Trabalho Prático AEDS II

Filas, Pilhas e Complexidade

Danilo Pimentel de Carvalho Costa
E-mail: danilo.pimentel@dcc.ufmg.br
Matrícula: 2016058077

Introdução

O propósito do programa é analisar possíveis melhorias no desempenho de uma cantina no atendimento de seus usuários, levando em conta o tempo que estes gastam para conseguir seu almoço. O objetivo final é, através da simulação e experimentação de diferentes combinações das diferentes partes da estrutura de uma cantina, descobrir melhores maneiras de atender os clientes.

Desenvolvimento

Dado o problema, é preciso modelar as partes da cantina no sistema de forma que seja possível sua simulação. Assim, as partes consideradas para a implementação foram:

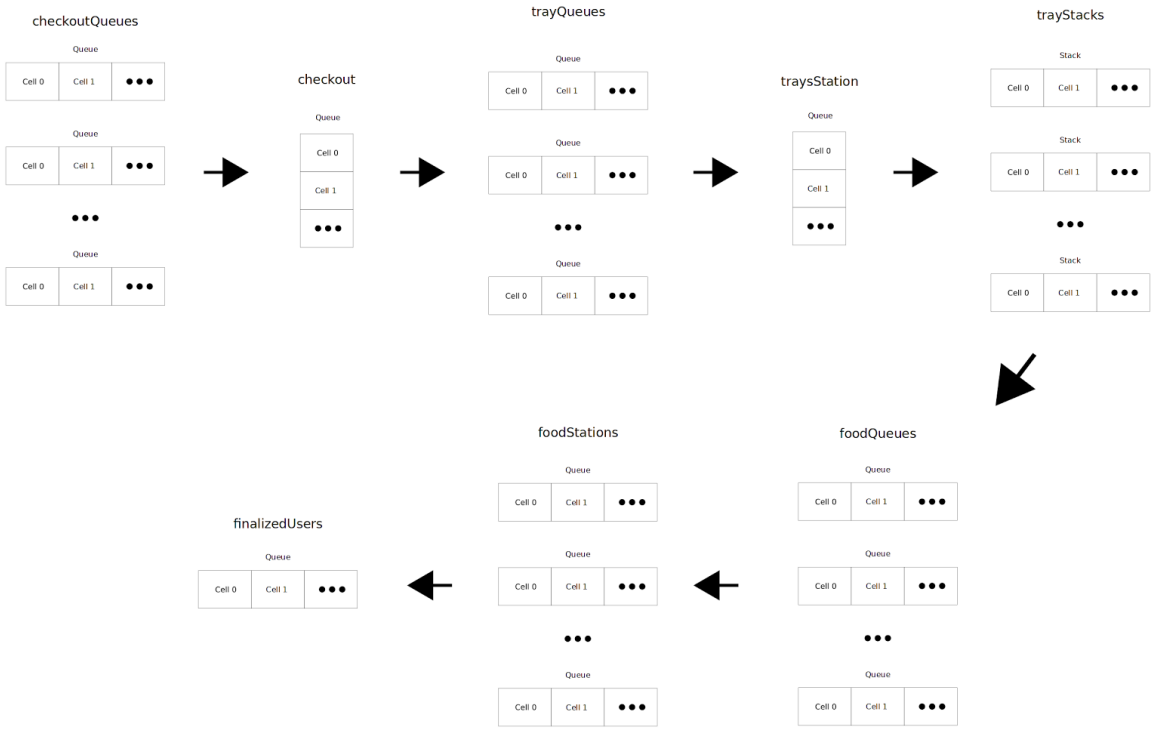
1. Fila(s) do caixa
2. Caixa(s)
3. Fila(s) de bandeja e talheres
4. Balcão(ões) de bandejas
5. Pilha(s) de bandeja e talheres

6. Fila(s) para comida
7. Balcão(ões) para comida

Assim, seguindo a ordem dos 7 itens acima, podemos visualizar o fluxo do usuário pelas partes da cantina. Todas as partes possuem um indicativo de plural, indicando que a solução dada pode ser configurada para simular diferentes combinações de quantidade destas partes.

Vista a modelagem do problema e a estrutura considerada, a implementação foi feita através da utilização dos Tipos de Dados Abstratos Fila e Pilha. Cada parte da cantina utiliza um destes TADs, como mostrado na seção de Implementação.

