Universidade Federal do Rio de Janeiro
Ciência da Computação

Avaliação e Desempenho 2011/1

Trabalho de Simulação

Luiz Filipe de Sá Estrella **DRE:** 107390627
Fernando de Mesentier Silva **DRE:** 107390520
Rodrigo de Moura Canaan **DRE:** 107362200
Vinicius José Serva Pereira **DRE:** 106050355

RIO DE JANEIRO, JUNHO 2011

_____

Luiz Filipe de Sá Estrella

_____

Fernando de Mesentier Silva

_____

Rodrigo de Moura Canaan

_____

Vinicius José Serva Pereira

Atestamos que todos os integrantes do grupo participaram na elaboração do trabalho como um todo. Realizando as seguintes tarefas:

- Modelagem do simulador;
- Codificação do simulador;
- Elaboração do programa de otimização;
- Geração de gráficos para análise;
- Elaboração do relatório;
- Documentar e comentar os códigos.

Sumário

Anexos

1 Introdução

1.1 Funcionamento geral do simulador

O simulador projetado para o trabalho recebe seus parâmetros através de um arquivo texto, com a extensão *ini*, contendo o valor de cada entrada desejada. A entrada de parâmetros foi desenvolvida dessa forma para que fosse possível criar e guardar diversos cenários diferentes ao mesmo tempo.

Ao iniciar o simulador, deve-se entrar com o nome do arquivo contendo as entradas. São reconhecidos os seguintes parâmetros:

- **taxa_chegada:** Representa a taxa de chegada usada no sistema. Deve ser um valor inteiro e dentro do intervalo]0,1[;
- **num_clientes:** Representa o número de clientes em cada rodada. Somente valores inteiros, sem outras restrições;
- **num_rodadas:** Define o número de rodadas desejadas na simulação. A única restrição é ser um valor inteiro;
- **semente:** Este parâmetro determina qual é a semente do gerador de números pseudoaleatórios. Ele não é obrigatório, caso não for informado nenhum valor para o parâmetro, seu valor será obtido aleatoriamente através do tempo do sistema.
- **debug:** Caso seja marcado como *true,* o simulador apresenta no console cada etapa do sistema, detalhadamente.
- **mostrar_resultados:** Caso seja marcado como *true*, é apresentado no console o resultado dos cálculos de esperança de cada rodada.
- **guardar_estatisticas:** Se o usuário desejar guardar todos os dados estatísticos de cada rodada executada, este parâmetro deve ser marcado como *true*. Será gerada uma pasta com o nome do arquivo carregado para inicializar os valores dos parâmetros. Nesta pasta, o usuário irá encontrar um arquivo texto para cada esperança que queremos calcular, cada um contendo todas as médias de cada rodada executada. Existirá também um arquivo contendo todas as entradas utilizadas na simulação.
- **determina_transiente:** Esta booleana determina se o usuário deseja executar a simulação no intuito de determinar o fim da fase transiente. Neste modo a

coloração de cada cliente não é levada em consideração e todos os cálculos das esperanças são em relação ao tempo total e o número total de clientes servidos em toda a simulação. Este método foi utilizado na obtenção do fim da fase transiente usada no trabalho, descrita em detalhes na sessão 3.

- **tamanho_transiente:** Representa o número de clientes que não devem ser coletados na geração dos dados estatísticos.

Os seguintes parâmetros são usados para verificar a correção do simulador, e são explicados em maiores detalhes na sessão 2:

- **deterministico:** Se este parâmetro for verdadeiro, as amostras de tempo exponencial não são geradas de forma aleatória, mas utilizando a média. Este parâmetro pode ser usado para um teste de correção próprio ou combinado com os parâmetros *dois_por_vez* e *interrupção_forcada* para rodar variações do modo de execução básico que testam certas partes do sistema de maneira mais efetiva.
- **dois_por_vez:** As chegadas ocorrem dois clientes por vez, com metade da taxa especificada pelo usuário. Esse parâmetro deve ser utilizado em conjunto com o *deterministico*. Esse modo de execução garante que, mesmo gerando os tempos de forma determinística, haverá variância entre os tempos de espera na fila 1, o que nos ajuda a testar as estruturas relacionadas a essa fila.
- **interrupcao_forcada:** As chegadas ocorrem de maneira que, a cada duas chegadas, o segundo cliente sempre interromperá o segundo serviço do primeiro cliente. Também deve ser usado em conjunto com o parâmetro *determinstico*. Esse modo de execução nos ajuda a testar se as interrupções estão sendo tratadas corretamente na fila 2, e gera variância entre os tempos de espera na fila 2, mesmo rodando de maneira determinística.

Todas as entradas do tipo booleano são opcionais. Caso o usuário não informe alguma dessas variáveis elas recebem, automaticamente, o valor *false*.

Quando carregamos o simulador com os parâmetros desejados ele se inicia. Foram utilizados três tipos diferentes de eventos na criação do simulador, mas um deles é utilizado somente para um dos casos de teste de correção. Podemos explicar a implementação destes eventos, resumidamente, da seguinte forma:

- **Chegada de um novo cliente:** Um novo cliente é gerado e chega ao sistema. Ele é inserido na fila 1. Caso o servidor esteja ocupado com alguém da fila 2 o evento

termino de serviço é removido da fila de eventos e o cliente em serviço é alocado de volta no início da fila 2. O tempo de serviço restante deste cliente é guardado e será utilizado na próxima vez que ele entrar em serviço.

- **Término de serviço:** Na ocorrência de um evento deste tipo, é necessário primeiro saber a qual fila pertence o cliente em serviço. Se o cliente for da fila 1 ele é inserido no fim da fila 2. Se o cliente for da fila 2 ele já pode ser removido do sistema e contabilizamos como mais um cliente servido.

- **Chegada Artificial:** Nas variações do modo de execução padrão, ativadas usando os parâmetros *dois_por_vez* ou *interrupção_forcada*, há a necessidade de fazer com que cada chegada de um cliente gere uma chegada adicional. No caso *dois por vez*, a chegada adicional ocorrerá imediatamente, e no de interrupção forçada, o tempo de chegada será calculado de forma que a chegada extra ocorra na metade do segundo serviço do primeiro cliente.

  De qualquer forma, é necessário diferenciar uma chegada normal de uma chegada "extra", que é desencadeada por uma chegada normal quando rodamos o programa nessas variações. Para evitar que as chegadas extras também gerem um novo evento de chegada, o que faria com que a taxa de chegada crescesse arbitrariamente, foi criado um tipo de evento de chegada que indica ao programa que essa já é uma chegada extra, que não deve desencadear mais uma chegada adicional. Esse tipo de evento foi chamado *chegada artificial*.

Nosso simulador funciona em *loop* enquanto o número de clientes que se deseja servir não foi alcançado. Este número de clientes depende também de outros fatores como a rodada atual, mas isso será explicado melhor mais a frente.

Ao iniciar o simulador criamos um evento do tipo *chegada de um novo cliente* e o colocamos na lista de eventos. Após cada evento deste tipo que passa pelo *loop* do simulador criamos um novo evento do mesmo tipo. Depois de tratarmos os eventos existe um condicional que verifica se o servidor está vazio. Caso ele esteja vazio verificamos primeiro se existe alguém na fila 1, isso devido ao fato de que a fila 1 tem prioridade sobre a fila 2, se existir ele é inserido no servidor e seu evento do tipo *término de serviço* é gerado. Se não a fila 1 estiver vazia então alguém da fila 2 entra no servidor, isso se também não estiver vazia. Antes de gerar seu término de serviço é verificado se este cliente já foi interrompido alguma vez, caso tenha sido o seu tempo

de serviço não será gerado pelo nosso gerador, e sim é seu tempo de serviço restante guardado anteriormente.

Os dados usados para as estatísticas são coletados antes de cada nova chegada ao sistema e a cada término de serviço. Ao longo de uma rodada, os dados são acumulados em variáveis, no fim de cada rodada obtemos o valor médio de cada dado. Ao finalizar a execução da simulação são apresentados no console todos os intervalos de confiança calculados e os valores médios estimados.

1.2 Tipo de Linguagem

Utilizamos a linguagem de programação C++ para desenvolver o sistema. Além de ser a linguagem com que o grupo poderia ter mais facilidade e liberdade na programação queríamos que a compilação do simulador fosse simples e possível em praticamente todos os sistemas operacionais.

Para compilar o simulador basta ter um compilador de C++, no nosso caso utilizamos o MingW no Windows.

1.3 Estruturas internas

Na criação do simulador codificamos as seguintes classes:

- **Cliente:** Nesta classe armazenamos todas as informações de estado do cliente e seus dados usados nos cálculos estatísticos. Temos por exemplo a que fila o cliente pertence, seu instante de chegada ao sistema, a que rodada ele pertence e etc. Além disso, temos algumas variáveis auxiliares, como a variável booleana que determina se o cliente já foi interrompido ou não, entre outras.
- **Evento:** Esta classe é utilizada como o evento em si. Armazena seu tipo e seu tempo de acontecimento.
- **GeradorTempoExponencial:** Esta classe é responsável por gerar o tempo entre chegadas exponencialmente distribuído. Os números pseudoaleatórios usados na inversa da CDF da distribuição exponencial não são gerados nesta classe, e sim na classe *Mersenne*, a qual essa classe herda.

- **Simulador :** Classe principal do sistema. Nela temos todo o controle dos eventos e dos clientes. A coleta de dados para a estatística e a geração dos arquivos texto com os dados é feito nela.
- **Mersenne:** Esta classe é responsável pela geração dos números pseudoaleatórios utilizados na classe *GeradorTempoExponencial*. A explicação de seu funcionamento e o motivo de sua utilização são explicados no próximo tópico.
- **Main:** Responsável por inicializar o simulador com as entradas requisitadas. É feito também o calculo dos intervalos de confiança no fim da simulação.

1.4 Geração de variáveis aleatórias

Para realizar a simulação é necessário gerar o tempo de chegada dos clientes e tempos de serviço, que são variáveis aleatórias distribuídas exponencialmente. Para isso foi utilizada a inversa da CDF da distribuição exponencial. Após as devidas transformações, temos o seguinte resultado:

$$x = -\log(1 - y)/\lambda$$

Onde λ é a taxa da distribuição exponencial (que no programa pode ser 1, para serviços ou a entrada dada pelo usuário para chegadas), a variável *y* é um número pseudo-aleatório distribuído uniformemente, no intervalo [0,1[e x é o tempo gerado.

As bibliotecas base do C/C++ disponibilizam os métodos *srand* e *rand* como meios para geração de números pseudo-aleatórios. No entanto, esses meios geram valores dentro do intervalo [0, *RAND_MAX*], onde *RAND_MAX* é o maior inteiro possível de 4 bytes($2^{16} - 1$). Assim, como precisamos de valores dentro do intervalo [0, 1[, é necessário fazer a divisão pelo valor *RAND_MAX*. Porém, a partir de pesquisas sobre esses meios, descobrimos que ao executar essa divisão os números obtidos perdem sua distribuição uniforme, afetando assim as qualidades das variáveis aleatórias dependentes dos mesmos.

Devido às limitações explicadas dos métodos *rand* e *srand* era necessário outro gerador de números pseudo-aleatórios. Após pesquisar sobre alternativas para geração desses números, encontramos o algoritmo chamado de *Mersenne Twister*[1].

[1] A implementação utilizada no trabalho não foi criada pelo grupo, mas foram feitos diversos testes para confirmar sua funcionalidade. O código foi obtido em: http://www.agner.org/random/

O algoritmo de *Mersenne Twister* recebe este nome por usar como período um primo de *Mersenne*. Um primo de *Mersenne* é um primo que é potencia de 2 menos 1:

$$M_p = 2^p - 1$$

O código é baseado em uma matriz de recorrência linear sobre um campo binário finito. Este algoritmo foi desenvolvido tendo em mente *Simulações de Monte Carlo*[2] e outras tipos de simulação em mente.

1.5 Método para análise

O método utilizado para análise foi o Batch. Neste método, a partir de uma mesma semente, temos a execução de várias rodadas. A simulação não é reiniciada, assim, a independência entre medidas não é total. Na prática, se as rodadas forem grandes o suficiente, esta dependência entre as medidas não é um problema.

Um dos motivos da escolha do método Batch foi pela possibilidade de precisar estimar a fase transiente somente uma vez.

Todo o cliente gerado em uma determinada rodada é marcado como pertencente a ela e seus dados só são coletados se o cliente for pertencente à rodada atual com que o simulador está trabalhando. No fim de uma rodada podemos ter clientes da rodada anterior ainda no sistema, eles não são removidos do sistema, mas seus dados não são coletados para análise.

1.6 O fator mínimo

Para obtermos os fatores mínimos utilizamos somente uma máquina, com as seguintes especificações:

- Pentium Dual-Core E240 1.60 Ghz, rodando com *overclock* a 3.0 Ghz
- 3 Gb DDR2 de memória RAM
- OS Windows 7

O cálculo dos fatores mínimos, para cada valor de utilização proposto, levou em média 10 horas. Levando em conta que, para taxas pequenas como 0.2 o tempo foi de aproximadamente 2 horas, mas para a taxa de 0.45 levou 16 horas.

[2] São simulações que dependem repetidamente de amostras aleatórias para computar seus resultados.

2 Teste de Correção

Para verificar que as estatísticas estão sendo coletadas e calculadas corretamente, criamos alguns casos de teste determinísticos, onde podemos calcular facilmente o resultado teórico esperado e comparar com o resultado do compilador.

O método mais simples de simulação determinística é fazer com que todos os tempos exponenciais gerados sejam iguais às médias. Assim, para uma taxa de chegada de clientes λ, e serviço com taxa 1, todos os intervalos entre chegadas serão de tamanho 1/λ e todos os serviços acabarão um segundo após seu começo.

Nessas condições, supondo as condições de equilíbrio ($2\lambda = \rho < 1$), os resultados esperados são:

- **$E[W_1] = E[W_2] = 0$**, pois nunca nenhum cliente esperará na fila.
- **$E[T_1] = E[T_2] = 1$**, devido ao fato de que o único tempo que os clientes passam no sistema é enquanto estão sendo servidos, e o serviço sempre demora 1 segundo.
- **$V[W_1] = V[W_2] = 0$**, já que todos os tempos de espera em fila são de mesmo tamanho e igual à zero.
- **$E[N_{q1} = E[N_{q2}] = 0$**, pois as filas estão sempre vazias
- **$E[N_1 = E[N_2] = \lambda$**, pois a cada 1/λ segundos o servidor fica exatamente 1 segundo servindo um cliente pela primeira vez e 1 segundo servindo o mesmo cliente pela segunda vez. A utilização do servidor é então: $\rho_1 = \rho_2 = 1 / (1/\lambda) = \lambda$. Como **$E[N_1] = \rho 1 + E[N_{q1}]$** e **$E[N_{q1}] = 0$**, **$E[N_1]= \rho_1 = \lambda$**, e o mesmo vale para $E[N_2]$.

Apesar de o modo determinístico simples servir para testes básicos, ele tem dois problemas:

1. As estruturas internas do sistema nunca são realmente testadas. Nenhum cliente fica nas filas, e nenhum cliente da fila 2 é interrompido por um cliente da fila 1.;
2. Várias das grandezas que queremos estimar são nulas. Os tempos médios de espera, os tamanhos médios das filas e as variâncias dos tamanhos das filas.

Devido a isso, se houver um erro em nosso tratamento e um cliente puder tomar um caminho inesperado dentro do sistema, os dados desse teste não nos indicarão esse erro. Se houver um erro de cálculo das grandezas nulas, o erro também não será visto.

Para verificar a correção dessas grandezas, criamos outro modo de teste determinístico, que chamamos de modo *dois por vez*. Nesse modo, a cada chegada que o sistema fosse gerar, ele gera duas chegadas simultâneas. Novamente, todos os tempos são gerados deterministicamente, usando as médias.

Para compensar o fato de que cada chegada na verdade traz dois clientes, trabalhamos com metade da taxa de chegada fornecida pelo usuário, para manter a taxa de chegada efetiva que o usuário desejava testar.

Supondo novamente que a taxa λ fornecida pelo usuário seja tal que o sistema esteja em equilíbrio (λ < 0,5), o funcionamento do sistema se dará da seguinte forma:

- As chegadas ocorrem a cada (2/λ) segundos, duas chegadas por vez;
- Um dos clientes, não importa qual, terá seu evento de chegada tratado primeiro e será colocado no servidor. Esse cliente não esperará na fila. Seu companheiro irá esperar exatamente um segundo na fila, enquanto o primeiro é servido;
- Quando o primeiro cliente terminar seu primeiro serviço, ele irá para fila 2, onde esperará um segundo, enquanto o segundo cliente é servido pela primeira vez;
- Após o término do primeiro serviço do segundo cliente, o primeiro será servido pela primeira vez, ocupando o servidor por um segundo, enquanto o segundo cliente espera por um segundo na fila 2. Quando o primeiro cliente terminar seu serviço e sair do sistema, o segundo entra no servidor e é servido por um segundo.

Dessa forma, a cada dois clientes, um fica na fila 1 por um segundo, outro não espera na fila 1. Logo, obtemos os seguintes valores:

- **$E[W_1] = 1/2$**;
- **$E[T_1] = 3/2$**;

Para calcular $V[W_1]$, fazemos $V[W_1] = E[W_1^2] – E[W_1]^2$

- **$E[W_1^2] = 1/2$;**
- **$E[W_1]^2 = 1/4$;**
- **$V[W_1] = 1/4$.**

Todos os clientes esperam um segundo na fila 2 e um segundo no servidor. Em cada par, o que entrar em serviço primeiro irá esperar na fila 2 enquanto o outro é servido vindo da fila 1, e o segundo irá esperar na fila 2 enquanto o primeiro é servido na fila 2.

