PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE TECNOLOGIA

CHRISTOPHER OLIVEIRA	**RA:18726430**
GIULIANO SANFINS	**RA:17142837**
MATHEUS MORETTI	**RA:18082974**
MURILO ARAUJO	**RA:17747775**
VICTOR REIS	**RA:18726471**

SISTEMAS OPERACIONAIS A - EXPERIMENTO 5

CAMPINAS
2020

SUMÁRIO

1. INTRODUÇÃO

O experimento tem como objetivo principal a aplicação de todas as ferramentas apresentadas durante o semestre, sendo elas as filas de mensagens, memória compartilhada, mutex, semáforos, e threads, e com elas buscar implementar uma solução para o problema clássico do barbeiro dorminhoco.

Diferentemente das outras atividades, não recebemos um código base para correção lógica e sintática, apenas a explicação do problema e o que a solução deveria conter, e após a implementação da solução, rodar testes e com base nas saídas obtidas, fazer uma análise e gerar uma conclusão.

2. DISCUSSÃO

A respeito da primeira parte do experimento foi pedido para que implementássemos a solução do problema, do barbeiro dorminhoco, utilizando filas de mensagens como mecanismo de sincronismo, e memória compartilhada, para o acesso dos processos barbeiros e clientes aos vetores, também foi utilizado semáforo para a garantia da exclusão mútua no acesso a memória compartilhada. Na implementação é pedido que entre 20 clientes, a cada um que chega ao “salão” verifica se algum dos 2 barbeiros estão livre, se não, é colocado entre 7 cadeiras de espera, ao sentar é pedido que ele envie uma *string* de número aleatório entre 2 e 1023, se estas também estiverem todas ocupadas, o cliente deve ir embora ou voltar depois, ao cliente sentar na cadeira do barbeiro, o barbeiro deve ordenar o vetor recebido e mandá-lo de volta ao cliente.

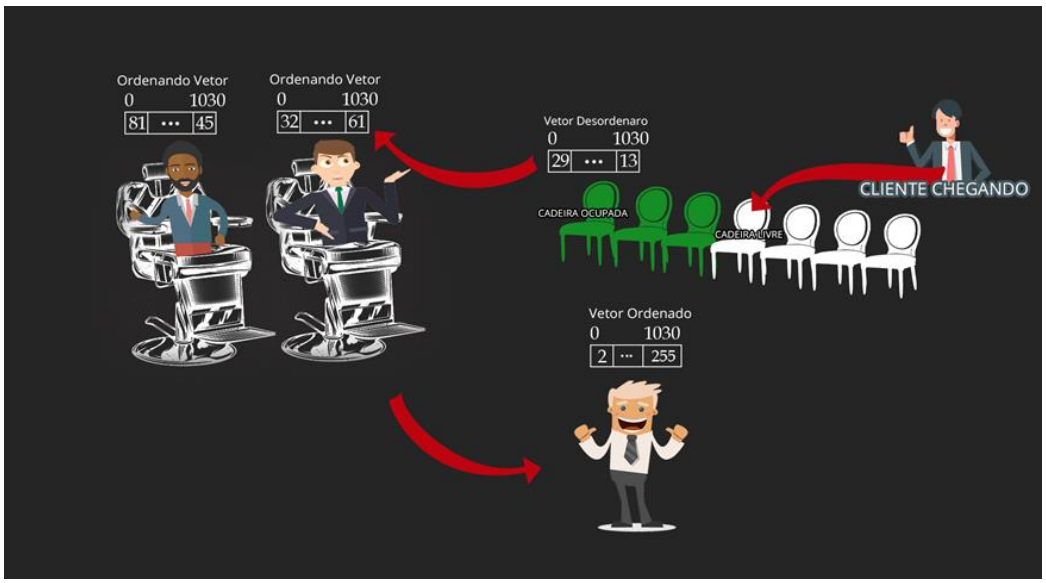

Figura 1: Representação gráfica do problema do Barbeiro Dorminhoco

Para a segunda parte, é pedido outra implementação, onde é exigido o uso de threads ao invés de processos, mutex no lugar de semáforos, semáforos ao invés de fila de mensagens. Na implementação é exigido agora 3 barbeiros e 27 clientes, mantendo o número de cadeiras de espera, diferente da primeira implementação o vetor deverá ser ordenado crescentemente. A lógica da implementação continua a mesma da primeira parte do experimento.

4 PERGUNTAS

4.1 Pergunta 1

Qual é o recurso comum que necessita de exclusão mútua?

R: As cadeiras de espera são o recurso que requer o uso de exclusão mútua.

4.1 Pergunta 2

De que maneira (leitura, escrita, ambos) barbeiros e clientes vão acessar o recurso comum?

R: Através do uso de Memória compartilhada.

4.1 Pergunta 3

Como os números foram colocados na *string*?