A seguir se encontra um diagrama da solução proposta e implementada no programa:



Implementação

Observações

- Por preferência, o código do programa foi desenvolvido em Inglês.
- Todo o código encontra-se comentado no formato utilizado para gerar a documentação através do programa *Doxygen.*
- Foi utilizado um repositório git para auxílio no versionamento do código. O acesso a este encontra-se neste link: https://github.com/danilo-p/aeds-ii-tp-1.
- Toda a inicialização, compilação, execução, geração de documentação a partir do código fonte, testes de memória foi feito através *Make*. Todas as *tasks* necessárias para estas ações são explicadas abaixo.

Makefile

- *setup*: Cria as pastas necessárias para o funcionamento do programa.
- *main*: Compila o programa e guarda o arquivo executável *main* na pasta *bin* localizada na raiz do projeto.
- *run*: Executa o programa.
- *docs*: Gera a documentação do projeto a partir do código fonte, através do programa *Doxygen*.
- *valgrind*: Executa o programa *Valgrind* para detectar vazamentos de memória.

Headers

A fim de organizar melhor o código do programa, as funções e estruturas foram divididas em *headers*. Os arquivos ".h" estão localizados na pasta *include* na raiz do projeto, enquanto os arquivos ".c" estão na pasta *src*, também na raiz do projeto.

Aqui não será explicado o código contido nos *headers*, a fim de resumir e não carregar esta breve documentação. **Assim, só será explicado o código contido no arquivo *main.c*, que contém toda a lógica da simulação.** Para mais detalhes da implementação dos

headers, consulte o código fonte e a documentação gerada através do comando *make docs.*