Assim, temos:

- **$E[W_2] = 1$;**
- **$E[T_2] = 2$;**
- **$V[W_2] = 0$.**

A cada 2/λ segundos, a fila 1 passa um segundo ocupada, e a fila 2, dois segundos. Assim, obtemos:

- **$E[N_{q1}] = 0{,}5\lambda$;**
- **$E[N_{q2}] = 2 / (2/\lambda) = \lambda$;**
- **$E[N_1] = 1{,}5\lambda$;**
- **$E[N_2] = 2\lambda$.**

Esse modo nos dá uma garantia maior de que o simulador funciona corretamente, mas ainda não testa as interrupções na fila 2, e a variância de W_2 ainda é nula. Assim, se houver um erro no tratamento do cliente interrompido, ou no cálculo da variância de W_2, esse teste não nos dará pista alguma.

Para testar essas funcionalidades que faltam, criamos um terceiro modo de teste determinístico, que chamamos de modo de interrupção forçada.

Como no modo dois por vez, cada chegada normal do sistema gerará uma chegada artificial adicional. Porém, em vez de agendar essa chegada para o momento atual, a agendamos para o tempo de 1 serviço e meio no futuro. Como cada chegada efetivamente gera duas chegadas, trabalhamos também com metade da taxa fornecida.

Dessa forma, supondo λ fornecido < 0,5, o sistema funciona da seguinte maneira:

A cada 2/λ segundos, chega um cliente. Esse cliente não terá que esperar na fila 1, e será servido pela primeira vez em um segundo. Ele entrará em seu segundo serviço, e meio segundo depois, será interrompido pela chegada artificial.

Esse cliente então esperará um segundo na fila 2 enquanto o cliente artificial é servido na fila 1. Note que o cliente artificial, assim como o também não precisa esperar na fila 1.

Quando o primeiro serviço do segundo cliente terminar, o primeiro retoma seu segundo serviço. Como o simulador salva o tempo restante no momento da interrupção, ele ficará apenas meio segundo no servidor, e sairá do sistema. Esse cliente teve $W_2 = 1$ e $T_2 = 2$.

O segundo cliente teve que esperar este último serviço de meio segundo, pois o cliente anterior (que ele interrompeu quando chegou ao sistema) chegou à fila 2 antes. Ele será servido em sequência, ficando mais um segundo no servidor. Seus tempos serão $W_2 = 0,5$ e $T_2 = 1,5$.

Como nenhum cliente perde tempo esperando na fila 1,temos:

- **$E[W_1] = 0$;**
- **$E[T_1] = 1$;**
- **$V[W_1] = 0$;**
- **$E[N_{q1}] = 0$;**
- **$E[N_1] = \lambda$.**

Para as médias teóricas das medidas da fila 2, temos:

- **$E[W_2] = 0,75$;**
- **$E[T_2] = 1,75$;**
- **$E[W_2^2] = 0,625$;**
- **$V[W_2] = 0,0625$.**

Para calcular $E[Nq_2]$ e $E[N_2]$, procedemos da seguinte forma:

A cada $2/\lambda$ segundos, um cliente fica na fila por um segundo, e um cliente fica por meio segundo. Então a utilização da fila é:

- **$E[N_{q2}] = (3/4) * \lambda$;**
- **$E[N_2] = (7/4) * \lambda$.**

No anexo I, vemos trechos do output de debug desses três modos, que mostram como as estruturas internas do cliente se comportam, e também resultados calculados em simulações com duas taxas diferentes, que refletem os resultados teóricos para cada modo de execução.

É interessante notar que, mesmo para simulações determinísticas, o resultado encontrado pelo simulador pode não ser exato, por motivos de erro de aproximação nas divisões feitas e porque o estado inicial do sistema (vazio) não é o estado típico do sistema, o que pode afetar os dados, em particular para rodadas pequenas.

3 Estimativa da fase transiente

Um problema que precisa ser resolvido na simulação é que, ao iniciarmos o simulador, o fazemos num estado ,ou seja, número de clientes em cada região interna, que não necessariamente reflete o estado típico do sistema.

Para melhorar a qualidade das simulação, é importante descartar alguns dos primeiros clientes que chegarem ao sistema, ou seja, não considerá-lo no cálculo para estimar as grandezas em que estamos interessados.

Para descartar os *T* primeiros resultados, basta atribuir o valor *T* ao parâmetro *tamanho_transiente*. Se esse valor for diferente de zero, o programa fará uma rodada de simulação extra no início, que não coleta dado algum. Assim, se a entrada for *tamanho_transiente* = *T*, *num_rodadas* = *N* e *num_clientes* = *C*, o simulador fará uma rodada de tamanho *T*, que representa a fase transiente e será descartada, seguida de *N* rodadas de tamanho *C*, que serão usadas para coletar as estatísticas.

O valor de *T* é escolhido da seguinte forma:

O parâmetro *determina_transiente* é usado para indicar que o simulador deve rodar em modo de estimação da fase transiente. Este modo funciona de forma parecida com o batch implementado, com a diferença de que a cada *n* clientes servidos, calculamos a média de todas as grandezas até ali, ou seja, dividimos os valores acumulados.

Usamos esse modo de execução para obter os valores médios das grandezas que queremos medir para um determinado número de clientes. Rodando o programa várias vezes nesse modo para um dado λ, mas com sementes diferentes, podemos comparar as medidas obtidas a cada número de clientes servidos e, comparando as medidas, estimar a partir de quantos clientes o sistema entra em equilíbrio, ou seja, a partir de que ponto os resultados obtidos com cada semente variam pouco entre si.

Uma vez determinado o valor *T* (que é diferente para cada λ escolhido), podemos passar *T* como entrada do programa, usando a variável *tamanho_transiente* para que o programa descarte os *T* primeiros serviços, o que deve nos dar resultados mais precisos.

Como rodamos o programa com várias sementes para determinar o valor de *T*, garantimos que o valor encontrado não é especifico para uma única semente, mas temos uma certa confiança de que é independente da semente escolhida.

Utilizando o método descrito, com 500 rodadas e 300 clientes por rodada, ou seja, 150 mil clientes, executamos 5 simulações, cada uma com semente diferente,

para cada taxa de chegada pedida. A partir dos dados obtidos na simulação, geramos gráficos para cada dado estatístico pedido.

Analisando os gráficos obtidos, podemos observar que os casos mais críticos são quando λ=0.4 e λ=0.45. A variável aleatória, que mais demora a se estabilizar, é a variância de W_2. Então, foi decidido, que o tamanho da fase transiente deveria ser determinado a partir da análise desta variável aleatória.

Foi decidido, que um bom tamanho para a fase transiente, quando λ=0.4 e λ=0.45 era de 60 mil clientes. Isso se deve ao fato de que, observando os gráficos gerados(figuras 1 e 2), dos valores de $V(W_2)$, pode se ver que a partir da rodada 200, ou seja, 60 mil clientes servidos, já temos uma estabilização dos valores, apesar de seu comportamento abrupto.

Para termos uma melhor garantia de que, o valor escolhido para o tamanho da fase transiente, era adequado, executamos uma simulação com λ=0.45, com 300 clientes por rodada e mil rodadas, para 10 sementes diferentes. Analisando o gráfico obtido (figura 3), pode-se ver que o comportamento da variável aleatório é como esperado. Podemos garantir então, que o tamanho escolhido para a fase transiente, para λ=0.4 e λ=0.45, é adequado.

Para os valores de λ, 0.1, 0.2 e 0.3, o tamanho da fase transiente escolhido foi de 20 mil clientes. Podemos observar que, a partir dos gráficos gerados (figuras 4,5 e 6) para o valor de $V(W_2)$, a partir de 20 mil clientes servidos já se pode observar a estabilidade da variável aleatória.

Utilizamos então, os seguintes valores para a fase transiente:

λ	**Tamanho da fase transiente**
0.1	20 mil
0.2	20 mil
0.3	20 mil
0.4	60 mil
0.45	60 mil

Além dos gráficos para $V(W_2)$, é interessante observar os obtidos para as outras variáveis aleatórias, principalmente para e λ=0.45. Incluímos no anexo II, os gráficos que consideramos importantes para as análises.

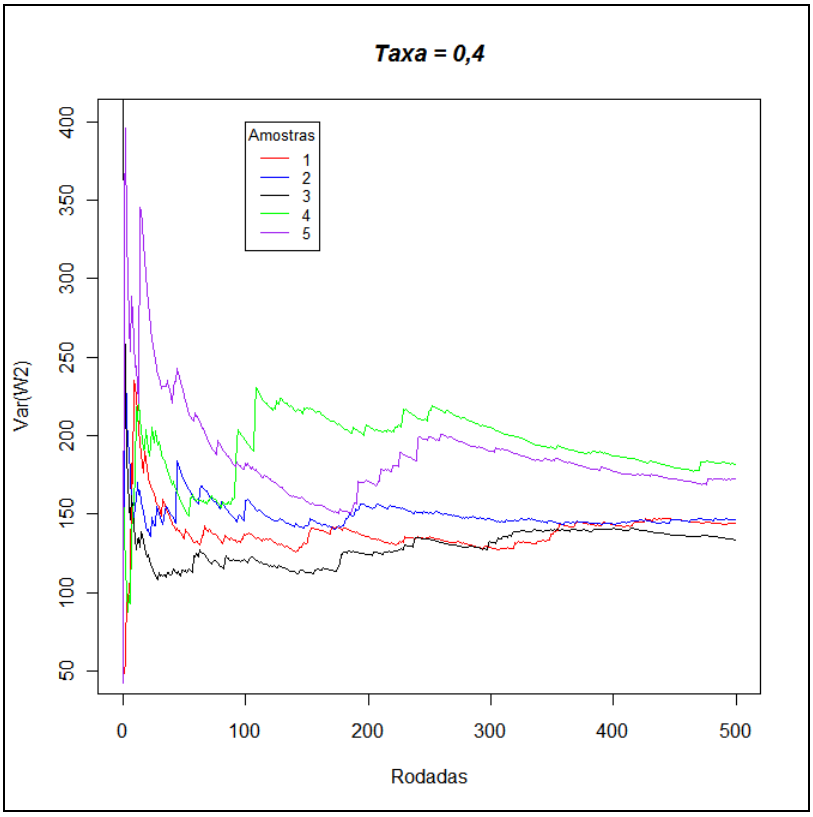

Figura 1: Variância de W2, com taxa = 0.4

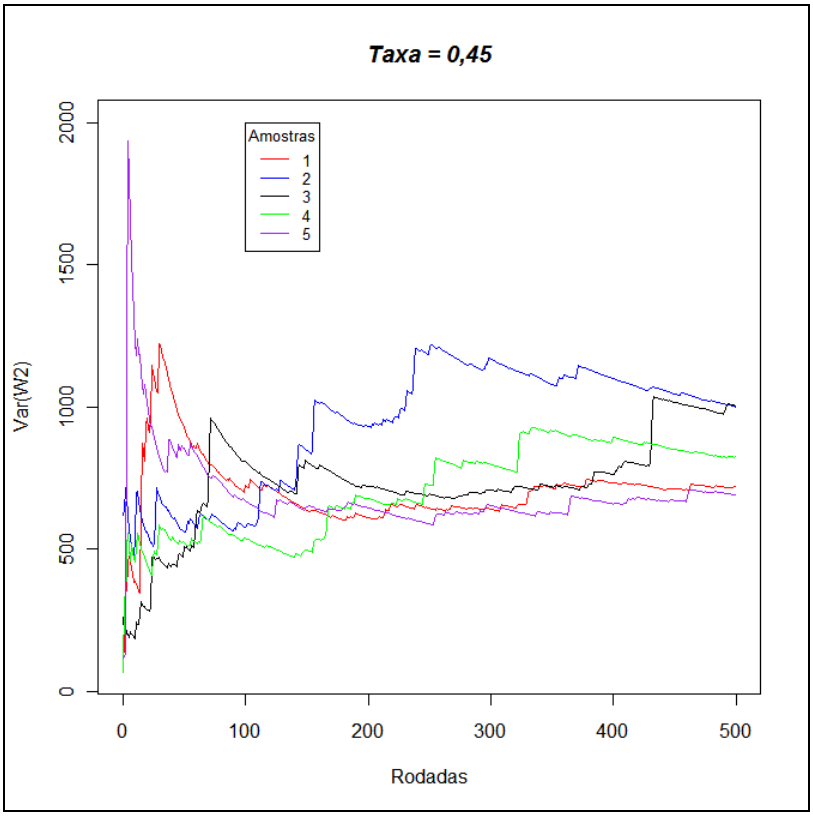

Figura 2:Variância de W2, com taxa = 0.45

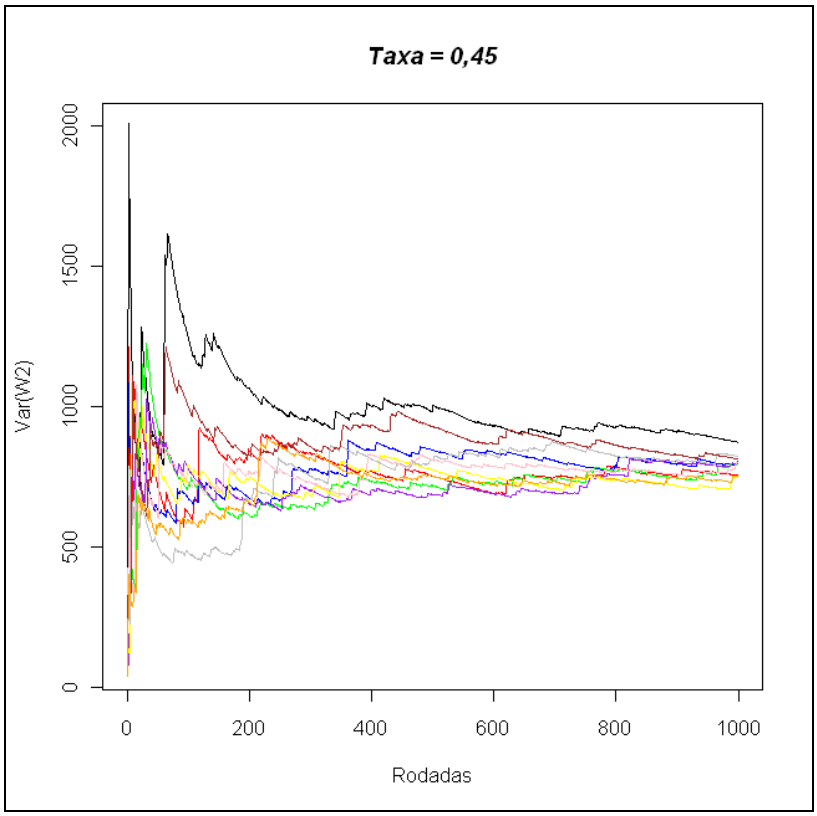

Figura 3: Variância de W2, com taxa = 0.45. Com 300 clientes por rodada e mil rodadas.

