Relatório do Projeto de Programação Concorrente

E-commerce de Venda de Camisetas

Autor

Leandro Beloti Kornelius - 211020900

Dep. Ciência da Computação - Universidade de Brasilia (UnB)

CIC0202 - Programação Concorrente ministrada por Eduardo Adilio Pelinson Alchieri

08/01/2025

Abstract. This report addresses a problem involving race conditions, requiring concurrent programming resources to solve process communication issues through shared memory.

Resumo. Este relatório aborda um problema que contém condições de corrida, necessitando recursos de programação concorrente para solucionar problemas de comunicação de processos através da memória compartilhada.

1. Introdução

A Programação Concorrente é um paradigma de programação que permite a resolução de problemas que necessitam de execuções simultâneas compartilhando do mesmo recurso, o que exige técnicas de sincronização e comunicação para evitar condições de corrida e deadlocks.

Com isso, a disciplina de Programação Concorrente visa introduzir aos estudantes os conceitos e soluções para problemas clássicos desse paradigma de suma importância nos dias de hoje. Nesse aspecto, foram abordadas as principais técnicas para solucionar condições de corrida: locks, variáveis de condição, sinais, etc. Para praticar tais conceitos, foi proposto um trabalho para desenvolvermos um algoritmo que trate problemas de comunicação entre processos através de uma memória compartilhada.

Dessa forma, o trabalho requer que seja elaborado um problema que contenha concorrência entre os processos e proponha uma solução para tal situação. O problema deve conter comunicação entre os processos através de recursos compartilhados, ou seja, situações em que múltiplos processos manipulam a mesma informação e o resultado depende de qual dos processos irá executar primeiro. Assim, a solução deve conter os mecanismos de sincronização de processo vistos ao decorrer da disciplina e fazer uso da biblioteca POSIX Pthreads.

O problema proposto pelo autor deste relatório, Leandro Beloti Kornelius, é um e-commerce de camisetas. As particularidades e solução do problema são explicadas nas seções abaixo deste relatório.

2. Problema do E-commerce de Camisetas

A empresa de camisetas VDL, focada em moda streetwear, realiza a venda de blusas como principal fonte de crescimento e sustentabilidade, permitindo o pagamento de seus funcionários e a expansão do negócio. Para manter a operação contínua, a VDL depende de fornecedores que abastecem regularmente o estoque de materiais

essenciais para a produção de suas peças. Com um bom fluxo de abastecimento e produção de novas camisetas, a empresa consegue minimizar a falta de produtos de qualidade aos seus clientes.

Atualmente, a VDL trabalha com duas linhas de camisetas: Fast e Unique. A linha Fast é composta por camisetas básicas, de fácil produção, que garantem agilidade e maior frequência de reposição no estoque. Para produzir uma unidade dessa linha, são necessários 500g de algodão, uma etiqueta e um logo da marca. Por outro lado, a linha Unique é composta por camisetas personalizadas manualmente, com uma produção mais demorada devido ao processo artesanal de estamparia. Cada camiseta da linha Unique exige 500g de algodão, uma etiqueta, um logo da marca e uma estampa personalizada, tornando a produção mais complexa, porém com maior valor agregado ao produto.

Realize um algoritmo que simula esse processo de produção, abastecimento e venda de camisetas utilizando threads da biblioteca POSIX Pthreads para representar os sócios, os fornecedores e os clientes da VDL.

Em função da VDL visar a maior satisfação de seus clientes, os sócios priorizam a produção de camisetas da linha Unique quando há materiais suficientes, mas produzem camisetas da linha Fast caso a personalização esteja indisponível.

Os clientes que chegam ao site querem realizar a compra de uma camiseta dando preferência para camisetas da linha Unique, mas se estiverem indisponíveis, compram da linha Fast. Caso não haja blusas disponíveis, eles aguardam a reposição do estoque para realizar a compra.

O abastecimento dos materiais para a produção de novas camisetas ocorre em intervalos regulares definidos pelo fornecedor.

