**Atividade Prática de Sistemas Operacionais**

**Tácio Belmonte, Marivaldo Júnior**

1Instituto de Matemática – Universidade Federal da Bahia (UFBA)  
Bahia – Ba – Brazil.

3Departamento de Ciência da Computação  
Universidade Federal da Bahia (UFBA) – Salvador, BA – Brazil

{marivaldojr@dcc.ufba.br,taciobelmonte@dcc.ufba.br}

***Resumo.*** *A necessidade de maior poder computacional e os avanços constantes na tecnologia dos processadores fez com que a multiprogramação fosse uma técnica cada vez mais aplicada e estudada. Neste trabalho, descrevemos um problema e propomos uma solução para ele utilizando técnicas de programação paralela.*

**1. Introdução**

A multiprogramação é uma técnica utilizada para aproveitar todo o poder de processamento dos computadores atuais. Com ela é possível realizar tarefas diferentes simultaneamente, minimizando o tempo de resposta ao usuário e facilitando a solução de problemas que podem ser divididos em diferentes instâncias.

Há duas formas de implementar a programação paralela: usando Threads ou criando novos processos. Para solucionar o problema proposto, utilizamos threads. A descrição do problema bem como detalhes da solução e os resultados encontrados estão descritos nas seções a seguir.

**2. Apresentação do Problema**

O problema que propusemos para ser resolvido com o uso da programação paralela trata-se de uma busca no cliente de e-mail (específicamente o Gmail) utilizando o protocolo IMAP e a biblioteca libcurl. O software permite que o usuário faça uma busca por uma certa quantidade de mensagens (o mesmo define quantas mensagens deseja recuperar) no servidor que hospeda os seus e-mails.

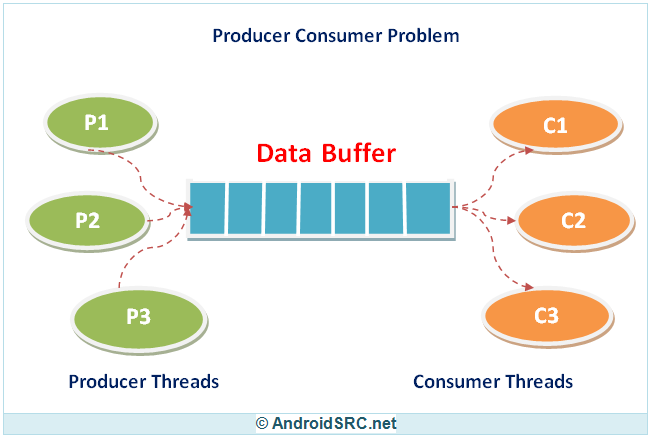
A idéia é permitir que o usuário possa fazer um backup de todas (ou algumas) as suas mensagens de forma rápida e sem a necessidade de recorrer a outros mecanismos. De uma forma simples, o usuário consegue definir quantas mensagens deseja recuperar. O programa faz então uma busca no servidor e gera um arquivo para cada e-mail contendo todo o html da mensagem, em formato txt.

Em termos técnicos, temos duas threads que podem executar em paralelo as atividades relacionadas à busca e recuperação da informação para posterior processamento.

Este modelo apresenta a área crítica como um vetor limitado, utilizado para armazenar os IDs dos e-mails. Uma thread acessa o buffer para armazenar os IDs gerados, enquanto a outra acessa para recuperar o ID e fazer a busca no servidor.

**3. Modelagem do problema e a solução adotada**

Para esse problema, optamos por modelar o problema utilizando a abordagem do produtor-consumidor. Teremos duas threads, que desempenham estes dois papéis. O produtor será responsável por gerar o identificador dos e-mails de acordo com a quantidade solicitada pelo usuário. Cada identificador gerado é armazenado no buffer pelo produtor. O consumidor, por sua vez, ficará responsável por recuperar os identificadores no buffer e então realizar a busca do e-mail no servidor utilizando o protocolo IMAP e a biblioteca libcurl. Posteriormente, os dados retornados serão utilizados para a para a criação do arquivo txt, contendo as informações relevantes do e-mail recuperado.



*Fonte: [3] Ilustração do problema do produtor-consumidor*

**4. Detecção da Região Crítica e Condições de Corrida**

No problema proposto, temos a região crítica representada por um vetor de inteiros com capacidade limitada. A condição de corrida ocorre nesse caso quando temos o produtor tentando salvar um e-mail no buffer e temos o consumidor querendo acessar o buffer para recuperar os dados para posterior processamento. O vetor, representando a área crítica nesse contexto, precisa ser acessado por todas as threads e isso gera uma condição de corrida que precisa ser tratada. Do contrário, poderíamos ter um caos na execução do programa, no qual produtor e consumidor poderiam acessar a mesma região simultâneamente. Para garantir a exclusão mútua, utilizamos semáforos, que garante que somente um processo ou thread possa acessar a região crítica, evitando assim problemas que poderiam ocorrer devido ao acesso simultâneo das threads ao recurso.

**5. Comunicação Inter-processos**

A comunicação entre os processos acontece através do uso do vetor compartilhado mencionado na seção anterior. Tanto a thread produtora como a consumidora têm acesso a essa área de memória, e o consumidor usa os dados que o produtor cria para fazer suas tarefas.

**6. Implementação da Solução**

Para resolver o problema proposto utilizamos a biblioteca libcurl, uma biblioteca em C que permite o uso de diversos protocolos de internet, entre eles os protocolos que permitem acesso a email utilizados neste trabalho (IMAP).

Já para resolver os problemas de programação paralela, utilizamos múltiplas threads e um vetor para representar o buffer. Usamos semáforos para evitar condições de corrida (foram necessários três deles), um mutex para garantir a exclusão mútua e dois outros semáforos para gerenciar o buffer cheio e vazio. Dessa forma, garantimos que um processo não executará enquanto o outro estiver usando a região crítica e que o produtor não irá adicionar itens novos no buffer caso esteja cheio e nem o consumidor remover quando estiver vazio.

**7. Análise dos Resultados**

A implementação da solução proposta para o problema nos revelou o quão poderoso é utilizar a multiprogramação para resolver problemas no qual uma grande quantidade de dados precisa ser recuperado. A programação paralela permitiu que o problema fosse particionado e resolvido por múltiplas threads, o que tornou a resolução mais rápida, quando comparada à utilização de técnicas tradicionais, sem o uso da multiprogramação.

**8. Limitações e melhorias**

A idéia seria permitir que o usuário pudesse buscar por mensagens utilizando filtros e também retornar os arquivos anexos aos e-mails, ao invés de somente salvar a string dos e-mails em arquivo. Contudo, devido a falta de documentação específica para a realização dessas tarefas, optamos por manter a abordagem mais simplificada e genérica. Seria interessante também se o retorno dos dados fossem formatados, sem as tags html, o que não ocorre nessa versão.

**Referências**

[1] - Tanenbaum, A. S. (2009), Sistemas Operacionais Modernos, Pearson, 3rd edition.

[2] - Libcurl Website, Disponível em <https://curl.haxx.se/libcurl/>, acessado em 23 de Setembro de 2016.

[3] - androidsrc.net. 2015. *Producer Consumer Problem Jav: Detailed Explanation*[ONLINE] at <http://androidsrc.net/producer-consumer-problem-java-detailed-explanation> [Accessed 3 October 2016]