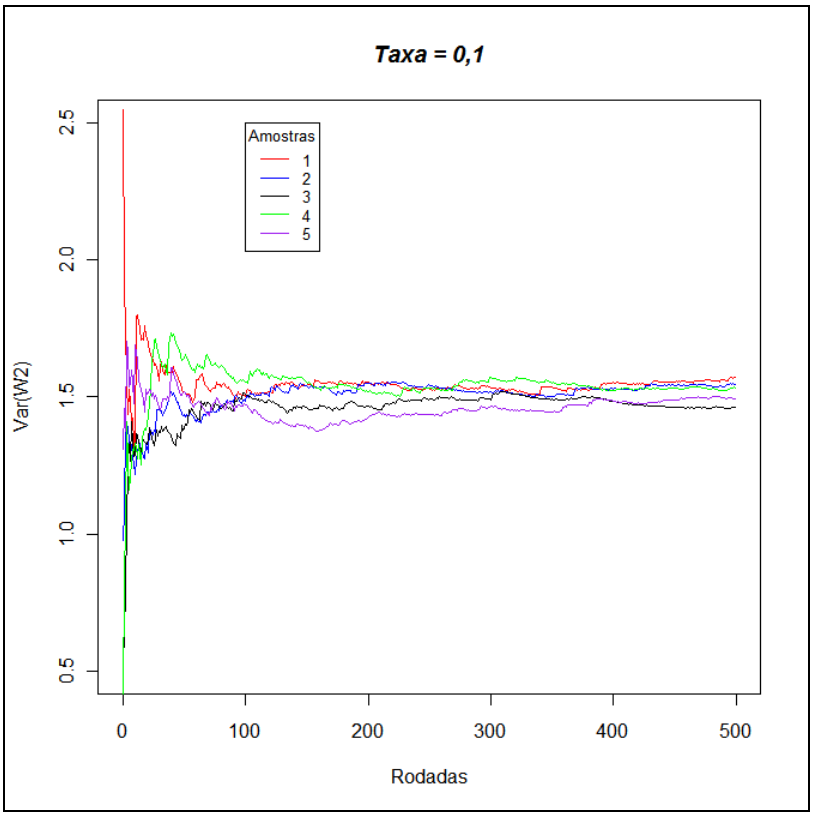

Figura 4: Variância de W2, com taxa = 0.1

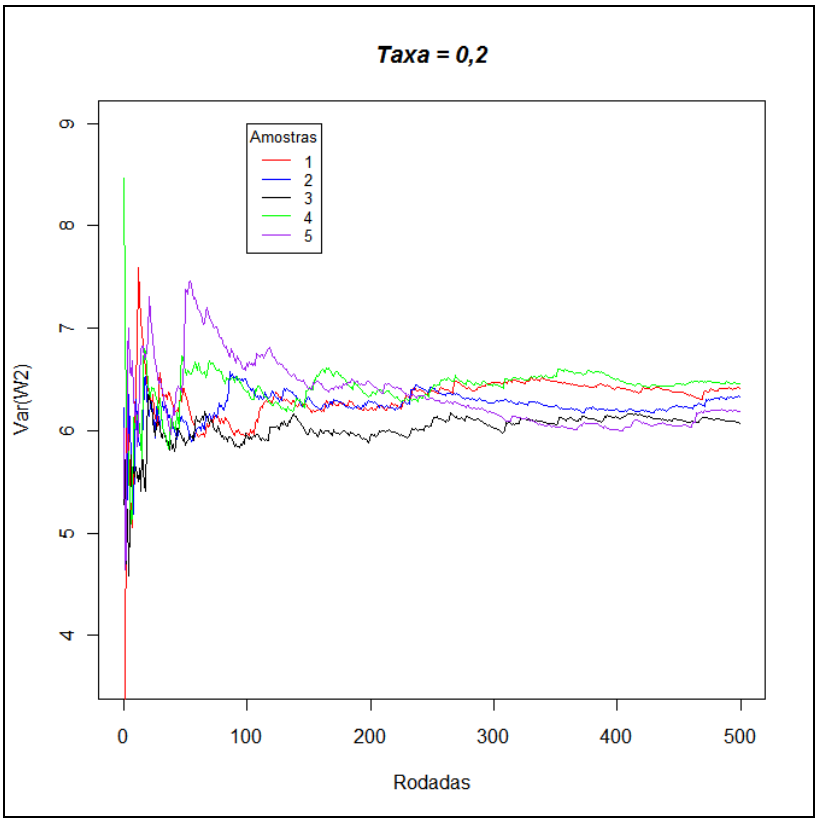

Figura 5: Variância de W2, com taxa = 0.2

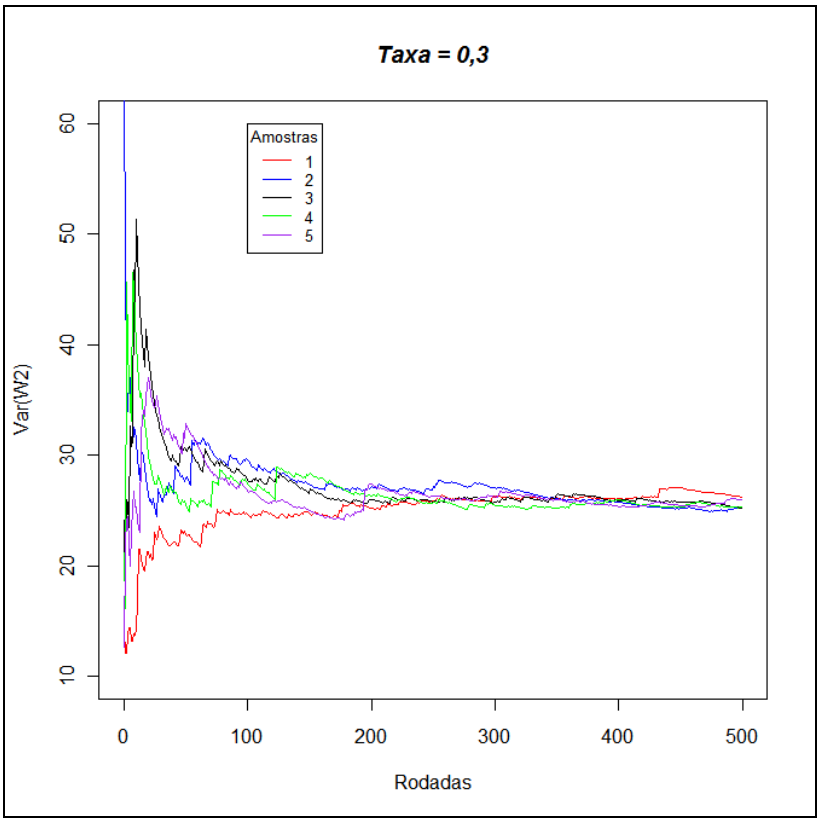

Figura 6: Variância de W2, com taxa = 0.3

4 Tabelas com comentários e resultados pertinentes

Para calcularmos os valores estimas dentro do intervalo pedido, decidimos realizar a simulação com mil rodadas e 30 mil clientes por rodada. Utilizamos o tamanho da fase transiente encontrado no tópico anterior. Logo, para λ = 0.1, 0.2 e 0.3 o tamanho da fase transiente será de 20 mil clientes, e para λ = 0.4 e 0.45 será de 60 mil clientes.

Escolhemos realizar a simulação com 30 mil clientes por rodada com base no tamanho das fases transientes encontradas. Como temos um mínimo de 20 mil e um máximo de 60 mil, para o tamanho das fases transientes, analisamos que seria adequado termos 30 mil clientes por rodada.

O número de rodadas da simulação foi escolhido devido ao fato de usarmos o método batch. Neste método, quanto maior for o número de rodadas menor é a influência da fase transiente.

Nas tabelas contidas neste tópico, utilizamos as seguintes fórmulas para obter os valores analíticos:

- **$E[W_1]$** = λ / (1- λ)
- **$E[T_1]$** = 1 / (1- λ)
- **$E[N_1]$** = λ / (1- λ)
- **$E[N_{q1}]$** = λ^2 / (1- λ)
- **$V(W_1)$** = λ(2-λ) / $(1-\lambda)^2$
- **$E[W_2]$** = [(λ + 1) / ((1- λ)*(1-2λ))] - 1
- **$E[T_2]$** = (λ + 1) / ((1- λ)*(1-2λ))
- **$E[N_2]$** = λ*(λ + 1) / ((1- λ)*(1-2λ))
- **$E[N_{q2}]$ =** λ*[(λ + 1) / ((1- λ)*(1-2λ))] – 1
- Para **$V(W_2)$**, não conhecemos resultados analíticos

Encontramos então os seguintes resultados:

Fila 1

$E[N_1]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	0.111111	0.111059	0.115710%
0.4	0.250000	0.249932	0.136510%
0.6	0.428571	0.428571	0.159219%
0.8	0.666666	0.666113	0.202219%
0.9	0.818181	0.818442	0.223574%

$E[N_{q1}]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	0.011111	0.011102	0.406826%
0.4	0.050000	0.049959	0.361896%
0.6	0.128571	0.128522	0.350879%
0.8	0.266666	0.266193	0.387332%
0.9	0.368181	0.368440	0.401037%

$E[T_1]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	1.111111	1.110896	0.083110%
0.4	1.250000	1.249549	0.106375%
0.6	1.428571	1.428730	0.127753%
0.8	1.666666	1.665244	0.169350%
0.9	1.818181	1.818651	0.194107%

$E[W_1]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	0.111111	0.111047	0.386303%
0.4	0.250000	0.249763	0.336806%
0.6	0.428571	0.428441	0.323300%
0.8	0.666666	0.665444	0.356100%
0.9	0.818181	0.818681	0.373304%

$V(W_1)$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	0.234568	0.234512	0.712799%
0.4	0.562500	0.561437	0.661431%
0.6	1.040816	1.040318	0.672054%
0.8	1.777778	1.767676	0.786847%
0.9	2.305785	2.305012	0.849597%

Fila 2

$E[N_2]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	0.527777	0.152732	0.158145%
0.4	0.500000	0.500033	0.240097%
0.6	1.392857	1.392537	0.367841%
0.8	4.666666	4.666188	0.722730%
0.9	11.863636	11.931689	1.354020%

$E[N_{q2}]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	0.052777	0.052754	0.323979%
0.4	0.300000	0.300053	0.356234%
0.6	1.092857	1.092508	0.451195%
0.8	4.266666	4.266168	0.784695%
0.9	11.41363	11.481606	1.404781%

$E[T_2]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	1.527777	1.527714	0.122755%
0.4	2.500000	2.499803	0.201949%
0.6	4.642857	4.641741	0.328206%
0.8	11.666666	11.661364	0.684003%
0.9	26.363636	26.495436	1.316546%

$E[W_2]$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	0.527777	0.527648	0.292727%
0.4	1.500000	1.499982	0.319478%
0.6	3.642857	3.641517	0.411988%
0.8	10.666666	10.661314	0.746115%
0.9	25.363636	25.495285	1.367358%

$V(W_2)$

ρ	Valor Analítico	Valor médio Estimado	Tamanho do Intervalo de Confiança
0.2	-	1.491388	0.667198%
0.4	-	6.398810	0.859755%
0.6	-	25.867145	1.205046%
0.8	-	164.437179	2.283034%
0.9	-	793.850260	4.368728%

Pode-se observar que em todas as tabelas o valor analítico está dentro do intervalo de confiança proposto. Como já era esperado, quanto maior a utilização, maior o intervalo. No ponto mais crítico, a variância de W_2, vemos que o intervalo de confiança conseguiu ficar abaixo dos 5% propostos.

5 Otimização

Para que fosse possível calcular o fator mínimo para cada taxa, criamos outro programa, também em C++, que é responsável por este cálculo. Para os cálculos, foram utilizados os mesmos tamanho da fase transiente obtidos no tópico 3.

Inicialmente, a ideia do grupo, era que o programa fosse incrementando o número de rodadas, para um tamanho de rodada fixo, até obter uma simulação onde todos os intervalos de confiança, em relação ao valor médio estimado, fossem menores do que 5%. Após isso, o tamanho da rodada seria incrementado e a operação repetida até um número máximo do tamanho das rodadas.

Porém, após alguns testes, foi observado que utilizando o mesmo número de rodadas e tamanho da rodada, mas com sementes diferentes, o valor do intervalo de confiança era muito instável. Muitas vezes passando do intervalo proposto. Logo, o intervalo máximo permitido nos testes, foi alterado para 4%. Assim, teríamos uma maior garantia no fator mínimo encontrado.

Para podermos garantir que o fator mínimo encontrado, não dependesse da semente, é executado um teste de dependência no fim de cada obtenção do valor mínimo. A simulação é então feita com quinhentas sementes diferentes. O fator mínimo obtido só é aceito, se em pelo menos 95% das diferentes simulações, obtivemos todos os intervalos de confiança menores que 5%.

Obtivemos então os seguintes resultados:

ρ	Fase transiente	Número de rodadas	Tamanho das rodadas	Fator mínimo
0.2	20000	375	2150	826250
0.4	20000	275	3900	1092500
0.6	20000	300	7250	2195000
0.8	60000	300	24000	7220000
0.9	60000	600	48200	28940000

6 Conclusão

Foram encontradas algumas dificuldades no percurso da criação do simulador. Uma das primeiras que tivemos que enfrentar, foi demostrar que o simulador estava funcionando corretamente, devido ao fato de que não tínhamos nenhum método claro e já documentado de demonstrar este fato, sem utilizar fórmulas analíticas.

Na determinação do fim da fase transiente também encontramos algumas dificuldades. Só após criarmos os gráficos para cada valor de utilização é que foi possível estabelecer um número razoável de clientes a ser descartados. Mesmo assim, o processo ainda pode ser considerado empírico.

O simulador criado pelo grupo obteve bons tempos de execução, e foi implementado de forma coesa e relativamente simples. Um dos objetivos do grupo era que o simulador fosse rápido e prático. Sendo assim, consideramos que conseguimos atingir nossos objetivos.

Apesar de não termos criado nenhuma interface gráfica, o carregamento das entradas através de arquivos externos, facilita e possibilita organizar melhor os diferentes cenários em que o usuário quer executar a simulação.

No que tange o software criado para obtenção do fator mínimo, o grupo considerou sua implementação não muito otimizada. O tempo de execução do programa foi considerado demasiadamente extenso. Isso já era esperado pelo grupo, devido ao fato de que, o programa irá rodar a simulação para um valor muito extenso de rodadas e tamanhos de rodadas diferentes. Porém, apesar de não ser rápido, consideramos que os fatores mínimos obtidos são razoavelmente bons.

A realização deste trabalho ajudou aos integrantes do grupo a entenderem melhor como um simulador funciona. Ao utilizarmos os conhecimentos obtidos em sala de uma forma prática, as explicações teóricas se tornam mais claras.

ANEXOS

ANEXO I
Outputs do simulador

Outputs do modo determinístico simples

Exemplo de execução do programa rodando em modo determinístico simples, com de interrupção forçada com λ = 0.2, ou seja, chegadas a cada 5 segundos. Mostra um ciclo de atendimento, que começa com a chegada no instante 5. A chegada do instante 10 é o próximo cliente a ser atendido.

```
Evento sendo tratado: Evento do tipo Nova Chegada no instante 5
A fila de Eventos tem tamanho 0 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O servidor esta vazio
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Agendando proxima chegada para o instante 10
          Inserindo o cliente 0 na fila 1
          Transferindo cliente 0 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 0 para o
instante 6
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 6
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 1
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Fim de servico na fila 1. Inserindo cliente 0 na fila 2
          Dados do cliente 0: W1 =  0, T1 = 1
          Transferindo cliente 0 da fila 2 para o servidor.
          Agendando termino do segundo servico do cliente 0 para o
instante 7
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 7
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 2
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Fim de servico na fila 2. Removendo cliente 0 do sistema
          Dados do cliente 0: W2 =  0, T2 = 1
A fila de Eventos tem tamanho 1

Evento sendo tratado: Evento do tipo Nova Chegada no instante 10
A fila de Eventos tem tamanho 0 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O servidor esta vazio
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Agendando proxima chegada para o instante 15
          Inserindo o cliente 1 na fila 1
          Transferindo cliente 1 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 1 para o
instante 11
A fila de Eventos tem tamanho 2
```

Output do resultado de uma rodada com 10.000 clientes no modo determinístico simples com λ = 0.2.

```
Imprimindo resultados da rodada 1 :
     E[W1] = 0
     E[T1] = 1
     V[W1] = 0
     E[Nq1] = 0
     E[N1] = 0.199992

     E[W2] = 0
     E[T2] = 1
     V[W2] = 0
     E[Nq2] = 0
     E[N2] = 0.199992
```

Output do mesmo caso, com λ = 0.4.

```
Imprimindo resultados da rodada 1 :
     E[W1] = 0
     E[T1] = 1
     V[W1] = 0
     E[Nq1] = 0
     E[N1] = 0.399968

     E[W2] = 0
     E[T2] = 1
     V[W2] = 0
     E[Nq2] = 0
     E[N2] = 0.399968
```

Outputs do modo determinístico "dois por vez"

Exemplo de execução do programa rodando em modo determinístico "dois por vez" com λ = 0.2, ou seja chegadas a cada 10 segundos (lembrando que o este modo trabalha com metade da taxa dada pelo usuário). Mostra um ciclo de atendimento, que começa com duas chegadas no instante 10. As duas chegada sdo instante 20 são clientes do ciclo seguinte

```
Evento sendo tratado: Evento do tipo Nova Chegada no instante 10
A fila de Eventos tem tamanho 0 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O servidor esta vazio
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Agendando proxima chegada para o instante 20
          Inserindo o cliente 0 na fila 1
          Agendando chegada artificial para o instante 10
          Transferindo cliente 0 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 0 para o
instante 11
A fila de Eventos tem tamanho 3

Evento sendo tratado: Evento do tipo Chegada Artificial no instante 10
A fila de Eventos tem tamanho 2 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 1
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
```

```
          Inserindo o cliente 1 na fila 1
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 11
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 1
     Numero de pessoas na fila 1: 1
     Numero de pessoas na fila 2: 0
          Fim de servico na fila 1. Inserindo cliente 0 na fila 2
          Dados do cliente 0: W1 =  0, T1 = 1
          Transferindo cliente 1 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 1 para o
instante 12
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 12
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 1 esta em servico, vindo da fila 1
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 1
          Fim de servico na fila 1. Inserindo cliente 1 na fila 2
          Dados do cliente 1: W1 =  1, T1 = 2
          Transferindo cliente 0 da fila 2 para o servidor.
          Agendando termino do segundo servico do cliente 0 para o
instante 13
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 13
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 2
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 1
          Fim de servico na fila 2. Removendo cliente 0 do sistema
          Dados do cliente 0: W2 =  1, T2 = 2
          Transferindo cliente 1 da fila 2 para o servidor.
          Agendando termino do segundo servico do cliente 1 para o
instante 14
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 14
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 1 esta em servico, vindo da fila 2
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Fim de servico na fila 2. Removendo cliente 1 do sistema
          Dados do cliente 1: W2 =  1, T2 = 2
A fila de Eventos tem tamanho 1

Evento sendo tratado: Evento do tipo Nova Chegada no instante 20
A fila de Eventos tem tamanho 0 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O servidor esta vazio
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Agendando proxima chegada para o instante 30
          Inserindo o cliente 2 na fila 1
          Agendando chegada artificial para o instante 20
          Transferindo cliente 2 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 2 para o
instante 21
A fila de Eventos tem tamanho 3
```

```
Evento sendo tratado: Evento do tipo Chegada Artificial no instante 20
A fila de Eventos tem tamanho 2 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 2 esta em servico, vindo da fila 1
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Inserindo o cliente 3 na fila 1
A fila de Eventos tem tamanho 2
```

Output do resultado de uma rodada com 10.000 clientes no modo determinístico "dois por vez" com λ = 0.2:

```
Imprimindo resultados da rodada 1 :
     E[W1] = 0.5
     E[T1] = 1.5
     V[W1] = 0.250025
     E[Nq1] = 0.099992
     E[N1] = 0.299976

     E[W2] = 1
     E[T2] = 2
     V[W2] = 0
     E[Nq2] = 0.199984
     E[N2] = 0.399968
```

Ou*tput do mesmo modo, com 10.000 clientes e λ = 0.4:*

```
Imprimindo resultados da rodada 1 :
     E[W1] = 0.5
     E[T1] = 1.5
     V[W1] = 0.250025
     E[Nq1] = 0.199968
     E[N1] = 0.599904

     E[W2] = 1
     E[T2] = 2
     V[W2] = 0
     E[Nq2] = 0.399936
     E[N2] = 0.799872
```