A solução deve fazer uso de mutexes e variáveis de condição para coordenar as operações de produção, venda e reabastecimento, evitando deadlocks e garantindo a correta manipulação dos estoques. Assim, a empresa pode equilibrar sua produção e venda, maximizando a disponibilidade de produtos e a satisfação dos clientes.

3. Solução

Para resolver o problema proposto foi usado locks e variáveis de condição. O código que contém a solução e os mecanismos de sincronização de processos foi armazenado em um repositório no GitHub, o qual pode ser acessado no link ao lado: https://github.com/LeandroKornelius/Concurrent-Programming-Project. Na solução proposta, o código foi dividido em cinco partes: declaração de variáveis, função de produção de novas camisetas, função de venda de camisetas, função de abastecimento dos insumos de produção, função principal do programa e resultados.

3.1 Declaração de Variáveis

Neste momento, foram definidas as variáveis globais que serão usadas em outras partes do programa. Sob essa ótica, da *linha 6 a 20* foram definidos as quantidades iniciais dos estoques. Ou seja, tem a quantidade inicial dos materiais necessários para produção das camisetas. Ademais, tem as quantidades iniciais das camisetas de cada linha, Fast e Unique.

```
6   // Initial amount of materials to make the shirts
7   #define COTTON 2000 // amount in grams
8   #define TAGS 4
9   #define LOGOS 4
10  #define PERSONALIZATION 2
11
12  // Stock of shirts
13  int stockFast = 10;      You, 14 hours ago • ✨ feat: pr
14  int stockUnique = 2;
15
16  // Available materials
17  int cotton = COTTON;
18  int tags = TAGS;
19  int logos = LOGOS;
20  int personalization = PERSONALIZATION;
21
22  // Mutex and condition variables
23  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
24  pthread_cond_t condMaterials = PTHREAD_COND_INITIALIZER;
25  pthread_cond_t condStock = PTHREAD_COND_INITIALIZER;
```

Observa-se através do código acima que o programa tem material suficiente para produzir 2 camisetas Unique e 2 camisetas Fast. Ademais, o estoque de camisetas é iniciado com 2 camisetas Unique e 10 camisetas Fast.

A partir da *linha 22*, são declaradas as técnicas de sincronização que foram usadas para resolver o problema em questão: locks e variáveis de condição. O lock **mutex** será utilizado para impedir que produtores e clientes manipulem o estoque de camisetas simultaneamente. Já as variáveis de condição de materiais e de estoque (**condMaterials** & **condStock**, respectivamente) servem para alertar aos produtores e clientes que não precisam mais esperar para produzir novas camisetas ou realizar a compra, respectivamente.

3.2 Thread dos Produtores

Essa função simula a produção de novas camisetas de forma contínua pelos produtores.

```
27  void *produce(void *args) {
28      while (1) {
29          pthread_mutex_lock(&mutex); // locks to access the materials
30
31          // Checks to see if there are enought materials to produce a Unique shirt
32          if (cotton >= 500 && tags >= 1 && logos >= 1 && personalization >= 1) {
33              cotton -= 500;
34              tags--;
35              logos--;
36              personalization--;
37              stockUnique++;
38              printf("Produced one Unique Shirt. Unique Shirt stock: %d\n", stockUnique);
39              pthread_cond_broadcast(&condStock); // signals the clients that a just produced unique shirt is available
40          }
41
42          else if (cotton >= 500 && tags >= 1 && logos >= 1) {
43              cotton -= 500;
44              tags--;
45              logos--;
46              stockFast++;
47              printf("Produced one Fast Shirt. Fast Shirt stock: %d\n", stockFast);
48              pthread_cond_broadcast(&condStock); // signals the clients that a just produced fast shirt is available
49          }
50
51          else {
52              // There is not enough material to produce any of the two shirts, must wait for more material
53              printf("Waiting for more materials to produce shirts...\n");
54              pthread_cond_wait(&condMaterials, &mutex);
55          }
56
57          pthread_mutex_unlock(&mutex); // unlocks the materials so other threads can manipulate it
58          sleep(1);
59      }
60
61      return NULL;
62  }
```

O funcionamento dessa thread inicia-se bloqueando o **mutex** *(linha 29)* para impedir que outros processos manipulem simultaneamente os materiais necessários à produção e o estoque de camisetas.