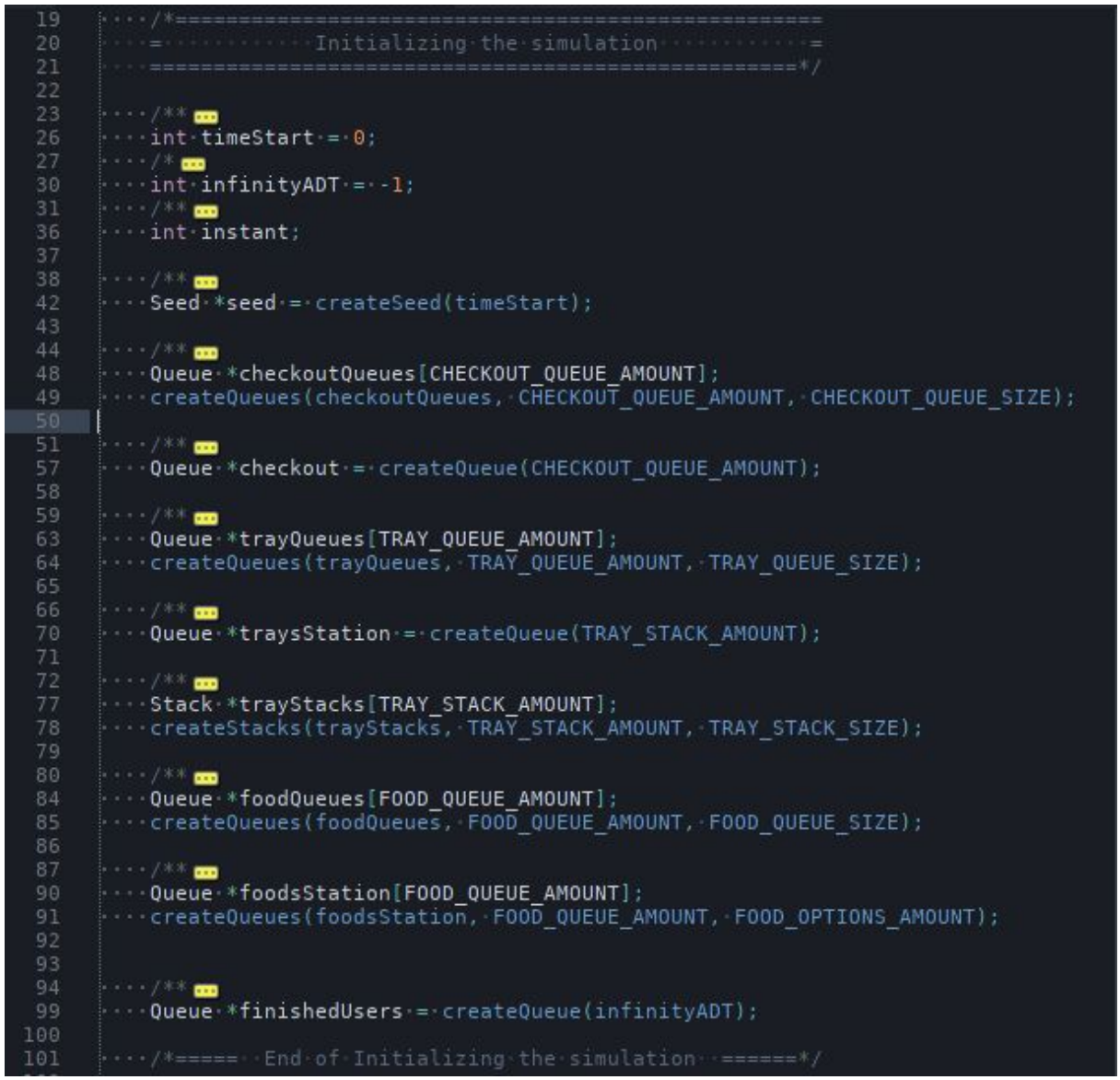
O arquivo principal

O arquivo main.c contém somente a função principal *main.* Esta é dividida em **Inicialização, Simulação, Apresentação de resultados e Finalização**.

A **Inicialização** consiste na criação das variáveis e TADs necessárias para a simulação. A imagem a seguir mostra o código desta seção. Explicando brevemente, os elementos mais importantes para a simulação são:

- *checkoutQueues*: A*rray* que guarda as filas de usuários do caixa.
- *checkout*: Fila que guarda os usuários que atualmente estão no serviço de caixa. A escolha da TAD fila foi arbitrária, e seu motivo se deve a facilidade de mudança dos usuários aqui presentes para as filas de bandeja.
- *trayQueues*: *Array* que guarda as filas de usuários que vão pegar as bandejas.
- *traysStation*: Fila de usuários que estão na vez de pegar uma bandeja da pilha de bandejas.
- *foodQueues*: *Array* de filas de usuários que estão aguardando para pegar a comida.
- *foodsStation*: *Array* de filas de usuários que já estão pegando a comida.
- *finishedUsers*: Fila de usuários que já finalizaram o serviço na cantina.

Seção de Inicialização da simulação



```
    /*================================================
    =            Initializing the simulation         =
    ================================================*/

    /** ...
    int timeStart = 0;
    /* ...
    int infinityADT = -1;
    /** ...
    int instant;

    /** ...
    Seed *seed = createSeed(timeStart);

    /** ...
    Queue *checkoutQueues[CHECKOUT_QUEUE_AMOUNT];
    createQueues(checkoutQueues, CHECKOUT_QUEUE_AMOUNT, CHECKOUT_QUEUE_SIZE);

    /** ...
    Queue *checkout = createQueue(CHECKOUT_QUEUE_AMOUNT);

    /** ...
    Queue *trayQueues[TRAY_QUEUE_AMOUNT];
    createQueues(trayQueues, TRAY_QUEUE_AMOUNT, TRAY_QUEUE_SIZE);

    /** ...
    Queue *traysStation = createQueue(TRAY_STACK_AMOUNT);

    /** ...
    Stack *trayStacks[TRAY_STACK_AMOUNT];
    createStacks(trayStacks, TRAY_STACK_AMOUNT, TRAY_STACK_SIZE);

    /** ...
    Queue *foodQueues[FOOD_QUEUE_AMOUNT];
    createQueues(foodQueues, FOOD_QUEUE_AMOUNT, FOOD_QUEUE_SIZE);

    /** ...
    Queue *foodsStation[FOOD_QUEUE_AMOUNT];
    createQueues(foodsStation, FOOD_QUEUE_AMOUNT, FOOD_OPTIONS_AMOUNT);


    /** ...
    Queue *finishedUsers = createQueue(infinityADT);

    /*=====  End of Initializing the simulation  ======*/
```

A **Simulação** consiste em um *loop* iterando de 1 em 1 “minuto” até o fim do tempo especificado. Neste *loop*, verificações são feitas para identificar os momentos para operações sobre as filas e pilhas da cantina. O *loop* itera a variável *instant*, que guarda o atual instante de tempo da simulação.