Outputs do modo determinístico de interrupção forçada

Exemplo de execução do programa rodando em modo determinístico de interrupção forçada com λ = 0.2, ou seja chegadas a cada 10 segundos (lembrando que o modo de interrupção forçada trabalha com metade da taxa dada pelo usuário). Mostra um ciclo de atendimento, que começa com a chegada no instante 10. A chegada do instante 20 já é a do primeiro cliente do ciclo seguinte.

```
Evento sendo tratado: Evento do tipo Nova Chegada no instante 10
A fila de Eventos tem tamanho 0 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O servidor esta vazio
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Agendando proxima chegada para o instante 20
          Inserindo o cliente 0 na fila 1
          Agendando chegada artificial para o instante 11.5
          Transferindo cliente 0 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 0 para o
instante 11
A fila de Eventos tem tamanho 3

Evento sendo tratado: Evento do tipo Termino de Servico no instante 11
A fila de Eventos tem tamanho 2 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 1
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Fim de servico na fila 1. Inserindo cliente 0 na fila 2
          Dados do cliente 0: W1 =  0, T1 = 1
          Transferindo cliente 0 da fila 2 para o servidor.
          Agendando termino do segundo servico do cliente 0 para o
instante 12
A fila de Eventos tem tamanho 3

Evento sendo tratado: Evento do tipo Chegada Artificial no instante
11.5
A fila de Eventos tem tamanho 2 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 2
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Evento sendo destruido: : Evento do tipo Termino de Servico
marcado para o instante 12
          Inserindo o cliente 1 na fila 1
          Transferindo cliente 1 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 1 para o
instante 12.5
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante
12.5
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 1 esta em servico, vindo da fila 1
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 1
          Fim de servico na fila 1. Inserindo cliente 1 na fila 2
          Dados do cliente 1: W1 =  0, T1 = 1
```

```
          Inserindo cliente 0 da fila 2 no servidor. Este cliente ja
foi interrompido alguma vez.
          Agendando termino do segundo servico do cliente 0 para 13
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 13
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 0 esta em servico, vindo da fila 2
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 1
          Fim de servico na fila 2. Removendo cliente 0 do sistema
          Dados do cliente 0: W2 =  1, T2 = 2
          Transferindo cliente 1 da fila 2 para o servidor.
          Agendando termino do segundo servico do cliente 1 para o
instante 14
A fila de Eventos tem tamanho 2

Evento sendo tratado: Evento do tipo Termino de Servico no instante 14
A fila de Eventos tem tamanho 1 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O cliente numero 1 esta em servico, vindo da fila 2
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Fim de servico na fila 2. Removendo cliente 1 do sistema
          Dados do cliente 1: W2 =  0.5, T2 = 1.5
A fila de Eventos tem tamanho 1

Evento sendo tratado: Evento do tipo Nova Chegada no instante 20
A fila de Eventos tem tamanho 0 apos remover o evento atual
Status do sistema (antes de resolver o evento):
     O servidor esta vazio
     Numero de pessoas na fila 1: 0
     Numero de pessoas na fila 2: 0
          Agendando proxima chegada para o instante 30
          Inserindo o cliente 2 na fila 1
          Agendando chegada artificial para o instante 21.5
          Transferindo cliente 2 da fila 1 para o servidor.
          Agendando termino do primeiro servico do cliente 2 para o
instante 21
A fila de Eventos tem tamanho 3
```

Output do resultado de uma rodada com 10.000 clientes no modo determinístico de interrupção forçada com λ = 0.2.

```
Imprimindo resultados da rodada 1 :
     E[W1] = 0
     E[T1] = 1
     V[W1] = 0
     E[Nq1] = 0
     E[N1] = 0.199984

     E[W2] = 0.75
     E[T2] = 1.75
     V[W2] = 0.0625063
     E[Nq2] = 0.149988
     E[N2] = 0.349972
```

Com λ = 0.4, temos o seguinte resultado, para uma rodada com 10.000 clientes:

```
Imprimindo resultados da rodada 1:
     E[W1] = 0
     E[T1] = 1
     V[W1] = 0
     E[Nq1] = 0
     E[N1] = 0.399936

     E[W2] = 0.75
     E[T2] = 1.75
     V[W2] = 0.0625063
     E[Nq2] = 0.299952
     E[N2] = 0.699888
```

ANEXO II
Gráficos gerados

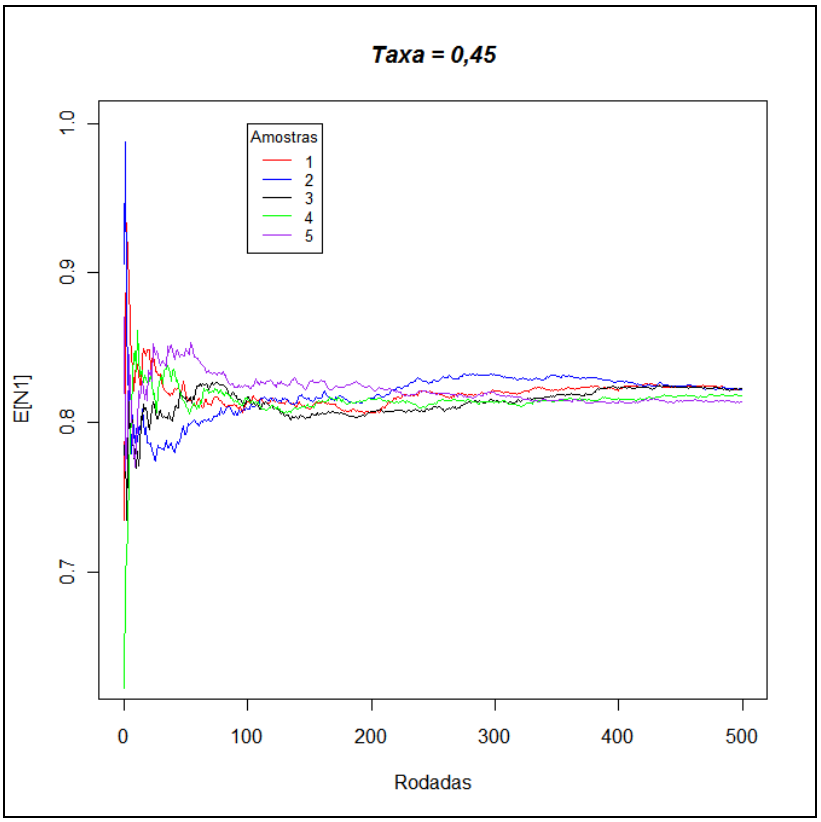

Figura 7: Esperança de N1, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

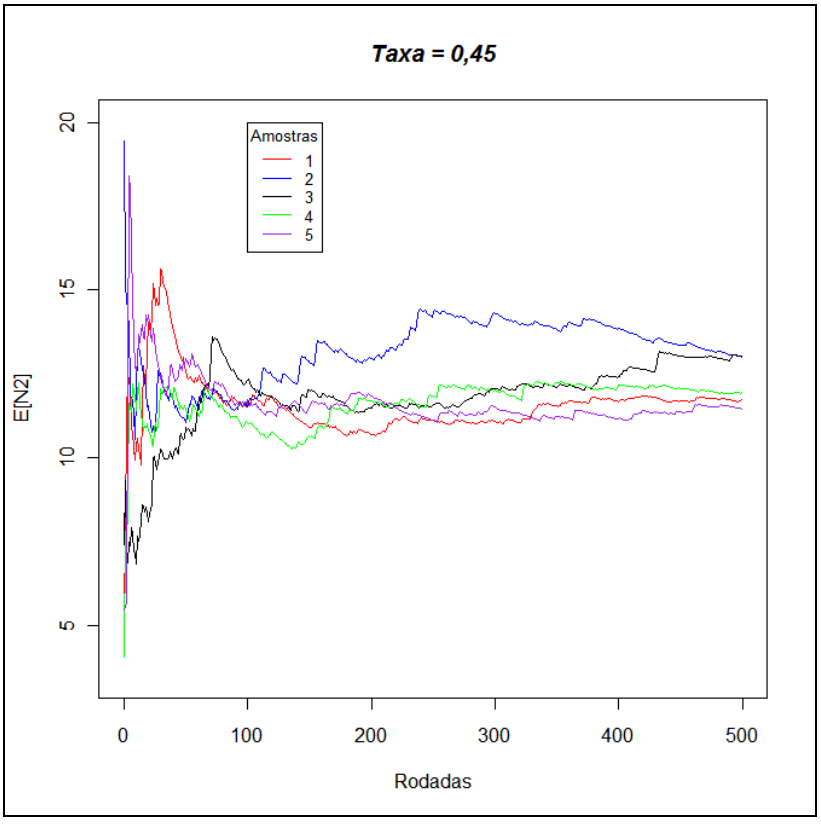

Figura 8: Esperança de N2, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

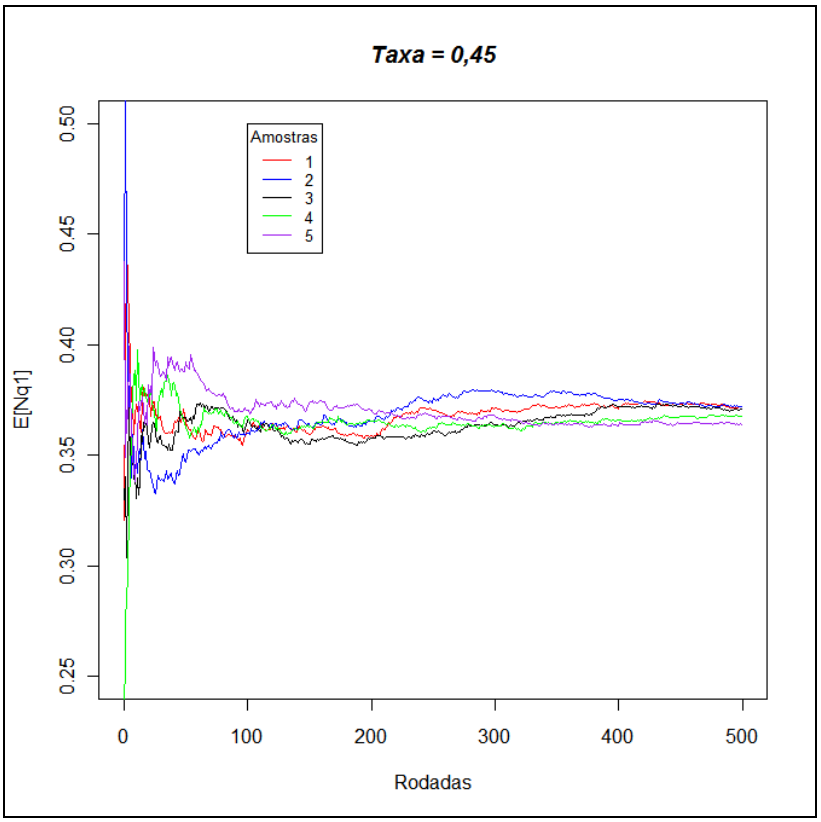

Figura 9: Esperança de Nq1, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

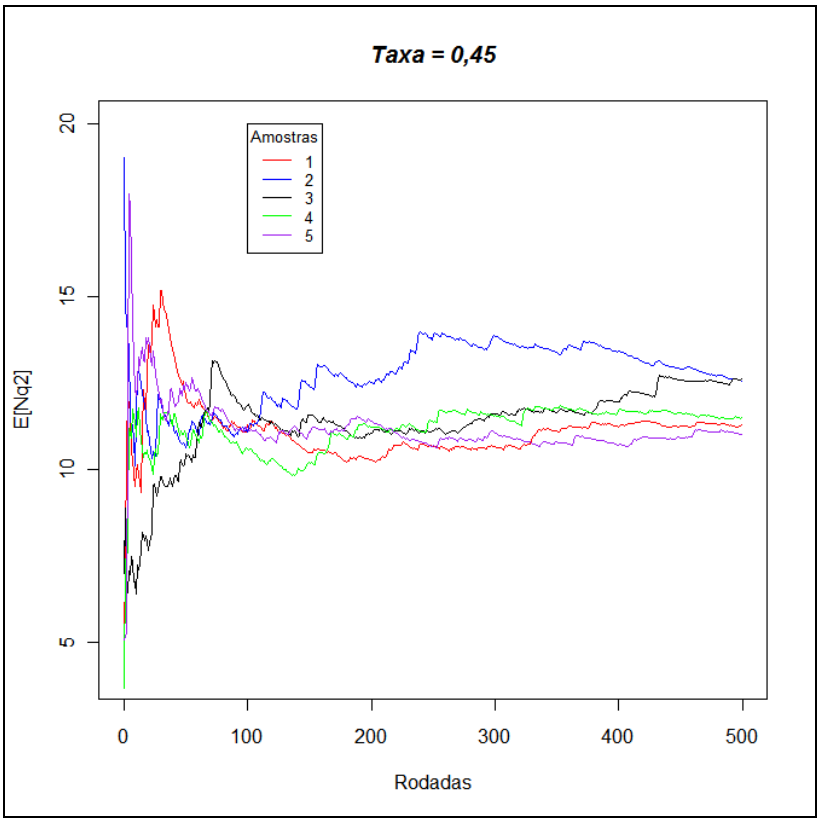

Figura 10: Esperança de Nq2, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