R: No nosso caso utilizamos a própria interpretação dos caracteres em inteiros, não sendo necessário transformar o *int* para *string* e vice-versa.

4.1 Pergunta 4

Como o barbeiro vai ter acesso aos valores a serem ordenados?

R: Através do uso de Memória compartilhada.

4.1 Pergunta 5

Como o cliente vai ter acesso aos resultados?

R:Através do uso de Memória compartilhada.

5. ANÁLISE DOS RESULTADOS

5.1 Parte 1

Nesta primeira parte do experimento do barbeiro dorminhoco, desenvolvemos uma aplicação com alguns requisitos solicitados pelo docente, onde tínhamos que comunicar o cliente com o barbeiro através de uma única fila de mensagens e com processos filhos. Onde se encontrava dois barbeiros considerados como processos pais e vinte clientes, que consequentemente eram os processos filhos.

E também, utilizamos memória compartilhada no programa, onde representavam as 7 cadeiras de espera no salão, tendo que indicar os clientes que entravam no salão e não tinha lugar, não podendo participar do programa como os outros processos filhos.

Não podíamos utilizar semáforo na elaboração desse programa, entretanto utilizamos semáforo apenas para melhor visualização da impressão dos resultados solicitados na função *apreciate_hair(),* deixando melhor a interpretação dos dados, que seriam o número do cliente, o número do barbeiro, o conteúdo do que se quer ordenar, o resultado da ordenação e o tempo de demora para ser atendido (desde o instante que entra e senta na sala de espera, até o momento em que começa o corte do cabelo, sendo o início de *cut_hair().*

A função *cut_hair(),* foi utilizada para realizar a ordenação do vetor, onde chamamos uma função que ordena o vetor através do método *Selection_Sort().* Podemos dizer que nosso algoritmo da parte 1 saiu como esperado, ordenando o vetor de maneira decrescente e imprimindo tudo que foi solicitado na função *apreciate_hair().* Abaixo podemos ver um começo da execução da primeira etapa.

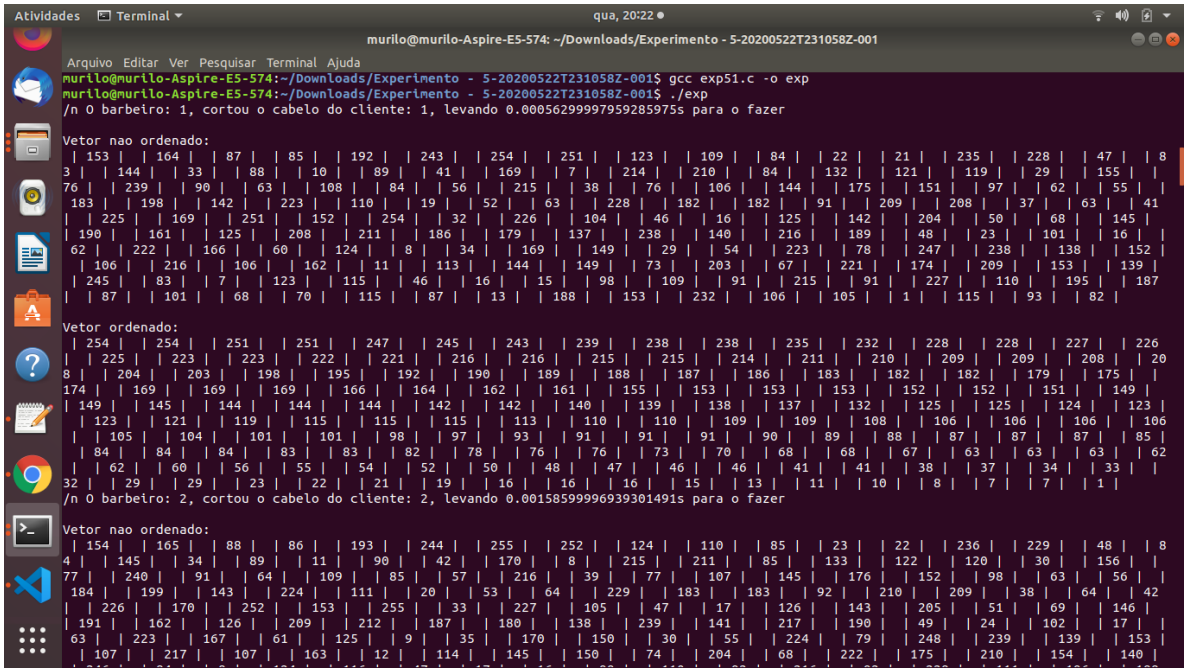

Figura 2: Execução da parte 1 do programa

5.2 Parte 2

Na segunda do Barbeiro Dorminhoco, realizamos a alteração de processos filhos, por threads, e também utilizamos mutex, ao em vez de semáforos convencionais. Também tiveram alterações nos valores, pois aumentou um barbeiro, sendo três threads para barbeiros e 27 processos para cliente, entretanto, sem alterar o valor das cadeiras de espera, permanecendo 7 cadeiras de espera. Em vez de ordenar o vetor de maneira decrescente como na primeira parte, ordenamos de maneira crescente, representando o corte do cabelo dorminhoco.