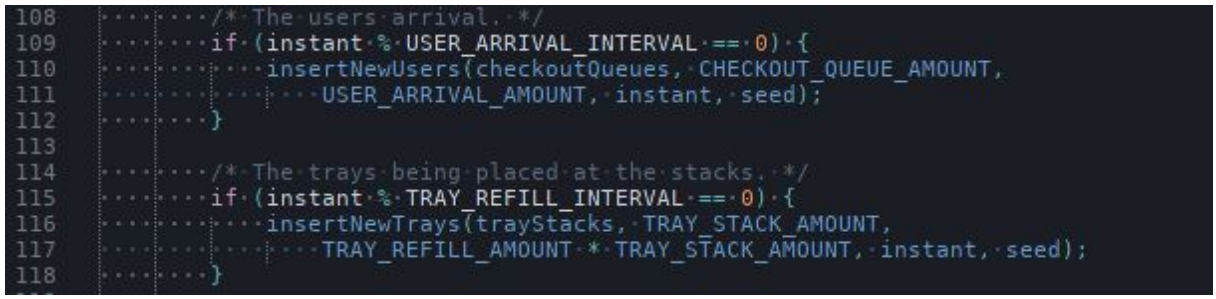
loop da Simulação

```
    /*==================================
    =            Simulation            =
    ==================================*/

    for(instant = timeStart; instant < DURATION; instant++) {
```

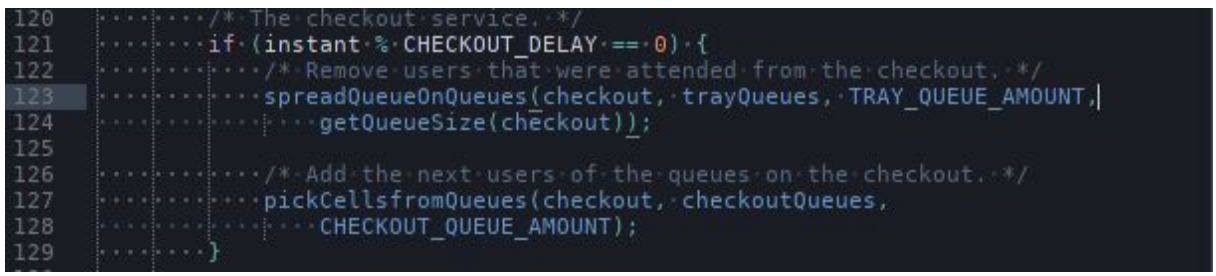
Os dois primeiros passos de uma iteração são as verificações para entrada de novos usuários e a reposição das bandejas. Quando os momentos destas duas ações chegam, os novos usuários são inseridos através da função *insertNewUsers* (*include/user.h* e *src/user.c*) e as novas bandejas são inseridas através da função *insertNewTrays (include/tray.h* e *src/tray.c)*.

Inserção de novos usuários e bandejas



O terceiro passo de uma iteração é a verificação para realização do serviço de caixa. Este passo consiste na retirada dos usuários do caixa (*checkout*) para as filas de bandejas (*trayQueues*) e na retirada de usuários das filas de caixa (*checkoutQueues*) para o caixa. A função *spreadQueueOnQueues (include/queue.h* e *src/queue.c*) pega os usuários do caixa e move estes para as filas de bandejas, enquanto a função *pickCellsFromQueues (include/queue.h* e *src/queue.c*) pega os primeiros usuários das filas de caixa e passa para o caixa.

Serviço de caixa



O quarto passo de uma iteração é a verificação para realização das operações referentes a bandeja. Estas são:

- Usuários que estão nas filas para bandejas (*trayQueues*) indo pegar uma bandeja. (Linha 157)

- Usuários que já estão prontos para pegar bandejas, que estão em *traysStation*, pegando bandejas da pilha de bandejas (*trayStacks*). (Linha 139)
- Usuários que conseguiram uma bandeja indo para as filas para pegar comida (*foodQueues*). (Linha 149)

Operações relacionadas a bandeja

```
131         /* The users picking trays from the stack. */
132         if (instant % TRAY_DELAY == 0) {
133             /* The desired amount of trays is the amount of users that will pick
134              * trays. */
135             int desiredTraysAmount = getQueueSize(traysStation);
136             Stack *usedTrays = createStack(desiredTraysAmount);
137
138             /* Remove the trays from the stacks. */
139             pickCellsfromStacks(usedTrays, trayStacks, TRAY_STACK_AMOUNT);
140
141             /*
142              * The amount of trays removed from the stacks. It can be different
143              * from the desired amount, because the stacks can be empty or the
144              * amount of trays can be lower than the needed amount.
145              */
146             int usedTraysAmount = getStackSize(usedTrays);
147
148             /* Remove usedTraysAmount amount of users from traysStation. */
149             spreadQueueOnQueues(traysStation, foodQueues,
150                     FOOD_QUEUE_AMOUNT, usedTraysAmount);
151
152             /*
153              * Add the next users from the tray queues to the trays station.
154              * Remaining users that were not attended will be in front of the
155              * new users.
156              */
157             pickCellsfromQueues(traysStation, trayQueues, TRAY_QUEUE_AMOUNT);
158
159             /* Destroys the auxiliar stack usedTrays. */
160             destroyStack(usedTrays, destroyTray);
161         }
```