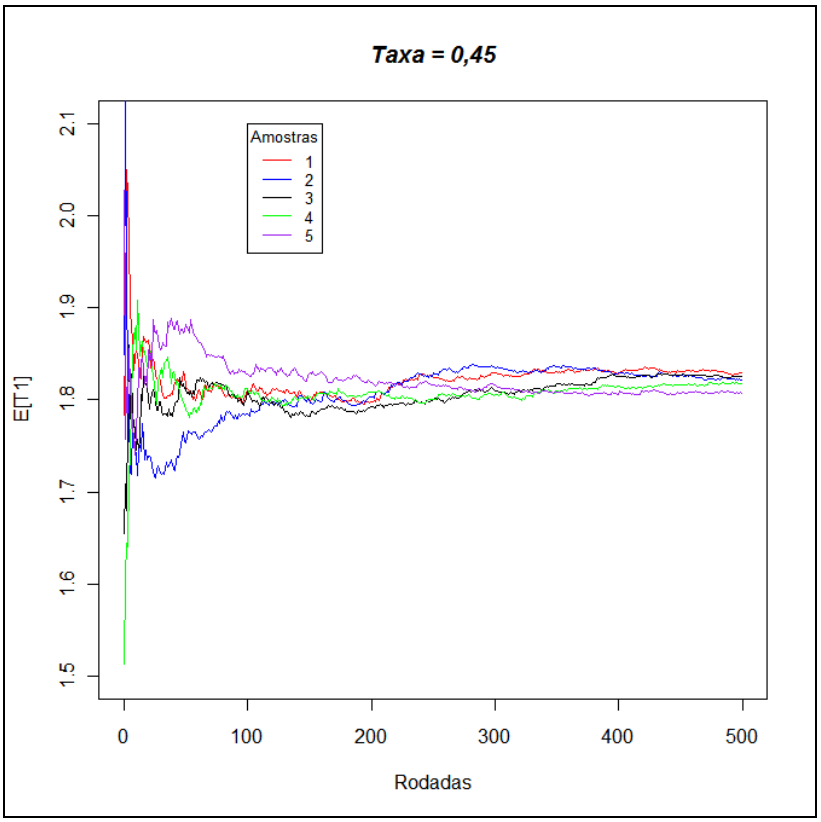

Figura 11: Esperança de T1, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

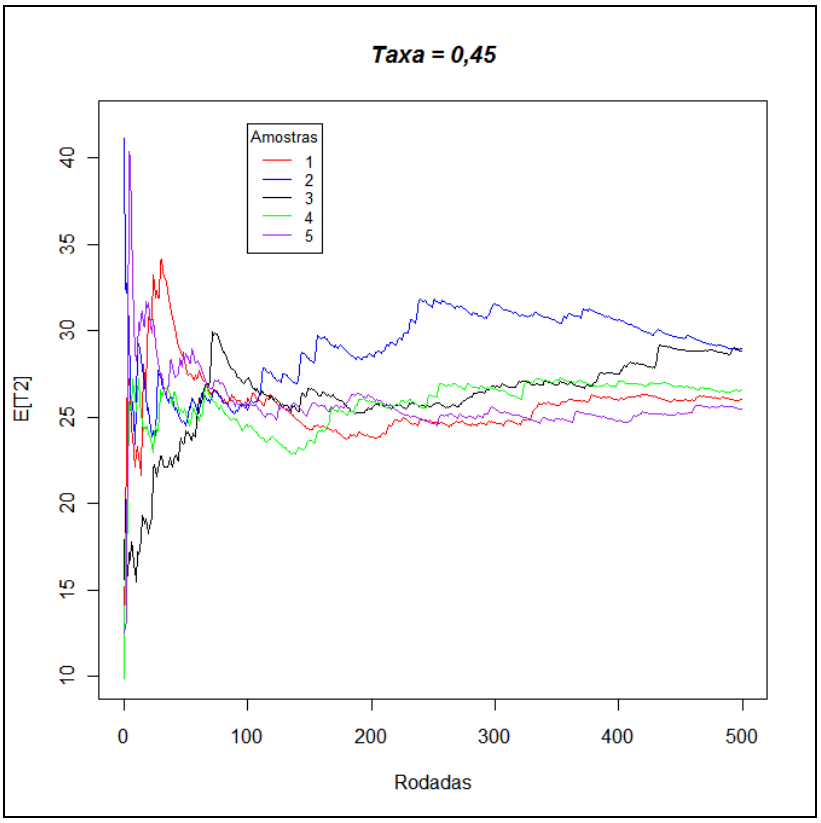

Figura 12: Esperança de T2, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

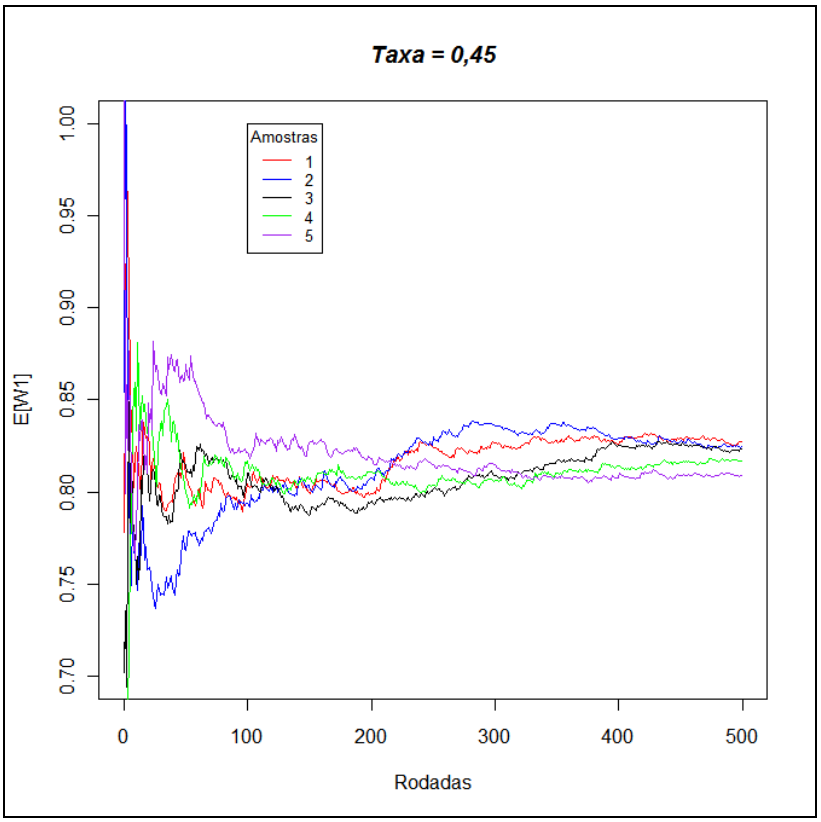

Figura 13: Esperança de W1, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

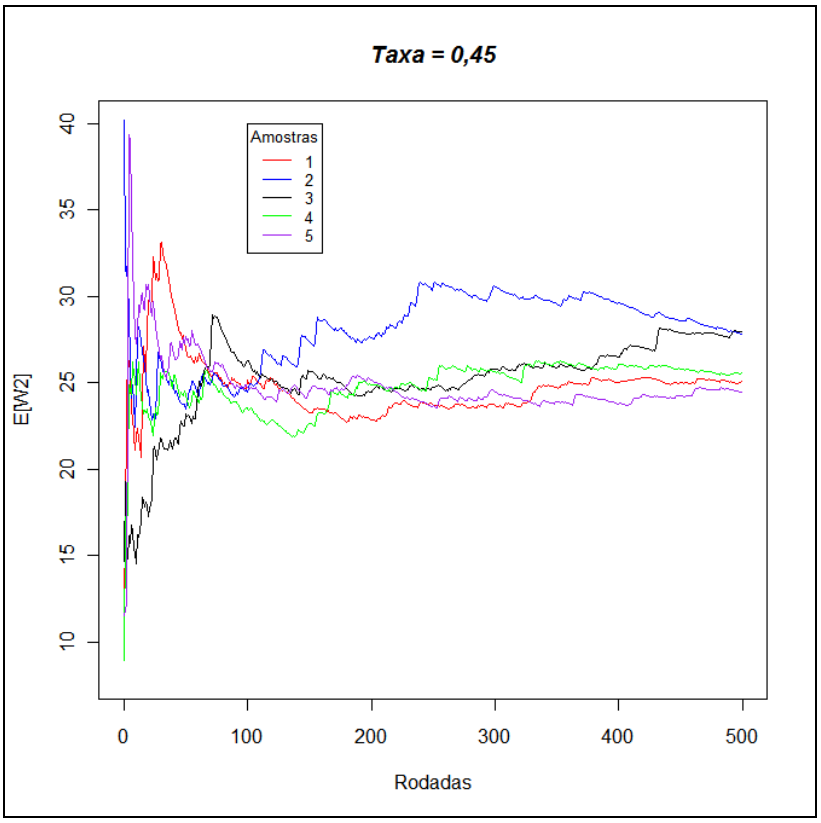

Figura 14: Esperança de W2, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

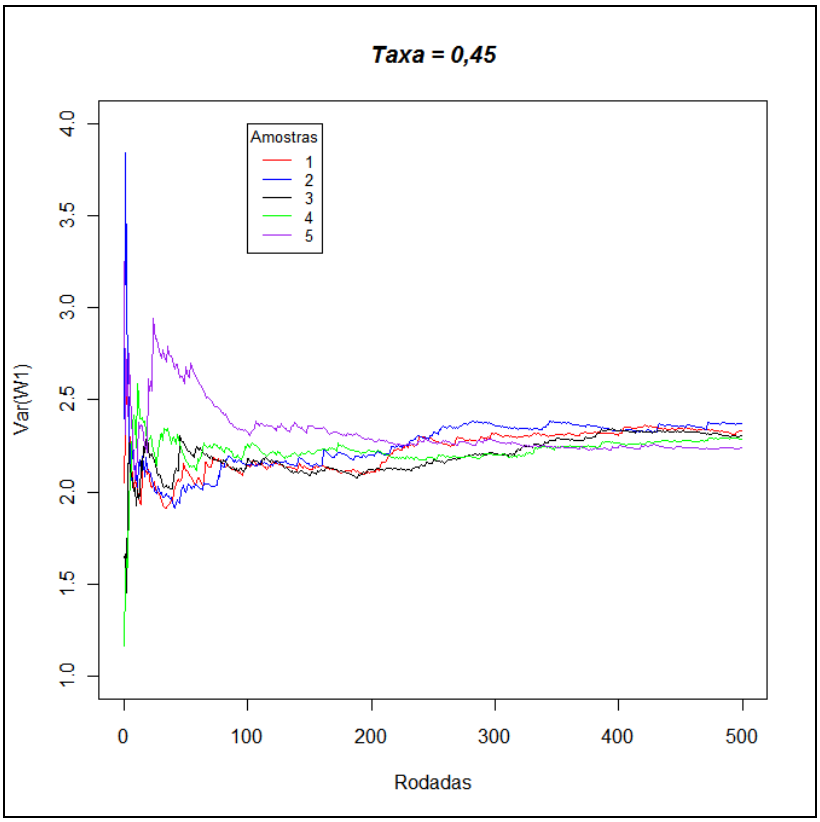

Figura 15: Variância de W1, com taxa = 0.45. Com 300 clientes por rodada e 500 rodadas

ANEXO III
Código fonte

Simulador.cpp

```
////////////////////////////////////////////////////////
/*
          Avaliação e Desempenho

          Trabalho de Simulação



Alunos:
     Luiz Filipe de Sá Estrella       DRE: 107390627
     Fernando de Mesentier Silva      DRE: 107390520
     Rodrigo de Moura Canaan          DRE: 107362200
     Vinicius José Serva Pereira      DRE: 106050355


*/
////////////////////////////////////////////////////////

#include "include\Simulador.h"
#include <math.h>
#define FILA_1 1
#define FILA_2 2
#define INTERROMPIDO 1
#define N_INTERROMPIDO 0

using namespace std;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
     ---------------------   -Contrutores & Destrutor----------------------
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

Simulador::Simulador(double ptaxa_chegada, double ptaxa_servico, bool
deterministico, bool dois_por_vez, bool interrupcao_forcada, int semente)
{

    /*
          No modo "dois por vez" e de "interrupção forçada", trabalhamos com
          metade da taxa de chegada fornecida pelo usuário. Assim mantemos a
taxa efetiva igual a que foi dada
     */
    if(dois_por_vez or interrupcao_forcada)
                    taxa_chegada = ptaxa_chegada/2;
    else
                    taxa_chegada = ptaxa_chegada;

    taxa_servico = ptaxa_servico;


     Setup(semente, deterministico);



     acumulaW1 = 0.0;
     acumulaT1 = 0.0;
     acumulaW2 = 0.0;
```

```
     acumulaT2 = 0.0;
     acumula_quadradoW1 = 0.0;
     acumula_quadradoW2 = 0.0;

     /*
          Inicia as variáveis que acumulam o (numero de pessoas * tempo) de
cada região do sistema
     */
     Nq1_parcial = 0.0;
     Nq2_parcial = 0.0;
     N1_parcial = 0.0;
     N2_parcial = 0.0;


}

Simulador::Simulador()
{
}

Simulador::~Simulador()
{
     //Destrutor
}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
          ------------------------Funções Membro---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

/*
     Função principal do simulador, executa a simulação.
*/
void Simulador::Roda(int num_clientes_por_rodada, int rodada_atual, bool
debug, bool deterministico, bool determina_transiente, bool dois_por_vez,
string nome_pasta, bool guardar_estatisticas, bool interrupcao_forcada, bool
mostrar_resultados)
{
    int num_servicos_tipo_1_rodada_atual = 0;
     int num_servicos_tipo_2_rodada_atual = 0;

     double tempo_inicio_rodada = tempo_atual;


     if(dois_por_vez)
     {
                    cout << "Rodando em modo dois por vez" << endl;
    }


     /*
          Loop principal do simulador.
          Enquanto o número total de clientes que queremos servir for maior que
o número de clientes já servidos por completo, rodamos a simulação
     */
     while(num_clientes_por_rodada > num_servicos_tipo_2_rodada_atual)
     {
          /*
               O primeiro evento é selecionado
          */
          Evento evento_atual = filaEventos.top();
          /*
```

```
            Ele é retirado da Fila
          */
          filaEventos.pop();

          double tempo_desde_evento_anterior =
evento_atual.GetTempoAcontecimento()-tempo_atual;
          tempo_atual=evento_atual.GetTempoAcontecimento();

          if(debug)
          {
               cout << endl << "Evento sendo tratado: Evento do tipo " <<
evento_atual.GetNome() << " no instante " << tempo_atual << endl;
               cout << "A fila de Eventos tem tamanho " << filaEventos.size()
<< " apos remover o evento atual" << endl;
               cout << "Status do sistema (antes de resolver o evento):" <<
endl;
               if(servidor_vazio)
                                    cout << "     O servidor esta vazio" <<
endl;
               else
                                    cout << "     O cliente numero " <<
cliente_em_servico.GetID() << " esta em servico, vindo da fila " <<
cliente_em_servico.GetFila() <<endl;
               cout << "     Numero de pessoas na fila 1: " << fila1.size()  <<
endl;
               cout << "     Numero de pessoas na fila 2: " << fila2.size() <<
endl;
        }



////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
          ------------------------COLETA DE DADOS---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

        /* Para cada uma das variaveis Nq1, Nq2, N1 e N2,
           calcula a "area sob a curva" desde o ultimo evento, multiplicando
           o valor da variavel pelo tamanho do intervalo entre o evento
anterior
           e o evento atual.
           Como entre eventos nada muda no servidor, cada uma dessas variaveis
           permaneceu constante
                     Nq1 e Nq2 sao o numero de pessoas nas filas de espera 1
e 2, respectivamente
                     N1 e N2 sao o numero de pessoas esperando cada tipo de
servico na fila
                     Ni = Nqi + 1 se houver um cliente vindo da fila i no
servidor
                     Ni = Nqi caso contrario
           No final da simulacao, teremos a área sob a curva de cada uma
dessas variaveis.
           Faltara dividir pelo tempo total decorrido para calcular a media.
        */
          if (rodada_atual != 0)
          {
               Nq1_parcial += fila1.size()*tempo_desde_evento_anterior;
               Nq2_parcial += fila2.size()*tempo_desde_evento_anterior;
               if (!servidor_vazio)
               {
                  if (cliente_em_servico.GetFila() == 1)
                  {
```

```
                    N1_parcial +=
(1+fila1.size())*tempo_desde_evento_anterior;
                    N2_parcial += fila2.size()*tempo_desde_evento_anterior;
                }
                else
                {
                      N1_parcial += fila1.size()*tempo_desde_evento_anterior;
                      N2_parcial +=
(1+fila2.size())*tempo_desde_evento_anterior;
                }
            }
            else
            {
                  N1_parcial += fila1.size()*tempo_desde_evento_anterior;
                  N2_parcial += fila2.size()*tempo_desde_evento_anterior;
            }
        }

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
     ------------------------FIM DA COLETA DE DADOS--------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////



        /*
            Tratamos aqui o evento do tipo nova chegada.
            Se o caso deterministico interrupcao_forcada estiver ligado
também temos a possibilidade de entrar no condicional com uma
chegada_artificial
        */
        if(evento_atual.GetTipo() == nova_chegada or evento_atual.GetTipo()
== chegada_artificial)
        {
            /*
                Condição para tratar a interrupção presente no sistema.
                Se o servidor estiver ocupado e este cliente for da fila 2
então o cliente que acabou de chegar irá interromper este serviço
            */
            if(!servidor_vazio && cliente_em_servico.GetFila() == FILA_2)
            {
              Evento evento_destruido = filaEventos.top();

                /*
                    No caso de interrupcao_forcada, é necessário realizar o
método RemoveTerminoServico() para remover o evento correto,
                    pois não temos garantia de que o evento a ser removido
é o do topo, pois pode haver mais de um evento de chega
                    na heap por vez.
                */
              if(interrupcao_forcada){
                    evento_destruido = RemoveTerminoServico();
                    if(debug)
                       {
                       cout << "          Evento sendo destruido: : Evento do
tipo " << evento_destruido.GetNome() << " marcado para o instante " <<
evento_destruido.GetTempoAcontecimento()<< endl;
                    }
              }
                 else{
                    /*
                        Remove o Evento de Término de serviço gerado por
```

```
este cliente da fila 2.
                        Como no modo de execução normal guardamos apenas um
evento de chegada e um de término de serviço
                        por vez na fila de enventos, e a chegada atual já foi
removida nesse ponto de execução
                        do programa, mas a próxima chegada ainda não foi
adicionada, podemos garantir que o
                        evento de término de serviço que devemos cancelar vai
estar no topo da heap.
                        */
                    filaEventos.pop();
                    if(debug)
                        {
                        cout << "          Evento sendo destruido: : Evento do
tipo " << evento_destruido.GetTipo() << " marcado para o instante " <<
evento_destruido.GetTempoAcontecimento()<< endl;
                    }
                }
                    /*
                        Marca o cliente como sendo um cliente Interrompido
                    */
                    cliente_em_servico.Interromper();
                    /*
                        O tempo restante para finalizar seu servico é guardado
                    */

    cliente_em_servico.SetTempoRestante(evento_destruido.GetTempoAcontecimento
() - tempo_atual);

                    /*
                        Cliente que foi interrompido volta a ser o primeiro da
fila 2.
                    */
                    fila2.push_front(cliente_em_servico);
                    /*
                        Deixa o servidor vazio
                    */
                    servidor_vazio = true;
                }

                /*
                    O novo cliente começa na fila 1
                */
                Cliente cliente_atual =
Cliente(id_proximo_cliente,tempo_atual,FILA_1, rodada_atual);


                if(servidor_vazio)
                {
                    cliente_atual.SetDiretoAoServidor(true);
                }

                id_proximo_cliente++;
                /*
                    Coloca o novo cliente na fila 1
                */
                fila1.push(cliente_atual);

                if(evento_atual.GetTipo() == nova_chegada)
            {
                 Evento proxChegada = Evento(nova_chegada,tempo_atual+gerador-
>GeraTempoExponencial(taxa_chegada, deterministico));//Agenda o Evento para a
próxima chegada
                  filaEventos.push(proxChegada);
```

```
                    if(debug)
                {
                                        cout << "          Agendando proxima
chegada para o instante " << proxChegada.GetTempoAcontecimento() << endl;
                }
            }

              if(debug)
              {
                   cout << "          Inserindo o cliente " <<
cliente_atual.GetID() << " na fila 1" << endl;
              }

              if(dois_por_vez and evento_atual.GetTipo() == nova_chegada)
              {
                              Evento artificial =
Evento(chegada_artificial,tempo_atual);
                                filaEventos.push(artificial);
                              if(debug)
                              {
                                                    cout << "          Agendando
chegada artificial para o instante " << artificial.GetTempoAcontecimento() <<
endl;
                              }
            }

            if(interrupcao_forcada and evento_atual.GetTipo() == nova_chegada)
            {
             Evento artificial =
Evento(chegada_artificial,tempo_atual+gerador-
>GeraTempoExponencial(taxa_servico*2/3, deterministico));
               filaEventos.push(artificial);
                              if(debug)
                              {
                                                    cout << "          Agendando
chegada artificial para o instante " << artificial.GetTempoAcontecimento() <<
endl;
                              }
            }


         }//Se o Evento, que está sendo tratado no momento, for do
termino_de_servico
         else if (evento_atual.GetTipo() == termino_de_servico)
         {
              if(cliente_em_servico.GetFila() == FILA_1)
              {
                   cliente_em_servico.SetFila(FILA_2);//O cliente que irá
terminar o serviço agora é definido como da fila 2
                   cliente_em_servico.SetInstanteChegada2(tempo_atual);

                   fila2.push_back(cliente_em_servico);// Coloca o cliente na
fila 2

                   //Se a fila 2 só tem o próprio cliente e não tem ninguem em
serviço, quer dizer que o cliente da fila 2 será atendido direto
                   if(fila2.size() == 1 && fila1.empty())
                        fila2.back().SetDiretoAoServidor(true);
                   else
                        fila2.back().SetDiretoAoServidor(false);

                   if(debug)
                   {
                        cout << "          Fim de servico na fila 1. Inserindo
```

```
cliente " << cliente_em_servico.GetID() << " na fila 2" << endl;
                    }


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
          ------------------------ COLETA DE DADOS---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

                    /*
                         Verifica se o cliente em serviço é da rodada(coloração
usada) em que estamos, pois só esse cliente entra nos dados desta rodada
                    */
                    if((cliente_em_servico.GetRodadaPertencente() ==
rodada_atual or determina_transiente) and rodada_atual != 0)
                    {
                         if(!cliente_em_servico.GetDiretoAoServidor())
                              cliente_W1 = cliente_em_servico.W1();
                         else
                              cliente_W1 = 0;


                         acumulaW1 += cliente_W1;
                         acumulaT1 += cliente_em_servico.T1();
                         acumula_quadradoW1 += cliente_W1*cliente_W1;

                         if(debug)
                         {
                              cout << "          Dados do cliente " <<
cliente_em_servico.GetID() << ": W1 =  " << cliente_W1 << ", T1 = " <<
cliente_em_servico.T1() << endl;
                         }
                         num_servicos_tipo_1_rodada_atual ++;
                         total_clientes_servidos_uma_vez ++;
                    }

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
     ------------------------FIM DA COLETA DE DADOS---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////



               }
               else
               {
                    //Acabou os 2 serviços do cliente, logo marcamos mais um
cliente servido totalmente
                    if(debug)
                    {
                         cout  <<"          Fim de servico na fila 2. Removendo
cliente " << cliente_em_servico.GetID() << " do sistema" << endl;
                    }
                    cliente_em_servico.SetInstanteSaida(tempo_atual);
```

