1-N produtor e consumidores

Gustavo Henrique Grützmann (19102053) Vitor Sorpile Geraldo (19102063)

I. INTRODUÇÃO

Este projeto tem como objetivo apresentar uma solução para o problema clássico "Produtor-Consumidor", porém limitando o número de produtores e de refeições produzidas. Nesse contexto, foi desenvolvida uma solução genérica para N produtores e N consumidores que são determinados por *defines* no código e cada consumidor/produtor é uma *thread* e é usado um semáforo para sincronização.

II. DIAGRAMAS

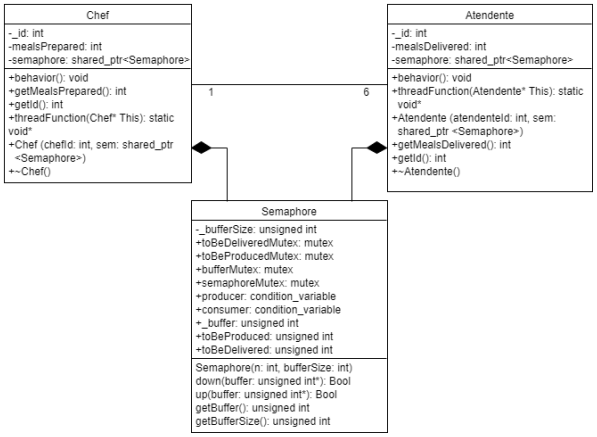

Fig. 1. Diagrama de Classes

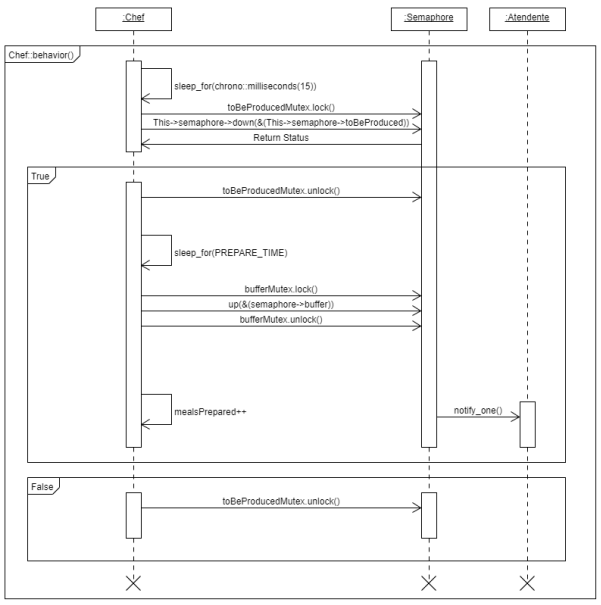

Fig. 2. Diagrama de Sequência da classe Chef

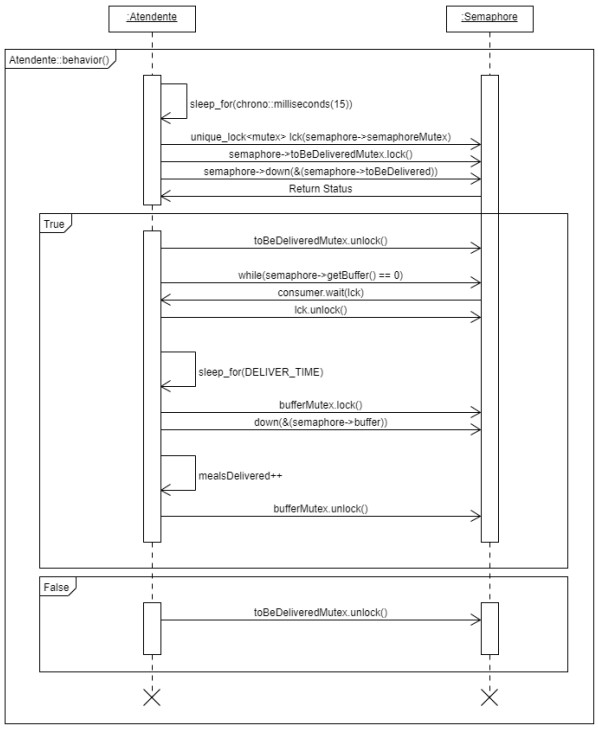

Fig. 3. Diagrama de Sequência da classe Atendente

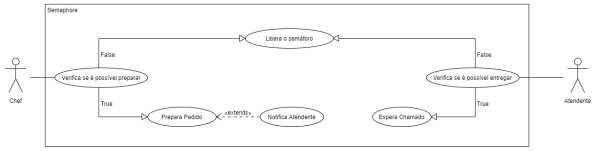

Fig. 4. Diagrama de Casos de Uso

III. CLASSES

A. *Chef*

Classe derivada de *std::thread* responsável por representar um produtor, possui dois métodos *get*, um para o id que é passado para o construtor e outro para o contador de refeições produzidas, um ponteiro para um objeto da classe Semaphore, também passado no construtor, e o método *behavior()* que define seu comportamento, implementado como um laço no qual o objeto verifica através do semáforo se ainda é preciso produzir refeições, caso sim, produz e notifica um consumidor, caso não, quebra o laço. Também possui o método *static threadBehavior()*, passado para o construtor de *thread* e que apenas chama *behavior()*.

B. Atendente

Classe derivada de *std::thread* responsável por representar um consumidor, possui dois métodos *get* e um ponteiro para um objeto da classe Semaphore da mesma forma que a classe Chef e o método *behavior()* que define seu comportamento, implementado como um laço no qual o objeto verifica através do semáforo se ainda é preciso entregar refeições, caso sim, entrega ou espera até uma ser produzida, caso não, quebra o laço. Também possui o método *static threadBehavior()*, passado para o construtor de *thread* e que apenas chama *behavior()*.

C. Semaphore

Classe responsável por controlar o acesso aos recursos compartilhados (buffer entre produtores e consumidores, contador de refeições a serem produzidas e contador de refeições a serem entregues). Utiliza *mutexes* e variáveis condicionais para limitar o acesso às regiões críticas e notificar os consumidores quando uma refeição é produzida e possui os métodos *down* e *up* para controlar se é possível retirar/acrescentar ao buffer e dois métodos *get*, um para o tamanho do buffer e outro para o buffer em si.