O quinto e último passo do *loop* se refere ao passo da simulação em que dos usuários se servem da comida e terminam o processo dentro da cantina. Como são várias estações de comida (*foodStations*), é preciso checar todas as estações conferindo se algum usuário já chegou na última opção de comida (na implementação, se a fila da estação está cheia). Se alguma das filas estiver cheia, o primeiro usuário é retirado, é marcado o tempo de finalização deste para posterior cálculo da média gasta pelos usuários e este é levado para a fila de usuários finalizados (*finishedUsers*). Depois de abrir mais espaço para mais usuários se servirem, os primeiros usuários da fila para comida (*foodQueues*) são trazidos para as estações de comida.

Usuários servindo comida

```
163         /* Users picking food */
164         if (instant % FOOD_DELAY == 0) {
165             int i;
166
167             /*
168              * Here its needed to check queue per queue if it is full. If is the
169              * case, the first user of the queue is done.
170              */
171             for(i = 0; i < FOOD_QUEUE_AMOUNT; i++) {
172                 if (isQueueFull(foodsStation[i])) {
173                     Cell *finishedUserCell = popCellFromQueue(foodsStation[i]);
174
175                     User *finishedUser = (User *) finishedUserCell->data;
176
177                     finishedUser->finishedInstant = instant;
178
179                     pushCellOnQueue(finishedUsers, finishedUserCell);
180                 }
181             }
182
183             /*
184              * Move the next users from the foodQueues to the foodsStations.
185              */
186             Queue *usersGoingToFoodStation = createQueue(FOOD_QUEUE_AMOUNT);
187             pickCellsfromQueues(usersGoingToFoodStation, foodQueues,
188                     FOOD_QUEUE_AMOUNT);
189             spreadQueueOnQueues(usersGoingToFoodStation, foodsStation,
190                     FOOD_QUEUE_AMOUNT, getQueueSize(usersGoingToFoodStation));
191             destroyQueue(usersGoingToFoodStation, destroyUser);
192         }
```

Na **Apresentação de resultados**, a média de tempo gasto por um usuário é apresentada, e, opcionalmente, o estado final das partes da cantina. Para ver o estado final das partes da cantina, uma opção *VERBOSE_RESULTS* deve ser habilitada nas configurações do programa (*src/config.c*).

Apresentação de Resultados

```
    /*=======================================================
    =            Presenting Simulation Results              =
    =======================================================*/

    if (VERBOSE_RESULTS) {
        printf("\n--- seed --\n");
        printSeed(seed);

        printf("\n--- checkoutQueues --\n");
        printQueues(checkoutQueues, CHECKOUT_QUEUE_AMOUNT, printUser);

        printf("\n--- checkout --\n");
        printQueue(checkout, printUser);

        printf("\n--- trayQueues --\n");
        printQueues(trayQueues, TRAY_QUEUE_AMOUNT, printUser);

        printf("\n--- traysStation --\n");
        printQueue(traysStation, printUser);

        printf("\n--- trayStacks --\n");
        printStacks(trayStacks, TRAY_STACK_AMOUNT, printTray);

        printf("\n--- foodQueues --\n");
        printQueues(foodQueues, FOOD_QUEUE_AMOUNT, printUser);

        printf("\n--- foodsStation --\n");
        printQueues(foodsStation, FOOD_QUEUE_AMOUNT, printUser);

        printf("\n--- finishedUsers --\n");
        printQueue(finishedUsers, printUser);
    }

    printf("\n-------------------------------------------\n");

    printf("\nTime user spent average: %.4f minutes\n",
        getTimeUserSpentAverage(&finishedUsers, timeStart));

    printf("\n-------------------------------------------\n\n");

    /*======  End of Presenting Simulation Results  ======*/
```

Na **Finalização**, os espaços alocados na memória são liberados através de funções de destruição das estruturas e TADs.