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
        ------------------------ COLETA DE DADOS---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

                    //Verifica se o cliente em serviço é da rodada(coloração
usada) em que estamos, pois só esse cliente entra nos dados desta rodada
                    if((cliente_em_servico.GetRodadaPertencente() ==
rodada_atual or determina_transiente) and rodada_atual != 0)
                    {
                         total_clientes_servidos_duas_vezes ++;
                         num_servicos_tipo_2_rodada_atual ++;

                         if(cliente_em_servico.VerificaInterrompido() ==
N_INTERROMPIDO && cliente_em_servico.GetDiretoAoServidor())
                              cliente_W2 = 0;
                         else
                              cliente_W2 = cliente_em_servico.W2();

                         acumulaW2 += cliente_W2;
                         acumulaT2 += cliente_em_servico.T2();
                         acumula_quadradoW2 += cliente_W2*cliente_W2;

                         if(debug)
                         {
                              cout << "          Dados do cliente " <<
cliente_em_servico.GetID() << ": W2 =  " << cliente_W2 << ", T2 = " <<
cliente_em_servico.T2() << endl;
                         }
                    }

                    if(rodada_atual == 0)
                         num_servicos_tipo_2_rodada_atual ++;

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
      ------------------------FIM DA COLETA DE DADOS---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

               }

               //
               servidor_vazio = true;//Como alguem foi servido marcamos o
servidor como vazio
          }
          /*
               Se não tem ninguem no servidor
          */
          if(servidor_vazio == true )
          {
               //Fila 1 tem prioridade
               if(!fila1.empty())
               {
                    cliente_em_servico = fila1.front();//O primeiro cliente da
fila 1 entra em servico
                    fila1.pop();//O cliente é removido da fila 1

                    servidor_vazio = false;//O servidor agora está ocupado
                    double duracao = gerador-
>GeraTempoExponencial(taxa_servico, deterministico);
                    Evento proxTerminoServico =
Evento(termino_de_servico,tempo_atual+duracao);//Agendar evento de termino de
```

```
servico
                    filaEventos.push(proxTerminoServico);
                    cliente_em_servico.SetDuracaoPrimeiroServico(duracao);

                    if(debug)
                    {
                         cout << "          Transferindo cliente " <<
cliente_em_servico.GetID() << " da fila 1 para o servidor." << endl;
                         cout << "          Agendando termino do primeiro
servico do cliente " << cliente_em_servico.GetID() <<" para o instante " <<
proxTerminoServico.GetTempoAcontecimento() << endl;;
                    }
               }
               else if(!fila2.empty())
               {
                    cliente_em_servico = fila2.front();
                    fila2.pop_front();
                    servidor_vazio = false;

                    //Se o cliente, que veio da fila 2, não foi interrompido,
gere para ele o seu tempo de serviço
                    if(cliente_em_servico.VerificaInterrompido() ==
N_INTERROMPIDO)
                    {
                         double duracao = gerador-
>GeraTempoExponencial(taxa_servico,deterministico);
                          Evento proxTerminoServico =
Evento(termino_de_servico,tempo_atual+duracao);//Agenda evento de termino de
servico
                         filaEventos.push(proxTerminoServico);
                         cliente_em_servico.SetDuracaoSegundoServico(duracao);

                         if(debug)
                         {
                              cout << "          Transferindo cliente " <<
cliente_em_servico.GetID() << " da fila 2 para o servidor." << endl;
                              cout << "          Agendando termino do segundo
servico do cliente " << cliente_em_servico.GetID() << " para o instante " <<
proxTerminoServico.GetTempoAcontecimento() << endl;
                         }
                    }

                    else
                    {
                         Evento proxTerminoServico =
Evento(termino_de_servico,tempo_atual+cliente_em_servico.GetTempoRestante());/
/Agenda evento de termino de servico, com o tempo restante de servico do
cliente que foi interrompido
                         filaEventos.push(proxTerminoServico);
                         if(debug)
                         {
                              cout << "          Inserindo cliente " <<
cliente_em_servico.GetID() << " da fila 2 no servidor. Este cliente ja foi
interrompido alguma vez." << endl;
                              cout << "          Agendando termino do segundo
servico do cliente " << cliente_em_servico.GetID() << " para " <<
proxTerminoServico.GetTempoAcontecimento() << endl;
                             }
                    }


               }
          }
          if(debug)
```

```
                              cout << "A fila de Eventos tem tamanho " <<
filaEventos.size() << endl;
     }
     if( rodada_atual != 0)
     {
          if(!determina_transiente)
          {
               CalculaResultados(num_servicos_tipo_2_rodada_atual,
num_servicos_tipo_1_rodada_atual, tempo_atual - tempo_inicio_rodada,
rodada_atual, mostrar_resultados, nome_pasta, guardar_estatisticas);
          }
          else
          {
               CalculaResultados(total_clientes_servidos_duas_vezes,
total_clientes_servidos_uma_vez, tempo_atual, rodada_atual,
mostrar_resultados, nome_pasta, guardar_estatisticas);
          }
     }

}

void Simulador::CalculaResultados(int n, int servidos1, double t, int rodada,
bool mostrar_resultados, string nome_pasta, bool guardar_estatisticas)
{
    /*
          Divide cada uma das variaveis de fila Nq1, Nq2, N1 e N2 pelo tempo da
rodada
          para obter a media de cada uma delas
    */
    E_Nq1.push_back(Nq1_parcial/t);
    E_Nq2.push_back(Nq2_parcial/t);
    E_N1.push_back(N1_parcial/t);
    E_N2.push_back(N2_parcial/t);

    /*
          Divide cada um dos acumuladores dos clientes pelo numero de clientes
servidos
    */
    double EW1 = acumulaW1/servidos1;
    double ET1 = acumulaT1/servidos1;
    double EW2 = acumulaW2/n;
    double ET2 = acumulaT2/n;

    E_W1.push_back(EW1);
    E_T1.push_back(ET1);
    E_W2.push_back(EW2);
    E_T2.push_back(ET2);


     /*
          O estimador da variância é dado por (1/(n-1)) * somatório de ( ( Xi -
Xmédio)^2) para cada amostra i, se forem feitas n amostras
          mas
          somatório de ( ( Xi - Xmédio)^2) para cada amostra i =
          somatório de ( ( Xi^2 - 2Xi*Xmédio + Xmedio^2)) para cada amostra i
          passando o somatório para dentro, temos
          somatório de ( Xi^2) para cada amostra i - 2* somatório de (
Xi*Xmedio) para cada amostra i + somatório de (Xmédio)^2) para cada amostra i
          = acumula_quadradoX - 2*acumulaX*E[X] + n*E[X]^2

          mas E[X] = acumulaX/n, logo -2*acumulaX*E[X] = -2*E(X)^2*n
          e (numero de amostras)*E[X]^2 = acumulaX^2/n

          Assim, V(X) = (acumula_quadradoX - 2*acumulaX^2/n + acumulaX^2/n) *
```

```
1/(n-1)
          V(X) = (aumula_quadradoX - acumulaX^2/n) * 1/(n-1)

          para X = W1 ou X = W2
     */

     V_W1.push_back((acumula_quadradoW1 - EW1*EW1*servidos1)/(servidos1-1));
    V_W2.push_back((acumula_quadradoW2 - EW2*EW2*n)/(n-1));



    if(mostrar_resultados)
     {
          cout << endl <<endl << endl << "Imprimindo resultados da rodada "<<
rodada <<" :"<< endl;
          cout << "      E[W1] = " << E_W1.back() << endl;
          cout << "      E[T1] = " << E_T1.back() << endl;
          cout << "      V[W1] = " << V_W1.back() << endl;
         cout << "     E[Nq1] = " << E_Nq1.back() << endl;
         cout << "     E[N1] = " << E_N1.back() << endl << endl;
         cout << "     E[W2] = " << E_W2.back() << endl;
         cout << "     E[T2] = " << E_T2.back() << endl;
          cout << "      V[W2] = " << V_W2.back() << endl;
          cout << "      E[Nq2] = " << E_Nq2.back() << endl;
          cout << "      E[N2] = " << E_N2.back() << endl;
     }

     if(guardar_estatisticas)
          GeraDadosGrafico(rodada, E_N1.back(), E_N2.back(), E_Nq1.back(),
E_Nq2.back(), E_W1.back(), E_W2.back(), E_T1.back(), E_T2.back(),
V_W1.back(),V_W2.back(), nome_pasta);
}

/*
Limpa os resultados da rodada guardados pelo simulador.
Deve ser usada entre as rodadas para garantir que só consideramos
em cada rodada dados que tem origem em clientes daquela rodada
*/
void Simulador::LimpaResultadosParciais()
{

     acumulaW1 = 0.0;
     acumulaT1 = 0.0;
     acumulaW2 = 0.0;
     acumulaT2 = 0.0;

     Nq1_parcial = 0.0;
     Nq2_parcial = 0.0;
     N1_parcial = 0.0;
     N2_parcial = 0.0;

     acumula_quadradoW1 = 0.0;
     acumula_quadradoW2 = 0.0;

}

void Simulador::Setup(int semente, bool deterministico)
{


     total_clientes_servidos_uma_vez =0;
    total_clientes_servidos_duas_vezes = 0;

     gerador = GeradorTempoExponencial::GetInstancia();
```

```
	/*
		Se definimos alguma semente, então usar ela no gerador
	*/
	if (semente > 0)
		gerador->DefinirSemente(semente);


	servidor_vazio = true;
	id_proximo_cliente = 0;
	tempo_atual = 0.0;


	/*
		Limpa a heap de eventos, as filas e os dados
	*/
	while(!filaEventos.empty()) filaEventos.pop();
	while(!fila1.empty()) fila1.pop();
	fila2.clear();
	while(!E_N1.empty()) E_N1.pop_back();
	while(!E_N2.empty()) E_N2.pop_back();
	while(!E_Nq1.empty()) E_Nq1.pop_back();
	while(!E_Nq2.empty()) E_Nq2.pop_back();
	while(!E_W1.empty()) E_W1.pop_back();
	while(!E_W2.empty()) E_W2.pop_back();
	while(!E_T1.empty()) E_T1.pop_back();
	while(!E_T2.empty()) E_T2.pop_back();
	while(!E_T1.empty()) E_T1.pop_back();
	while(!E_T2.empty()) E_T2.pop_back();
	while(!V_W1.empty()) V_W1.pop_back();
	while(!V_W2.empty()) V_W2.pop_back();

	/*
		Coloca o primeiro evendo na "heap" de eventos
	*/
	filaEventos.push(Evento(nova_chegada,gerador-
>GeraTempoExponencial(taxa_chegada, deterministico)));


}

void Simulador::GeraDadosGrafico(int rodada, double pN1, double pN2, double
pNq1, double pNq2, double pW1, double pW2, double pT1, double pT2, double
pV_W1, double pV_W2, string nome_pasta)
{
	ofstream outputFile;
	string temp_pasta = nome_pasta;

	nome_pasta.append("/N1.txt");
	char *arquivo = (char*)nome_pasta.c_str();
	outputFile.open(arquivo, ios::app);
	outputFile << rodada <<"\t"<< pN1 << endl;
	outputFile.close();

	nome_pasta = temp_pasta;
	nome_pasta.append("/N2.txt");
	arquivo = (char*)nome_pasta.c_str();
	outputFile.open(arquivo, ios::app);
	outputFile << rodada <<"\t"<< pN2 << endl;
	outputFile.close();

	nome_pasta = temp_pasta;
	nome_pasta.append("/Nq1.txt");
	arquivo = (char*)nome_pasta.c_str();
```

```
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pNq1 << endl;
        outputFile.close();

        nome_pasta = temp_pasta;
        nome_pasta.append("/Nq2.txt");
        arquivo = (char*)nome_pasta.c_str();
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pNq2 << endl;
        outputFile.close();

        nome_pasta = temp_pasta;
        nome_pasta.append("/W1.txt");
        arquivo = (char*)nome_pasta.c_str();
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pW1 << endl;
        outputFile.close();

        nome_pasta = temp_pasta;
        nome_pasta.append("/W2.txt");
        arquivo = (char*)nome_pasta.c_str();
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pW2 << endl;
        outputFile.close();

        nome_pasta = temp_pasta;
        nome_pasta.append("/T1.txt");
        arquivo = (char*)nome_pasta.c_str();
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pT1 << endl;
        outputFile.close();

        nome_pasta = temp_pasta;
        nome_pasta.append("/T2.txt");
        arquivo = (char*)nome_pasta.c_str();
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pT2 << endl;
        outputFile.close();

        nome_pasta = temp_pasta;
        nome_pasta.append("/V_W1.txt");
        arquivo = (char*)nome_pasta.c_str();
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pV_W1 << endl;
        outputFile.close();

        nome_pasta = temp_pasta;
        nome_pasta.append("/V_W2.txt");
        arquivo = (char*)nome_pasta.c_str();
        outputFile.open(arquivo, ios::app);
        outputFile << rodada <<"\t"<< pV_W2 << endl;
        outputFile.close();
}

/*
Essa função é necessária para remover eventos de término de serviço da fila de
eventos
quando rodamos o programa no modo "dois por vez", pois nesse modo não temos
garantia
que o serviço a ser destruído está no topo da fila de eventos, mas sabemos que
é o único
evento do tipo "termino de servico" na fila.
Essa funçao varre a fila (remove o primeiro elemento e guarda numa fila
temporária)
```

```
até encontrar o evento de término de serviço, então deleta esse evento e
insere ordenadamente
os eventos removidos até ali, exceto o que realmente deveria ter sido removido
*/

Evento Simulador::RemoveTerminoServico(){
     priority_queue<Evento, vector<Evento>, greater<Evento> > filaTemp;
     Evento topo = filaEventos.top();
     while(topo.GetTipo() != termino_de_servico){
        cout << "oi" << endl;
        filaTemp.push(filaEventos.top());
        filaEventos.pop();
        topo = filaEventos.top();
     }
     filaEventos.pop();
     while(!filaTemp.empty()){
        filaEventos.push(filaTemp.top());
        filaTemp.pop();
     }
     return topo;
}


vector<double> Simulador::GetE_Nq1()
{
     return E_Nq1;
}
vector<double> Simulador::GetE_Nq2()
{
     return E_Nq2;
}
vector<double> Simulador::GetE_N1()
{
     return E_N1;
}
vector<double> Simulador::GetE_N2()
{
     return E_N2;
}
vector<double> Simulador::GetE_W1()
{
     return E_W1;
}
vector<double> Simulador::GetE_T1()
{
     return E_T1;
}
vector<double> Simulador::GetE_W2()
{
     return E_W2;
}
vector<double> Simulador::GetE_T2()
{
     return E_T2;
}
vector<double> Simulador::GetV_W1()
{
     return V_W1;
}
vector<double> Simulador::GetV_W2()
{
     return V_W2;
}
```

Simulador.h

```
#ifndef SIMULADOR_H_
#define SIMULADOR_H_
#include <vector>
#include <queue>
#include <deque>
#include <stdio.h>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include "Cliente.h"
#include "Evento.h"
#include "GeradorTempoExponencial.h"

using namespace std;

class Simulador{
    private:
          /*
               Fila de prioridade para podermos ordenar os eventos a partir de
seu tempo de acontecimento
          */
          priority_queue<Evento, vector<Evento>, greater<Evento> > filaEventos;
          queue<Cliente> fila1;
          /*
               Foi utillizado deque para podermos mover um cliente de volta
para a frente da fila 2 sem dificuldades
          */
          deque<Cliente> fila2;

          GeradorTempoExponencial* gerador;

          double tempo_atual;
          Cliente cliente_em_servico;
          bool servidor_vazio;
          int id_proximo_cliente;
          double taxa_chegada;
          double taxa_servico;
          double cliente_W1;
          double cliente_W2;

          int total_clientes_servidos_uma_vez;
         int total_clientes_servidos_duas_vezes;

          /*
               Variaveis que auxiliarão no cálculo do numero medio de clientes
em cada regiao do sistema,
               acumulando o produto N * tempo, onde N é o número de pessoas em
cada região do sistema.
               Nq1 = Número de pessoas na fila de espera para receber o
primeiro serviço
               Nq2 = Número de pessoas na fila de espera para receber o segundo
serviço
               N1 = Número total de pessoas no sistema que ainda não receberam
o primeiro serviço completo
               N2 = Número total de pessoas no sistema que já foram servidas
uma vez, mas ainda não completaram o segundo serviço
               OBS em um dado instante: Ni = Nqi + 1, se houver um cliente
vindo da fila i no servidor
                                   Ni = Nqi, caso contrário
```

```
        */
        double Nq1_parcial;
          double Nq2_parcial;
          double N1_parcial;
          double N2_parcial;