Em seguida, é feita uma verificação: se há material suficiente para realizar uma nova camiseta Unique *(linha 32)*. Somente se não houver material suficiente para produzir uma camiseta do tipo Unique, é verificado se é possível produzir uma camiseta da linha Fast *(linha 42)* a qual, se possível, será produzida. Ou seja, os produtores irão dar prioridade para confecção de camisetas do tipo Unique.

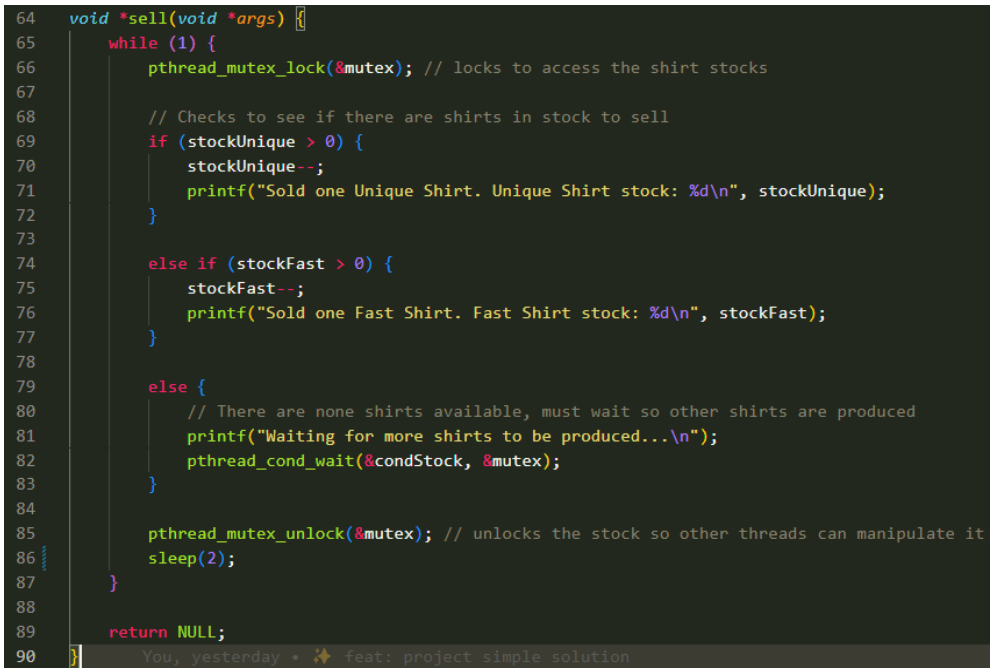
Ao produzir qualquer camiseta, é feita uma sinalização com **pthread_cond_broadcast** para que as threads dos clientes sejam notificadas sobre a disponibilidade de novas camisetas no estoque *(linhas 39 & 48)*.

Se não houver material para produção de ambas camisetas, a thread entra em espera utilizando a variável de condição **condMaterials** até que ocorra o abastecimento de insumos para confecção de mais camisetas *(linha 54)*.

Após a liberação do **mutex**, a thread entra em espera por um intervalo de 2 segundos, simulando o tempo de produção *(linha 57-58)*.

3.3 Thread dos Clientes

Essa função simula a venda de camisetas aos clientes de forma contínua.



```
64  void *sell(void *args) {
65      while (1) {
66          pthread_mutex_lock(&mutex); // locks to access the shirt stocks
67
68          // Checks to see if there are shirts in stock to sell
69          if (stockUnique > 0) {
70              stockUnique--;
71              printf("Sold one Unique Shirt. Unique Shirt stock: %d\n", stockUnique);
72          }
73
74          else if (stockFast > 0) {
75              stockFast--;
76              printf("Sold one Fast Shirt. Fast Shirt stock: %d\n", stockFast);
77          }
78
79          else {
80              // There are none shirts available, must wait so other shirts are produced
81              printf("Waiting for more shirts to be produced...\n");
82              pthread_cond_wait(&condStock, &mutex);
83          }
84
85          pthread_mutex_unlock(&mutex); // unlocks the stock so other threads can manipulate it
86          sleep(2);
87      }
88
89      return NULL;
90  }       You, yesterday • ✨ feat: project simple solution
```