Finalização da Simulação

```
    /*=======================================
    =            Finalizing Simulation      =
    =======================================*/

    destroySeed(seed);
    destroyQueue(checkout, destroyUser);
    destroyQueues(checkoutQueues, CHECKOUT_QUEUE_AMOUNT, destroyUser);
    destroyQueue(traysStation, destroyUser);
    destroyQueues(trayQueues, TRAY_QUEUE_AMOUNT, destroyUser);
    destroyStacks(trayStacks, TRAY_STACK_AMOUNT, destroyTray);
    destroyQueues(foodsStation, FOOD_QUEUE_AMOUNT, destroyUser);
    destroyQueues(foodQueues, FOOD_QUEUE_AMOUNT, destroyUser);
    destroyQueue(finishedUsers, destroyUser);

    /*======  End of Finalizing Simulation  ======*/
```

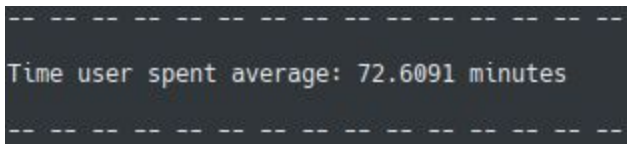
Análise

Foram determinadas as complexidades de todas as funções utilizadas na simulação. Estas se encontram nos comentários acima da implementação de cada função, nos seus arquivos fonte localizados na pasta *src* que se encontra na raiz do projeto.

Resultados

Com as configurações padrão, a média de tempo gasto foi de aproximadamente 73 minutos. A média alta é causada por uma defasagem no atendimento dos usuários no caixa. Como a cada instante de tempo chegam 2 usuários, 1 é atendido e outro aguarda o atendimento. Depois de 240 minutos, muitos usuários estão na fila esperando.

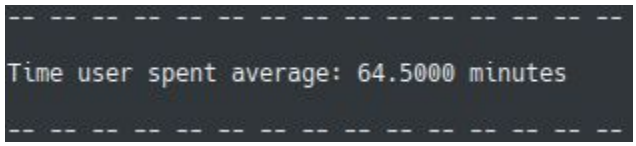
Configurações Padrão



Com a primeira sugestão de alterações, o tempo médio caiu para aproximadamente 64 minutos. A média alta é causada pela baixa quantidade de filas de bandeja. 2 usuários são atendidos por vez no caixa, mas quando estão na fila de bandeja, somente 1 consegue pegar a bandeja por vez. As configurações alteradas no arquivo de configuração foram:

- const int CHECKOUT_QUEUE_AMOUNT = 2;
- const int TRAY_STACK_AMOUNT = 2;

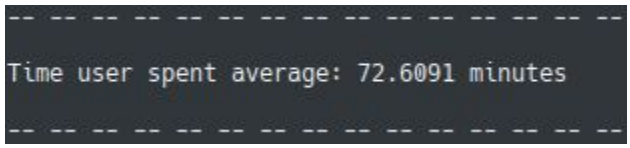
Primeira alteração



A segunda sugestão de alteração não alterou o tempo médio gasto pelos usuários em relação às configurações padrão. A média de tempo não se altera, por que colocando mais filas para bandeja não soluciona o problema com a quantidade baixa de filas e demora no atendimento do caixa. As configurações alteradas no arquivo de configuração foram:

- const int TRAY_QUEUE_AMOUNT = 2;
- const int TRAY_STACK_SIZE = 40;

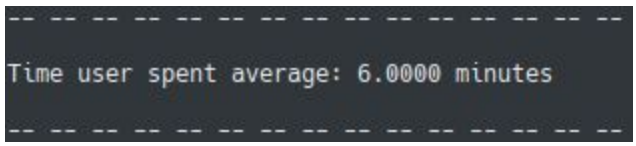
Segunda alteração



Uma alteração que solucionaria o problema com a demora nos serviços da cantina seria: 2 filas de caixa, 2 filas para bandeja, 2 pilhas de bandeja, reposição de 14 bandejas (mantém-se o intervalo padrão), 2 filas para comida. Com esta alteração, o tempo médio de cada usuário fica em 6 minutos. As configurações alteradas no arquivo de configuração foram:

- const int CHECKOUT_QUEUE_AMOUNT = 2;
- const int TRAY_QUEUE_AMOUNT = 2;
- const int TRAY_STACK_AMOUNT = 2;
- const int TRAY_REFILL_AMOUNT = 14;
- const int FOOD_QUEUE_AMOUNT = 2;

Melhor tempo médio



Conclusão

Comparando os resultados apresentados na seção anterior, podemos concluir que a última sugestão de alteração foi a que obteve melhor resultado (em média, 6 minutos).