          /*
               Acumuladores das informacoes dos clientes
          */
          double acumulaW1;
         double acumulaT1;
        double acumulaW2;
        double acumulaT2;
          double acumula_quadradoW1;
        double acumula_quadradoW2;

          vector<double> E_Nq1;
          vector<double> E_Nq2;
          vector<double> E_N1;
          vector<double> E_N2;
          vector<double> E_W1;
          vector<double> E_T1;
          vector<double> E_W2;
          vector<double> E_T2;
          vector<double> V_W1;
          vector<double> V_W2;

    public:
          Simulador();
          Simulador(double ptaxa_chegada, double ptaxa_servico, bool
deterministico, bool dois_por_vez, bool interrupcao_forcada, int semente);
          ~Simulador();
          void Roda(int num_total_clientes, int rodada_atual, bool debug, bool
deterministico, bool determina_transiente, bool dois_por_vez, string
nome_pasta, bool guardar_estatisticas,bool interrupcao_forcada, bool
mostrar_resultados);
          void CalculaResultados(int n, int servidos1, double t, int rodada,
bool mostrar_resultados, string nome_pasta, bool guardar_estatisticas);
          void LimpaResultadosParciais();
          void Setup(int semente, bool deterministico);
          void GeraDadosGrafico(int rodada, double pN1, double pN2, double
pNq1, double pNq2, double pW1, double pW2, double pT1, double pT2, double
pV_W1, double pV_W2, string nome_pasta);
          Evento RemoveTerminoServico();
          vector<double> GetE_Nq1();
          vector<double> GetE_Nq2();
          vector<double> GetE_N1();
          vector<double> GetE_N2();
          vector<double> GetE_W1();
          vector<double> GetE_T1();
          vector<double> GetE_W2();
          vector<double> GetE_T2();
          vector<double> GetV_W1();
          vector<double> GetV_W2();
};

#endif /*SIMULADOR_H_*/
```

GeradorTempoExponencial.cpp

```
#include "include\GeradorTempoExponencial.h"
#include <iostream>
#include <math.h>
#include <time.h>

//Inicialização da instancia
GeradorTempoExponencial* GeradorTempoExponencial::instancia = NULL;

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
     --------------------   -Contrutores & Destrutor----------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

GeradorTempoExponencial::GeradorTempoExponencial() : CRandomMersenne(0)
{
     int semente = (int) time(0); //Gera uma semente aleatória

     this->DefinirSemente(semente); //Define a semente do gerador como a
semente aleatória gerada acima
}

GeradorTempoExponencial::~GeradorTempoExponencial()
{
     //Destrutor
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
          --------------------   -Funções membro----------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*
     Retorna a instancia única do gerador de números aleatórios
*/
GeradorTempoExponencial* GeradorTempoExponencial::GetInstancia()
{
     if (instancia == NULL)
          instancia = new GeradorTempoExponencial();

     return instancia;
}

/*
     Define uma nova semente para o gerador de números aleatórios(resetando ele
antes)
*/
void GeradorTempoExponencial::DefinirSemente(int semente)
{
     CRandomMersenne::RandomInit(semente);
}
/*
     Função do Gerador de Números Aleatórios
*/
double GeradorTempoExponencial::Random()
{
     return CRandomMersenne::Random();
}

/*
```

```
     Se o modo deterministico for selecionado, gera a média de uma variável
exponencial com a taxa dada = 1/taxa
     Caso contrário, gera uma amostra de uma distribuição homogênea no
intervalo de 0 a 1
     e aplica a função inversa da CDF da exponencial para gerar uma amostra de
variável exponencial.
*/
double GeradorTempoExponencial::GeraTempoExponencial(double taxa, bool
deterministico)
{
    if (deterministico)
       return (1 / taxa);
     return (-1) * (log(1 - this->Random()) / taxa);
}
```

GeradorTempoExponencial.h

```
#include "Mersenne/randomc.h"

class GeradorTempoExponencial : CRandomMersenne
{
     private:
          static GeradorTempoExponencial *instancia;

     protected:
          GeradorTempoExponencial();
          ~GeradorTempoExponencial();

     public:
          static GeradorTempoExponencial* GetInstancia(); //Retorna a instancia
única do gerador de números aleatórios

          void DefinirSemente(int semente); //Muda a semente do gerador de
números aleatórios(resetando ele anteriormente)
          double Random(); //Função do Gerador de Números Aleatórios

          /*
        Se o modo deterministico for selecionado, gera a média de uma variável
exponencial com a taxa dada = 1/taxa
        Caso contrário, gera uma amostra de uma distribuição homogênea no
intervalo de 0 a 1
        e aplica a função inversa da CDF da exponencial para gerar uma amostra
de variável exponencial.
        */
          double GeraTempoExponencial(double taxa, bool deterministico);
};
```

Evento.cpp

```
#include "include\Evento.h"
#include <iostream>
#include <cstdlib>
#include <string>

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
          ---------------------   -Contrutores ----------------------
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

Evento::Evento(EnumTipo ptipo, double ptempo_acontecimento)
{
     this->tipo = ptipo;
     this->tempo_acontecimento = ptempo_acontecimento;
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
          ---------------------   -Funções membro----------------------
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

EnumTipo Evento::GetTipo()
{
     return tipo;
}

double Evento::GetTempoAcontecimento() const
{
     return tempo_acontecimento;
}
/*
     Método usado para podermos imprimir o nome dos eventos na tela, no modo
debug.
*/
string Evento::GetNome()
{
       if (tipo == 0)
          return "Nova Chegada";
       if (tipo == 1)
          return "Chegada Artificial";
       return "Termino de Servico";
}
```

Evento.h

```
#ifndef EVENTO_H_
#define EVENTO_H_

#include <stdio.h>
#include <cstdlib>
#include <iostream>

using namespace std;
enum EnumTipo{nova_chegada,chegada_artificial,termino_de_servico};

class Evento{
     private:
          EnumTipo tipo;
          double tempo_acontecimento;

     public:
          Evento(EnumTipo ptipo, double ptempo_acontecimento);
          EnumTipo GetTipo();
          string GetNome();
          double GetTempoAcontecimento() const;
          inline bool operator > (const Evento evento2) const { return (this-
>GetTempoAcontecimento() > evento2.GetTempoAcontecimento()); }
};

#endif /*EVENTO_H_*/
```

Cliente.cpp

```
#include "include\Cliente.h"
#include <iostream>
#include <cstdlib>
#include <string>

using namespace std;

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
          --------------------   -Contrutores ---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
Cliente::Cliente()
{
     this->id = 0;
     this->fila = 0;
     this->interrompido = 0;
}

Cliente::Cliente(int pid,double instante_chegada, int pfila, int rodada_atual)
{
     this->id = pid;
     this->fila = pfila;
     this->interrompido = false;
     this->instante_chegada1 = instante_chegada;
     this->direto_ao_servidor = false;
     this->rodada_pertencente = rodada_atual;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
          --------------------   -Funções membro---------------------
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

int Cliente::GetFila()
{
     return fila;
}

int Cliente::SetFila(int pfila)
{
     this->fila = pfila;
}

double Cliente::GetTempoRestante()
{
     return tempo_restante;
}

double Cliente::SetTempoRestante(double ptempo_restante)
{
     this->tempo_restante = ptempo_restante;
}

bool Cliente::VerificaInterrompido()
{
     return interrompido;
}

void Cliente::Interromper()
```

```
{
     this->interrompido = true;
}

void Cliente::SetInstanteChegada2(double t)
{
      this->instante_chegada2=t;
}

void Cliente::SetDuracaoPrimeiroServico(double duracao)
{
      this->duracao_primeiro_servico=duracao;
}

void Cliente::SetDuracaoSegundoServico(double duracao)
{
      this->duracao_segundo_servico=duracao;
}

void Cliente::SetInstanteSaida(double t)
{
      this->instante_saida=t;
}

int Cliente::GetID()
{
     return id;
}

void Cliente::SetDiretoAoServidor(bool pbool)
{
     this->direto_ao_servidor = pbool;
}

bool Cliente::GetDiretoAoServidor()
{
     return direto_ao_servidor;
}

int Cliente::GetRodadaPertencente()
{
     return rodada_pertencente;
}

double Cliente::W1()
{
        return instante_chegada2 - instante_chegada1 -
duracao_primeiro_servico;
}

double Cliente::T1()
{
        return instante_chegada2 - instante_chegada1;
}

double Cliente::W2()
{
        return instante_saida - instante_chegada2 - duracao_segundo_servico;
}

double Cliente::T2()
{
        return instante_saida - instante_chegada2;
}
```

Cliente.h

```
#include <stdio.h>
#include <cstdlib>
#include <iostream>

using namespace std;


class Cliente{
     private:
          int id;
          int fila;
          bool interrompido;
          double tempo_restante;
          bool direto_ao_servidor;
          //Informa a rodada que o cliente pertence, utilizado como modo de
"coloração"
          int rodada_pertencente;

          //Instante de chegada do cliente no sistema, na fila 1
          double instante_chegada1;
          //Instante em que o cliente chega na fila 2. É o mesmo instante em
que acaba seu primeiro serviço
          double instante_chegada2;
          //Instante em que acaba o segundo serviço do cliente. É o instante em
que ele é removido do sistema
          double instante_saida;
          //Duração do primeiro serviço
          double duracao_primeiro_servico;
          //Duração do segundo serviço
          double duracao_segundo_servico;

     public:
          Cliente();
          Cliente(int pid,double instante_chegada,int pfila, int rodada_atual);
          int GetID();
          int GetFila();
          int SetFila(int pfila);
          double GetTempoRestante();
          double SetTempoRestante(double ptempo_restante);
          bool VerificaInterrompido();
          void Interromper();
          void SetDiretoAoServidor(bool pbool);
          bool GetDiretoAoServidor();
          int GetRodadaPertencente();

          void SetInstanteChegada2(double t);
          void SetInstanteSaida(double t);
          void SetDuracaoPrimeiroServico(double duracao);
          void SetDuracaoSegundoServico(double duracao);

          double W1();
          double T1();
          double W2();
          double T2();
};
```

Main.cpp

```
#include "include\Simulador.h"
#include "include\Config.h"
#include <iostream>
#include <math.h>
#include <cstdlib>
#include <fstream>
#include <sstream>
#ifdef WIN32
    #include <direct.h>
    #define MKDIR(a) _mkdir(a)
#else
     #include <sys/stat.h>
     #define MKDIR(a) mkdir(a, 0777)
#endif
#define CONF95  1.96


using namespace std;

void CalculaIntervaloConfianca(int num_rodadas, vector<double> E_N1,
vector<double> E_N2, vector<double> E_Nq1, vector<double> E_Nq2,
vector<double> E_W1, vector<double> E_W2, vector<double> E_T1, vector<double>
E_T2, vector<double> V_W1, vector<double> V_W2);

int main(void)
{

     bool repeat = true;
     char repeat_temp;
     ofstream arquivo_entradas;


     Config<int> *num_rodadas = new Config<int>("num_rodadas");
    Config<int> *num_clientes = new Config<int>("num_clientes");
    Config<double> *taxa_chegada = new Config<double>("taxa_chegada");
    Config<bool> *deterministico = new Config<bool>("deterministico");
    Config<bool> *debug = new Config<bool>("debug");
     Config<bool> *mostrar_resultados = new Config<bool>("mostrar_resultados");
    Config<bool> *determina_transiente = new
Config<bool>("determina_transiente");
     Config<bool> *dois_por_vez = new Config<bool>("dois_por_vez");
     Config<bool> *guardar_estatisticas = new
Config<bool>("guardar_estatisticas");
     Config<bool> *interrupcao_forcada = new
Config<bool>("interrupcao_forcada");
     Config<int> *semente = new Config<int>("semente");
     Config<int> *tamanho_transiente = new Config<int>("tamanho_transiente");

     string nome_arquivo,valor,buffer,nome_pasta;
    ifstream entrada;
    size_t pos, fim,temp;

     cout<<"****************** Trabalho de Simulacao de AD
2011/1*******************"<< endl;
     cout<<"Simulador desenvolvido pelos alunos:"<<endl;
     cout<<"      Luiz Filipe de Sa Estrella         DRE: 107390627" << endl;
```

```
     cout<<"      Fernando de Mesentier Silva        DRE: 107390520" << endl;
     cout<<"      Rodrigo de Moura Canaan            DRE: 107362200" << endl;
     cout<<"      Vinicius Jose Serva Pereira        DRE: 106050355" << endl;
     cout<<"Como projeto final da disciplina Avaliacao e Desempenho," << endl;
     cout<<"Do curso Bacharelado em Ciencia da Computacao da UFRJ,"<<endl;
     cout<<"Sob orientacao do prof. Paulo Henrique de Aguiar Rodrigues"<<endl;
     cout<<"Durante o primeiro semestre de 2011."<<endl;

    while (repeat)
    {
         while (1)
         {
              cout<<endl<<"Entre com o nome do arquivo que contem as entradas:
";
              cin>>nome_arquivo;
              entrada.clear();
              entrada.open(nome_arquivo.c_str(), ifstream::in);
              if (!entrada.fail())
                   break;
              else
              {
                   cout<<"Erro abrindo arquivo(O nome do arquivo esta
correto?)"<<endl;

              }

         }

         while(std::getline(entrada, buffer))
         {
              num_rodadas->procurar_e_adicionar(buffer);
              num_clientes->procurar_e_adicionar(buffer);
              taxa_chegada->procurar_e_adicionar(buffer);
              deterministico->procurar_e_adicionar(buffer);
              debug->procurar_e_adicionar(buffer);
              mostrar_resultados->procurar_e_adicionar(buffer);
              determina_transiente->procurar_e_adicionar(buffer);
              dois_por_vez->procurar_e_adicionar(buffer);
              guardar_estatisticas->procurar_e_adicionar(buffer);
              interrupcao_forcada->procurar_e_adicionar(buffer);
              semente->procurar_e_adicionar(buffer);
              tamanho_transiente->procurar_e_adicionar(buffer);

         }

         num_rodadas->verificar_valor();
         num_clientes->verificar_valor();
         taxa_chegada->verificar_valor();
         deterministico->verificar_valor();
         debug->verificar_valor();
         mostrar_resultados->verificar_valor();
         determina_transiente->verificar_valor();
         dois_por_vez->verificar_valor();
         guardar_estatisticas->verificar_valor();
         interrupcao_forcada->verificar_valor();
         semente->verificar_valor();
         tamanho_transiente->verificar_valor();

         if( !num_rodadas->erro() and  !num_clientes->erro() and
!taxa_chegada->erro() and !tamanho_transiente->erro())
         {
              cout<< endl << num_rodadas->nome()<< ": "<<num_rodadas-
>num()<<endl;
              cout<<num_clientes->nome()<< ": "<<num_clientes->num()<<endl;
```

```
                cout<<taxa_chegada->nome()<< ": "<<taxa_chegada->num()<<endl;
                cout<<deterministico->nome()<< ": "<<deterministico-
>num()<<endl;
                cout<<debug->nome()<< ": "<<debug->num()<<endl;
                cout<<mostrar_resultados->nome()<< ": "<<mostrar_resultados-
>num()<<endl;
                cout<<determina_transiente->nome()<< ": "<<determina_transiente-
>num()<<endl;
                cout<<dois_por_vez->nome()<< ": "<<dois_por_vez->num()<<endl;
                cout<<guardar_estatisticas->nome()<< ": "<<guardar_estatisticas-
>num()<<endl;
                cout<<interrupcao_forcada->nome()<< ": "<<interrupcao_forcada-
>num()<<endl;
                cout<<tamanho_transiente->nome()<< ": "<<tamanho_transiente-
>num()<<endl;


                if(!semente->erro())
                     cout<<semente->nome()<< ": "<<semente->num()<<endl;

                cout << endl << "Os valores estao corretos? (S/N)" << endl;
                cin >> repeat_temp;
                if(repeat_temp == 'S' or repeat_temp == 's')
                {
                      cout<<endl<<"A simulacao esta sendo executada,
aguarde..."<<endl;
                      repeat = false;
                }
                else
                {
                     entrada.close();
                      repeat = true;

                }
          }
          else
          {
                entrada.close();
          }
     }

     /*
          No caso de guardarmos as estatisticas é gerado uma pasta com o nome
do arquivo entrada e
          criamos uma arquivo chamado "Entrada_usadas.txt" com todas as
entradas usadas. Para analises posteriores.
     */
     if(guardar_estatisticas->num())
     {
          nome_pasta = nome_arquivo.erase(nome_arquivo.size()-3,3);
          MKDIR(nome_pasta.c_str());

          string temp_pasta = nome_pasta;

          temp_pasta.append("/Entradas_usadas.ini");
          char *arquivo = (char*)temp_pasta.c_str();
          arquivo_entradas.open(arquivo, ios::app);

          arquivo_entradas<<num_rodadas->nome()<< " = "<<num_rodadas->num()<<
";" <<endl;
          arquivo_entradas<<num_clientes->nome()<< " = "<<num_clientes->num()<<
";" <<endl;
          arquivo_entradas<<taxa_chegada->nome()<< " = "<<taxa_chegada->num()<<
";" <<endl;
```

```
          arquivo_entradas<<deterministico->nome()<< " = "<<deterministico-
>num()<< ";" <<endl;
          arquivo_entradas<<debug->nome()<< " =  "<<debug->num()<< ";" <<endl;
          arquivo_entradas<<mostrar_resultados->nome()<< " =
"<<mostrar_resultados->num()<< ";" <<endl;
          arquivo_entradas<<determina_transiente->nome()<< " =
"<<determina_transiente->num()<< ";" <<endl;
          arquivo_entradas<<dois_por_vez->nome()<< " = "<<dois_por_vez->num()<<
";" <<endl;
          arquivo_entradas<<guardar_estatisticas->nome()<< " =
"<<guardar_estatisticas->num()<< ";" <<endl;
          arquivo_entradas<<interrupcao_forcada->nome()<< " =
"<<interrupcao_forcada->num()<< ";" <<endl;
          arquivo_entradas<<tamanho_transiente->nome()<< " =
"<<tamanho_transiente->num()<< ";" <<endl;

          if(!semente->erro())
                         arquivo_entradas<<semente->nome()<< " = "<<semente-
>num()<< ";" <<endl;

          arquivo_entradas.close();
     }

     Simulador simula;

     if(!semente->erro())
          simula = Simulador(taxa_chegada->num(),1,deterministico->num(),
dois_por_vez->num(), interrupcao_forcada->num(),semente->num());
     else
          simula = Simulador(taxa_chegada->num(),1,deterministico->num(),
dois_por_vez->num(), interrupcao_forcada->num(), -1);

     for( int i = 0 ; i < num_rodadas->num()+1 ; i++)
     {
          if( i == 0 )
               simula.Roda(tamanho_transiente->num(),i, debug->num(),
deterministico->num(), determina_transiente->num(), dois_por_vez->num(),
nome_pasta, guardar_estatisticas, interrupcao_forcada->num(),
mostrar_resultados->num());
          else
               simula.Roda(num_clientes->num(),i, debug->num(), deterministico-
>num(), determina_transiente->num(), dois_por_vez->num(), nome_pasta,
guardar_estatisticas, interrupcao_forcada->num(), mostrar_resultados->num());

          if(!determina_transiente->num())
               simula.LimpaResultadosParciais();
     }