IV. Compilação e Execução

O programa é compilado com um $Makefile$ e pode ser executado através do comando "./exercicio2 N", onde N é um argumento opcional, por padrão assume o valor 50, que representa quantas refeições serão produzidas e consumidas.

V. Testes

Para verificar o funcionamento correto do programa, foram executados vários testes com diferentes números de refeições a serem produzidas e consumidas. As flags utilizadas na compilação foram *-Wall -Wextra -Wshadow -Werror -std=c++17 -pthread -pedantic -g* e para verificar se existiam erros de memória foi o programa foi executado com o *valgrind* com as flags *–leak-check=full –track-origins=yes -s*.

```
vitorsorpile@DESKTOP-KNUPDEI:~/UFSC/SO/2_1-N_produtor_e_consumidores$ ./exercicio2
Atendente 1 entregou 11
Atendente 2 entregou 8
Atendente 3 entregou 8
Atendente 4 entregou 7
Atendente 5 entregou 9
Atendente 6 entregou 7
O atendente menos ocioso foi o 1
O atendente mais ocioso foi o 4
```

Fig. 5. Saída padrão

```
vitorsorpile@DESKTOP-KNUPDEI:~/UFSC/SO/2_1-N_produtor_e_consumidores$ ./exercicio2 200
Atendente 1 entregou 37
Atendente 2 entregou 22
Atendente 3 entregou 54
Atendente 4 entregou 27
Atendente 5 entregou 29
Atendente 6 entregou 31
O atendente menos ocioso foi o 3
O atendente mais ocioso foi o 2
```

Fig. 6. Saída com 200 refeições

```
valgrind --leak-check=full --track-origins=yes ./exercicio2
==32689== Memcheck, a memory error detector
==32689== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==32689== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==32689== Command: ./exercicio2
==32689==
Atendente 1 entregou 13
Atendente 2 entregou 8
Atendente 3 entregou 11
Atendente 4 entregou 10
Atendente 5 entregou 8
Atendente 6 entregou 0
O atendente menos ocioso foi o 1
O atendente mais ocioso foi o 6
==32689==
==32689== HEAP SUMMARY:
==32689==     in use at exit: 0 bytes in 0 blocks
==32689==   total heap usage: 30 allocs, 30 frees, 76,568 bytes allocated
==32689==
==32689== All heap blocks were freed -- no leaks are possible
==32689==
==32689== For lists of detected and suppressed errors, rerun with: -s
==32689== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Fig. 7. Saída com o valgrind