A execução da thread dos clientes inicia-se bloqueando o **mutex** *(linha 66)* para garantir acesso exclusivo ao estoque de camisetas. Nesse sentido, o cliente irá comprar uma camiseta sempre que disponível, dando prioridade para as da linha Unique *(em função do primeiro if - linha 69)*. Primeiro, é verificado se há camisetas Unique em estoque. Se sim, o cliente efetuará a compra. Se não houver estoque das camisetas personalizadas, é feita uma verificação para averiguar a disponibilidade das camisetas da linha Fast. Havendo estoque, a venda é realizada, reduzindo a quantidade correspondente no estoque *(linha 70 & 75)*.

Caso não tenha nenhuma camiseta disponível, a thread entra em espera utilizando a variável de condição **condStock** *(linha 82)* até que novas camisetas sejam produzidas pelas threads de produção.

Depois de realizar a venda ou de ser notificada, a thread libera o **mutex** *(linha 85)* e entra em espera por um intervalo de 1 segundo, simulando o tempo de compra do cliente *(linha 86)*.

3.4 Thread dos Fornecedores

Essa função simula o reabastecimento dos materiais necessários à produção de camisetas.

```
92  void *restock(void *args) {
93      while (1) {
94          pthread_mutex_unlock(&mutex); // locks the materials to manipulate it
95
96          cotton += 1000;
97          tags += 2;
98          logos += 2;
99          personalization += 1;
100         printf("Restocked the materials in the below amount:\nCotton: %d;\nTags: %d;\nLogos: %d;\nPersonalization: %d\n",
101                 cotton, tags, logos, personalization);
102
103         // Must signal that there is new material so new shirts can be produced and later bought
104         pthread_cond_signal(&condMaterials);
105         pthread_mutex_unlock(&mutex);
106         sleep(8);
107     }
108
109     return NULL;
110 }       You, yesterday • ✨ feat: project simple solution
```

A thread inicia bloqueando o **mutex** *(linha 94)* para manipular os materiais de forma segura. Em cada ciclo, é adicionado ao estoque uma quantidade fixa de insumos (1000 gramas de algodão, 2 tags, 2 logos e 1 personalização).

Após o reabastecimento, é possível fazer mais camisetas, sendo necessário avisar aos produtores em aguardo por novos materiais. Tal ação é feita usando uma sinalização com **pthread_cond_signal** *(linha 104)* para que as threads dos produtores possam ser notificadas sobre a disponibilidade de novos materiais.

Em seguida, a thread libera o mutex e entra em espera por um intervalo de 8 segundos, simulando o tempo necessário para o abastecimento dos insumos *(linha 106)*.

3.5 Função Principal

A função principal cria as threads de produtores, clientes e do fornecedor, utilizando a função **pthread_create** *(linhas 119, 124 & 128)*. São criadas duas threads de produtores, cinco threads de clientes e uma thread do fornecedor.

Após a criação das threads, a função principal utiliza **pthread_join** *(linhas 132, 135 & 137)* para aguardar a finalização de cada uma das threads.

```
112  int main() {
113      pthread_t thread_producers[2];   // 2 threads for producers
114      pthread_t thread_clients[5];     // 5 threads for clientes
115      pthread_t thread_supplier;       // 1 thread for supplier
116
117      // Creates the threads of the producers
118      for (int i = 0; i < 2; i++) {
119          pthread_create(&thread_producers[i], NULL, produce, NULL);
120      }
121
122      // Creates the threads of the clients
123      for (int i = 0; i < 5; i++) {
124          pthread_create(&thread_clients[i], NULL, sell, NULL);
125      }
126
127      // Creates the threads of the supplier
128      pthread_create(&thread_supplier, NULL, restock, NULL);
129
130      // Executes the threads - will require a force break using ctrl C
131      for (int i = 0; i < 2; i++) {
132          pthread_join(thread_producers[i], NULL);
133      }
134      for (int i = 0; i < 5; i++) {
135          pthread_join(thread_clients[i], NULL);
136      }
137      pthread_join(thread_supplier, NULL);
138
139      return 0;
140  }
```