    CalculaIntervaloConfianca(num_rodadas->num(), simula.GetE_N1(),
simula.GetE_N2(),  simula.GetE_Nq1(),  simula.GetE_Nq2(),  simula.GetE_W1(),
simula.GetE_W2(),  simula.GetE_T1(),  simula.GetE_T2(), simula.GetV_W1(),
simula.GetV_W2());
     entrada.close();

     system("pause");
     return 0;
}

/*
     Método responsável pelo cálculo dos intervalos de confiança.
*/
void CalculaIntervaloConfianca(int num_rodadas, vector<double> E_N1,
vector<double> E_N2, vector<double> E_Nq1, vector<double> E_Nq2,
```

```
vector<double> E_W1, vector<double> E_W2, vector<double> E_T1, vector<double>
E_T2, vector<double> V_W1, vector<double> V_W2)
{
     double estimador_media, estimador_var, intervalo;


     ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     ////////////////////////////////--------------Cálculo do Intervalo de W1----
----------//////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_W1[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)E_W1[i]) - estimador_media) *
(((double)E_W1[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("E[W1]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);

     ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     ////////////////////////////////--------------Cálculo do Intervalo de T1----
----------//////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_T1[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)E_T1[i]) - estimador_media) *
(((double)E_T1[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));
```

```
     printf("E[T1]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);




////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////
     ////////////////////////////////--------------Cálculo do Intervalo de V_W1--
------------/////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)V_W1[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)V_W1[i]) - estimador_media) *
(((double)V_W1[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("V(W1)\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);

     ////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////
     ////////////////////////////////--------------Cálculo do Intervalo de Nq1---
-----------/////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_Nq1[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
```

```
        estimador_var += (((double)E_Nq1[i]) - estimador_media) *
(((double)E_Nq1[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("E[Nq1]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);

     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     ////////////////////////////////-------------Cálculo do Intervalo de N1----
----------//////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_N1[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)E_N1[i]) - estimador_media) *
(((double)E_N1[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("E[N1]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);

     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     ////////////////////////////////-------------Cálculo do Intervalo de W2----
----------//////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_W2[i];
    }
    estimador_media /= (double)num_rodadas;
```

```
    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)E_W2[i]) - estimador_media) *
(((double)E_W2[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("E[W2]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);



     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     ////////////////////////////////-------------Cálculo do Intervalo de T2----
----------//////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_T2[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)E_T2[i]) - estimador_media) *
(((double)E_T2[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("E[T2]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);



     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     ////////////////////////////////-------------Cálculo do Intervalo de V_W2--
------------//////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;
```

```
    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)V_W2[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)V_W2[i]) - estimador_media) *
(((double)V_W2[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("V(W2)\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);


     ////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////
     ////////////////////////////////-------------Cálculo do Intervalo de Nq2---
----------//////////////////////////////////////
     ////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_Nq2[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)E_Nq2[i]) - estimador_media) *
(((double)E_Nq2[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("E[Nq2]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);

     ////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////
     ////////////////////////////////-------------Cálculo do Intervalo de N2----
----------//////////////////////////////////////
```

```
     ////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
     estimador_media = 0.0;
     estimador_var = 0.0;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_media += (double)E_N2[i];
    }
    estimador_media /= (double)num_rodadas;

    for(unsigned int i = 0; i < num_rodadas ; i++)
    {
        estimador_var += (((double)E_N2[i]) - estimador_media) *
(((double)E_N2[i]) - estimador_media);
    }
    estimador_var /= (double)(num_rodadas - 1);

    intervalo = CONF95 * (sqrt(estimador_var) / sqrt(num_rodadas));

     printf("E[N2]\n", (estimador_media - intervalo < 0) ? 0 : estimador_media
- intervalo);
     printf("     Intervalo = %lf ate %lf\n",(estimador_media - intervalo < 0)
? 0 : estimador_media - intervalo, estimador_media + intervalo);
     printf("     Media do Intervalo: %lf \n", estimador_media);
     printf("     Tamanho: %lf \n",  2.0 * intervalo);
     printf("     Tamanho em relacao a Media: %lf%% \n", (200.0 * intervalo) /
estimador_media);

}
```

mersene.h

(código não desenvolvido pelo grupo)

```
/*************************   mersenne.cpp
**********************************
* Author:        Agner Fog
* Date created:  2001
* Last modified: 2008-11-16
* Project:       randomc.h
* Platform:      Any C++
* Description:
* Random Number generator of type 'Mersenne Twister'
*
* This random number generator is described in the article by
* M. Matsumoto & T. Nishimura, in:
* ACM Transactions on Modeling and Computer Simulation,
* vol. 8, no. 1, 1998, pp. 3-30.
* Details on the initialization scheme can be found at
* http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
*
* Further documentation:
* The file ran-instructions.pdf contains further documentation and
* instructions.
*
* Copyright 2001-2008 by Agner Fog.
* GNU General Public License http://www.gnu.org/licenses/gpl.html
******************************************************************************
*/

#include "randomc.h"

void CRandomMersenne::Init0(int seed) {
   // Seed generator
   const uint32_t factor = 1812433253UL;
   mt[0]= seed;
   for (mti=1; mti < MERS_N; mti++) {
      mt[mti] = (factor * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
   }
}

void CRandomMersenne::RandomInit(int seed) {
   // Initialize and seed
   Init0(seed);

   // Randomize some more
   for (int i = 0; i < 37; i++) BRandom();
}


void CRandomMersenne::RandomInitByArray(int const seeds[], int NumSeeds) {
   // Seed by more than 32 bits
   int i, j, k;

   // Initialize
   Init0(19650218);

   if (NumSeeds <= 0) return;

   // Randomize mt[] using whole seeds[] array
```

```
   i = 1;  j = 0;
   k = (MERS_N > NumSeeds ? MERS_N : NumSeeds);
   for (; k; k--) {
      mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL)) +
(uint32_t)seeds[j] + j;
      i++; j++;
      if (i >= MERS_N) {mt[0] = mt[MERS_N-1]; i=1;}
      if (j >= NumSeeds) j=0;}
   for (k = MERS_N-1; k; k--) {
      mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL)) - i;
      if (++i >= MERS_N) {mt[0] = mt[MERS_N-1]; i=1;}}
   mt[0] = 0x80000000UL;  // MSB is 1; assuring non-zero initial array

   // Randomize some more
   mti = 0;
   for (int i = 0; i <= MERS_N; i++) BRandom();
}


uint32_t CRandomMersenne::BRandom() {
   // Generate 32 random bits
   uint32_t y;

   if (mti >= MERS_N) {
      // Generate MERS_N words at one time
      const uint32_t LOWER_MASK = (1LU << MERS_R) - 1;       // Lower MERS_R
bits
      const uint32_t UPPER_MASK = 0xFFFFFFFF << MERS_R;      // Upper (32 -
MERS_R) bits
      static const uint32_t mag01[2] = {0, MERS_A};

      int kk;
      for (kk=0; kk < MERS_N-MERS_M; kk++) {
         y = (mt[kk] & UPPER_MASK) | (mt[kk+1] & LOWER_MASK);
         mt[kk] = mt[kk+MERS_M] ^ (y >> 1) ^ mag01[y & 1];}

      for (; kk < MERS_N-1; kk++) {
         y = (mt[kk] & UPPER_MASK) | (mt[kk+1] & LOWER_MASK);
         mt[kk] = mt[kk+(MERS_M-MERS_N)] ^ (y >> 1) ^ mag01[y & 1];}

      y = (mt[MERS_N-1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
      mt[MERS_N-1] = mt[MERS_M-1] ^ (y >> 1) ^ mag01[y & 1];
      mti = 0;
   }
   y = mt[mti++];

   // Tempering (May be omitted):
   y ^=  y >> MERS_U;
   y ^= (y << MERS_S) & MERS_B;
   y ^= (y << MERS_T) & MERS_C;
   y ^=  y >> MERS_L;

   return y;
}


double CRandomMersenne::Random() {
   // Output random float number in the interval 0 <= x < 1
   // Multiply by 2^(-32)
   return (double)BRandom() * (1./(65536.*65536.));
}
int CRandomMersenne::IRandom(int min, int max) {
   // Output random integer in the interval min <= x <= max
   // Relative error on frequencies < 2^-32
```

```
   if (max <= min) {
      if (max == min) return min; else return 0x80000000;
   }
   // Multiply interval with random and truncate
   int r = int((double)(uint32_t)(max - min + 1) * Random() + min);
   if (r > max) r = max;
   return r;
}

int CRandomMersenne::IRandomX(int min, int max) {
   // Output random integer in the interval min <= x <= max
   // Each output value has exactly the same probability.
   // This is obtained by rejecting certain bit values so that the number
   // of possible bit values is divisible by the interval length
   if (max <= min) {
      if (max == min) return min; else return 0x80000000;
   }
#ifdef  INT64_SUPPORTED
   // 64 bit integers available. Use multiply and shift method
   uint32_t interval;                     // Length of interval
   uint64_t longran;                      // Random bits * interval
   uint32_t iran;                         // Longran / 2^32
   uint32_t remainder;                    // Longran % 2^32

   interval = uint32_t(max - min + 1);
   if (interval != LastInterval) {
      // Interval length has changed. Must calculate rejection limit
      // Reject when remainder >= 2^32 / interval * interval
      // RLimit will be 0 if interval is a power of 2. No rejection then
      RLimit = uint32_t(((uint64_t)1 << 32) / interval) * interval - 1;
      LastInterval = interval;
   }
   do { // Rejection loop
      longran  = (uint64_t)BRandom() * interval;
      iran = (uint32_t)(longran >> 32);
      remainder = (uint32_t)longran;
   } while (remainder > RLimit);
   // Convert back to signed and return result
   return (int32_t)iran + min;
#else
   // 64 bit integers not available. Use modulo method
   uint32_t interval;                     // Length of interval
   uint32_t bran;                         // Random bits
   uint32_t iran;                         // bran / interval
   uint32_t remainder;                    // bran % interval

   interval = uint32_t(max - min + 1);
   if (interval != LastInterval) {
      // Interval length has changed. Must calculate rejection limit
      // Reject when iran = 2^32 / interval
      // We can't make 2^32 so we use 2^32-1 and correct afterwards
      RLimit = (uint32_t)0xFFFFFFFF / interval;
      if ((uint32_t)0xFFFFFFFF % interval == interval - 1) RLimit++;
   }
   do { // Rejection loop
      bran = BRandom();
      iran = bran / interval;
      remainder = bran % interval;
   } while (iran >= RLimit);
   // Convert back to signed and return result
   return (int32_t)remainder + min;

#endif
}
```

random.h

(código não desenvolvido pelo grupo)

```
/*****************************   randomc.h   *****************
* Author:        Agner Fog
* Date created:  1997
* Last modified: 2008-11-16
* Project:       randomc.h
* Source URL:    www.agner.org/random
*
* Description:
* This header file contains class declarations and other definitions for the
* randomc class library of uniform random number generators in C++ language.
*
* Overview of classes:
* ====================
*
* class CRandomMersenne:
* Random number generator of type Mersenne twister.
* Source file mersenne.cpp
*
* class CRandomMother:
* Random number generator of type Mother-of-All (Multiply with carry).
* Source file mother.cpp
*
* class CRandomSFMT:
* Random number generator of type SIMD-oriented Fast Mersenne Twister.
* The class definition is not included here because it is not
* portable to all platforms. See sfmt.h and sfmt.cpp for details.
*
* Member functions (methods):
* ===========================
*
* All these classes have identical member functions:
*
* Constructor(int seed):
* The seed can be any integer. The time may be used as seed.
* Executing a program twice with the same seed will give the same sequence
* of random numbers. A different seed will give a different sequence.
*
* void RandomInit(int seed);
* Re-initializes the random number generator with a new seed.
*
* void RandomInitByArray(int const seeds[], int NumSeeds);
* In CRandomMersenne and CRandomSFMT only: Use this function if you want
* to initialize with a seed with more than 32 bits. All bits in the seeds[]
* array will influence the sequence of random numbers generated. NumSeeds
* is the number of entries in the seeds[] array.
*
* double Random();
* Gives a floating point random number in the interval 0 <= x < 1.
* The resolution is 32 bits in CRandomMother and CRandomMersenne, and
* 52 bits in CRandomSFMT.
*
* int IRandom(int min, int max);
* Gives an integer random number in the interval min <= x <= max.
* (max-min < MAXINT).
* The precision is 2^-32 (defined as the difference in frequency between
* possible output values). The frequencies are exact if max-min+1 is a
* power of 2.
*
* int IRandomX(int min, int max);
```

```
* Same as IRandom, but exact. In CRandomMersenne and CRandomSFMT only.
* The frequencies of all output values are exactly the same for an
* infinitely long sequence. (Only relevant for extremely long sequences).
*
* uint32_t BRandom();
* Gives 32 random bits.
*
*
* Example:
* ========
* The file EX-RAN.CPP contains an example of how to generate random numbers.
*
*
* Library version:
* ================
* Optimized versions of these random number generators are provided as
function
* libraries in randoma.zip. These function libraries are coded in assembly
* language and support only x86 platforms, including 32-bit and 64-bit
* Windows, Linux, BSD, Mac OS-X (Intel based). Use randoma.h from randoma.zip
*
*
* Non-uniform random number generators:
* =====================================
* Random number generators with various non-uniform distributions are
* available in stocc.zip (www.agner.org/random).
*
*
* Further documentation:
* ======================
* The file ran-instructions.pdf contains further documentation and
* instructions for these random number generators.
*
* Copyright 1997-2008 by Agner Fog.
* GNU General Public License http://www.gnu.org/licenses/gpl.html
******************************************************************************
*/

#ifndef RANDOMC_H
#define RANDOMC_H

// Define integer types with known size: int32_t, uint32_t, int64_t, uint64_t.
// If this doesn't work then insert compiler-specific definitions here:
#if defined(__GNUC__)
  // Compilers supporting C99 or C++0x have inttypes.h defining these integer
types
  #include <inttypes.h>
  #define INT64_SUPPORTED // Remove this if the compiler doesn't support 64-
bit integers
#elif defined(_WIN16) || defined(__MSDOS__) || defined(_MSDOS)
   // 16 bit systems use long int for 32 bit integer
  typedef   signed long int int32_t;
  typedef unsigned long int uint32_t;
#elif defined(_MSC_VER)
  // Microsoft have their own definition
  typedef   signed __int32  int32_t;
  typedef unsigned __int32 uint32_t;
  typedef   signed __int64  int64_t;
  typedef unsigned __int64 uint64_t;
  #define INT64_SUPPORTED // Remove this if the compiler doesn't support 64-
bit integers
#else
  // This works with most compilers
  typedef signed int          int32_t;
```

```
  typedef unsigned int       uint32_t;
  typedef long long           int64_t;
  typedef unsigned long long uint64_t;
  #define INT64_SUPPORTED // Remove this if the compiler doesn't support 64-
bit integers
#endif


/*****************************************************************************
System-specific user interface functions
*****************************************************************************/

void EndOfProgram(void);               // System-specific exit code
(userintf.cpp)

void FatalError(const char *ErrorText);// System-specific error reporting
(userintf.cpp)

#if defined(__cplusplus)               // class definitions only in C++
/*****************************************************************************
Define random number generator classes
*****************************************************************************/

class CRandomMersenne {                // Encapsulate random number generator
// Choose which version of Mersenne Twister you want:
#if 0
// Define constants for type MT11213A:
#define MERS_N   351
#define MERS_M   175
#define MERS_R   19
#define MERS_U   11
#define MERS_S   7
#define MERS_T   15
#define MERS_L   17
#define MERS_A   0xE4BD75F5
#define MERS_B   0x655E5280
#define MERS_C   0xFFD58000
#else
// or constants for type MT19937:
#define MERS_N   624
#define MERS_M   397
#define MERS_R   31
#define MERS_U   11
#define MERS_S   7
#define MERS_T   15
#define MERS_L   18
#define MERS_A   0x9908B0DF
#define MERS_B   0x9D2C5680
#define MERS_C   0xEFC60000
#endif

public:
   CRandomMersenne(int seed) {         // Constructor
      RandomInit(seed); LastInterval = 0;}
   void RandomInit(int seed);          // Re-seed
   void RandomInitByArray(int const seeds[], int NumSeeds); // Seed by more
than 32 bits
   int IRandom (int min, int max);     // Output random integer
   int IRandomX(int min, int max);     // Output random integer, exact
   double Random();                    // Output random float
   uint32_t BRandom();                 // Output random bits
private:
   void Init0(int seed);               // Basic initialization procedure
   uint32_t mt[MERS_N];                // State vector
```

```
   int mti;                            // Index into mt
   uint32_t LastInterval;              // Last interval length for IRandomX
   uint32_t RLimit;                    // Rejection limit used by IRandomX
};


class CRandomMother {                  // Encapsulate random number generator
public:
   void RandomInit(int seed);          // Initialization
   int IRandom(int min, int max);      // Get integer random number in desired
interval
   double Random();                    // Get floating point random number
   uint32_t BRandom();                 // Output random bits
   CRandomMother(int seed) {           // Constructor
      RandomInit(seed);}
protected:
   uint32_t x[5];                      // History buffer
};

#endif // __cplusplus
#endif // RANDOMC_H
```