Tivemos que utilizar um vetor de semáforo, liberando o cliente somente depois que ele passasse na ordenação, ou seja, cortasse o cabelo representativamente, ocasionando assim tempo adequado para cada cliente estar pronto e com o vetor completo ordenado para continuar sua execução no programa. Como foi concluído na parte 1, podemos dizer que na parte 2 também saiu como esperado os resultados e deu tudo certo como foi idealizado, imprimindo tudo como foi solicitado na função *apreciate_hair()* também. Abaixo podemos ver um começo da execução da segunda etapa.

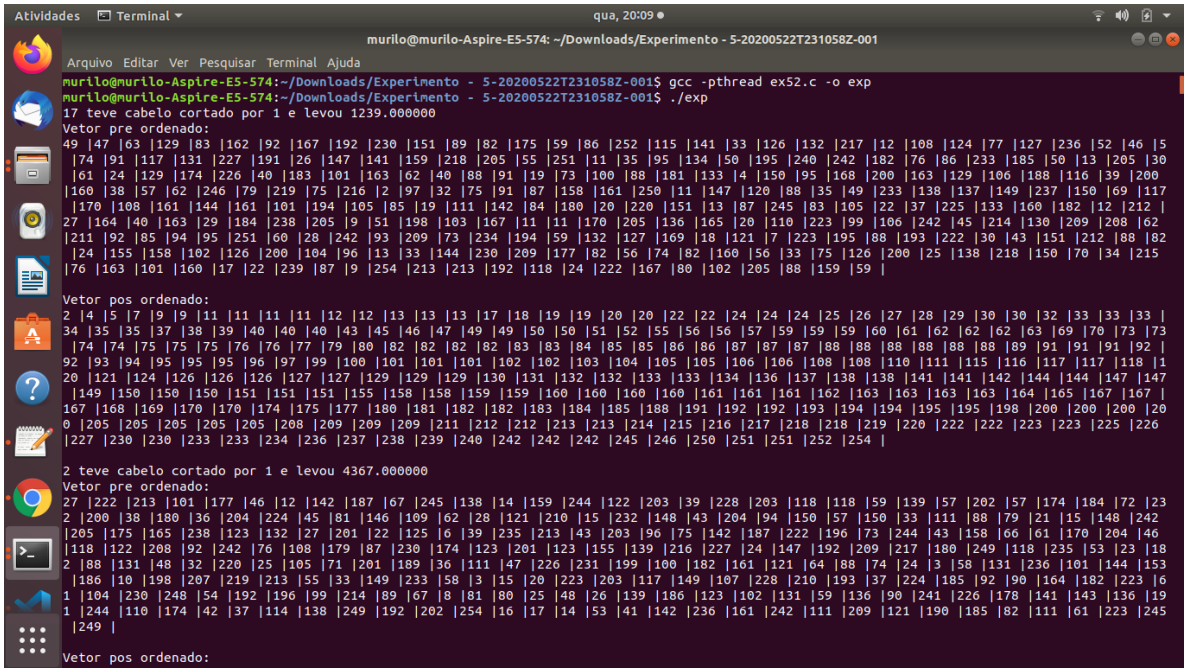

Figura 3: Execução da parte 2 do programa

6. CONCLUSÃO

Este quinto experimento realizado teve como propósito principal a aprendizagem de um novo problema para nós bem conhecido com quem trabalha com sistema operacional, que seria o problema do Barbeiro Dorminhoco, visando também a aprendizagem e aprimoramento dos mecanismos de trocas de mensagens para sincronização, mecanismo de semáforo, mecanismo de memória compartilhada, mecanismo de thread e também, de mutex, considerados mecanismos fundamentais para um sistema operacional.

Também com esse trabalho foi possível aprender mais sobre como analisar programas que envolvem compartilhamento de memória e processos paralelos, sabendo lidar com o tempo de processamento e se organizar para que o consumidor e a proposta do que for enunciado ser atendido adequadamente.

Destarte o objetivo almejado foi alcançado com êxito, correspondendo a todas expectativas de aprendizado sobre o assunto em questão, tornando nossos conhecimentos sobre processos cada vez mais diversificado, e nos ajudando a compreender como realmente tudo acontece no sistema operacional, mesclando ideias antes soltas, em algo muito mais consistente, pois tivemos a oportunidade de estar aplicando todos essas ideias e conceitos na prática em problemas dinâmicos.