o tempo restante onde o processo ocorrerá até não haver mais clientes será contabilizado como tempo extra. Todos os eventos processados utilizaram de um gerador de números aleatórios para suas determinações.

6.3 Resultados

- **Número total de clientes atendidos:** Informado ao final da simulação.
- **Número de clientes que fizeram saque, depósito e pagamento:** Informado ao final da simulação.
- **Tempo médio de espera na fila:** Calculado e informado ao final da simulação.
- **Tempo extra (pós expediente):** Calculado e informado ao final da simulação.

6.4 Discussão

Os resultados obtidos refletem a eficiência do sistema de atendimento da agência, mostrando como a fila e os guichês operam suas atividades.

Guichês interagem ao longo do tempo. O uso de uma fila única para os três guichês e a aleatoriedade na chegada de clientes e escolhas de transações são fatores determinantes na variação do tempo médio de espera.

Podemos levantar alguns pontos relevantes:

1. A simulação utiliza estruturas de dados como filas para gerenciar clientes e guichês.
2. O uso de geradores aleatórios para simular a chegada de clientes e tipos de transações adiciona realismo à simulação.
3. A análise dos tempos de espera e tipos de transações pode fornecer insights para otimizar o atendimento em agências bancárias.
4. A implementação em linguagem C destaca a importância de uma programação eficiente e bem estruturada para simulações complexas.
5. A simulação permite observar o impacto de diferentes cenários de atendimento e carga de trabalho em uma agência bancária.

6.5 Conclusão

A simulação de atendimento bancário com três guichês e uma fila única de clientes fornece uma visão detalhada sobre a dinâmica de filas e tempos de serviço em uma agência bancária.

Usando a linguagem de programação C, a simulação calcula o tempo médio de espera dos clientes, a quantidade de transações realizadas e outros parâmetros relevantes, permitindo a análise da eficiência do atendimento e a identificação de possíveis melhorias no processo.

Ao decorrer da criação do sistema estivemos em face a algumas imprecisões e dificuldades em nosso algoritmo, que foram resolvidas após sequenciais manutenções. Foram essas: tempo médio de espera dos clientes muito abaixo do esperado; quantidade de clientes atendidos contradizente ao tempo decorrido; ligeira dificuldade na manipulação da estrutura da fila; tempo de expediente não condizente com o funcionamento do código;

7. Bibliografia

- CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford.

****Algoritmos: Teoria e Prática.**** 3. ed. Rio de Janeiro: Elsevier, 2012.

- KNUTH, Donald E. ****The Art of Computer Programming, Volume 1: Fundamental Algorithms.**** 3. ed. Addison-Wesley Professional, 1997.

- SEDGEWICK, Robert; WAYNE, Kevin. ****Algorithms.**** 4. ed. Addison-Wesley Professional, 2011.

- DEITEL, Harvey M.; DEITEL, Paul J. C: Como Programar. 6ª ed. São Paulo: Pearson Prentice Hall, 2010.