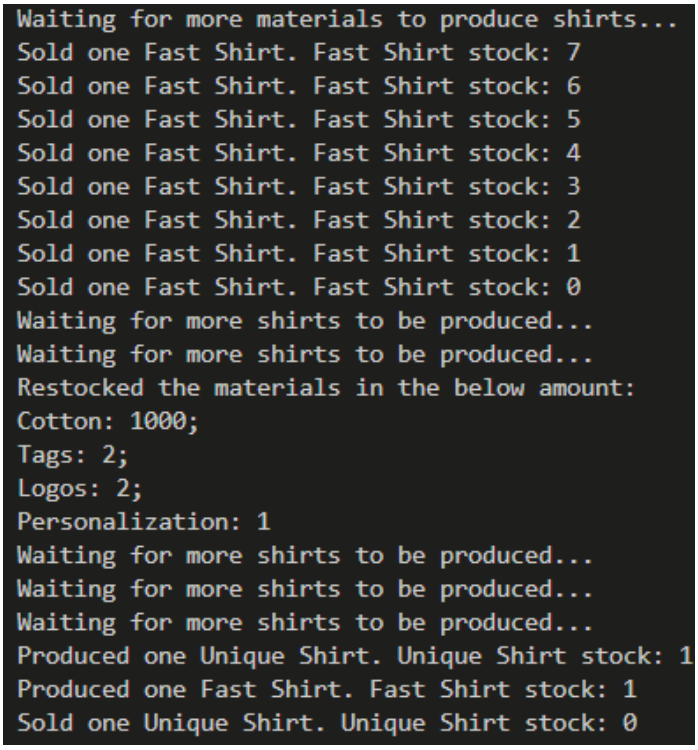
Como o programa é projetado para rodar continuamente, ele só pode ser interrompido manualmente pelo usuário (por exemplo, usando *Ctrl* + *C*).

3.6 Resultados

Com intuito de abordar mais a solução do problema proposto, assim como mostrar sua execução foi gravado um pequeno vídeo:

https://www.youtube.com/watch?v=mrfcEEooqQs

Ademais, nota-se na imagem abaixo que o programa realiza a produção e venda de ambas camisetas, abastecimento dos recursos necessários para confecção de novas unidades e espera de novas camisetas ou materiais:



4. Conclusão

A solução desenvolvida demonstra a importância da programação concorrente no gerenciamento eficiente de recursos compartilhados e na execução simultânea de múltiplas tarefas. No contexto deste projeto, foi essencial garantir que a produção, venda e reabastecimento de camisetas ocorresse de maneira ordenada, evitando conflitos e garantindo o fluxo contínuo do sistema.

Por meio do uso de locks e variáveis de condição, foi possível implementar uma comunicação eficaz entre as threads responsáveis por cada operação. Os locks asseguraram que apenas uma thread acessasse as variáveis críticas por vez, prevenindo inconsistências nos estoques de materiais e camisetas. As variáveis de condição, por sua vez, foram fundamentais para sincronizar as threads, permitindo que esperassem por eventos específicos, como a disponibilidade de materiais ou a chegada de novos estoques.

Essa abordagem possibilitou a criação de um sistema robusto, onde a produção se adapta dinamicamente à demanda dos clientes e ao fornecimento de insumos. As técnicas empregadas garantiram que os recursos fossem utilizados de maneira eficiente, maximizando a produtividade e evitando situações de espera indefinida.

5. Referências

- Aulas & códigos da disciplina
- https://www.geeksforgeeks.org/multithreading-in-c/
- https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html