Implementação do produtor-consumidor

Aplicação Stalker

Marcelo Henrique Bittencourt
Curso de Ciência da Computação, CCET
Universidade Estadual do Oeste do Paraná
Cascavel, Brasil
marcelohbittencourt@outlook.com

Matheus de Lara Dias da Silva
Curso de Ciência da Computação, CCET
Universidade Estadual do Oeste do Paraná
Cascavel, Brasil
matheus.vex@gmail.com

Nicolas Afonso Bertaglia Comissio
Curso de Ciência da Computação, CCET
Universidade Estadual do Oeste do Paraná
Cascavel, Brasil
nicolasbertaglia@gmail.com

Renan Tashiro
Curso de Ciência da Computação, CCET
Universidade Estadual do Oeste do Paraná
Cascavel, Brasil
renantashiro@hotmail.com

I. Introdução

Este trabalho tem como objetivo implementar uma aplicação produtor-consumidor. Aplicações produtor-consumidor se caracterizam por múltiplas threads utilizando os mesmos objetos, de modo que instruções críticas sejam realizadas de forma sincronizada entre as threads.

Criamos uma aplicação chamada Stalker, que realiza dois tipos de tarefas ao mesmo tempo: baixa páginas do site URI Online Judge (produtor) e extrai informações relevantes das páginas (consumidor). A aplicação foi implementada utilizando a linguagem de programação Python 3.

II. Descrição do problema

O problema do produtor-consumidor consiste em múltiplos processos que compartilham o mesmo buffer. O produtor insere dados no buffer, enquanto o consumidor retira dados do buffer [2].

Quando múltiplos processos acessam os mesmos dados simultaneamente, existe a possibilidade de ocorrer uma troca de contexto durante a atualização dos dados. Como instruções de escrita em memória podem não ser atômicas, uma eventual troca de contexto poderia fazer com que processos diferentes utilizassem cópias diferentes dos mesmos dados, gerando inconsistência nos mesmos.

Cada linguagem de programação resolve o problema do acesso concorrente de forma diferente. Internamente, as linguagens chamam uma instrução atômica do processador, durante a qual é garantido que não haverá troca de contexto entre as threads do mesmo processo. Para o programador, isso pode ser abstraído de diferentes formas: mutex, semáforos, métodos sincronizados, etc.

III. Descrição das implementações

A aplicação criada, Stalker, implementa uma solução para o problema do produtor-consumidor e tem como objetivo gerar estatísticas a partir das informações disponíveis no site URI Online Judge. Como o download (produção) e a análise (consumo) das páginas do site podem demorar, a utilização de múltiplas threads para realizar essas tarefas pode tornar o processo muito mais rápido.

A aplicação utiliza duas filas: a primeira para armazenar as URLs das páginas a serem baixadas e a segunda para armazenar as páginas a serem analisadas. As filas utilizam a classe queue da linguagem Python, que implementa métodos com acesso sincronizado [1].

Por enquanto, o consumidor apenas extrai informações gerais dos perfis dos usuários do site. Após a entrega do trabalho, existe a possibilidade de se adicionar novas funcionalidades à aplicação.

A. Pré-processamento

Antes de iniciar as threads para produção e consumo, a aplicação lê as URLs dos perfis a serem visitados de um arquivo de texto previamente preenchido. Além disso, o programa marca o tempo de início para analisar o desempenho do processo.

B. Produção

O produtor realiza as seguintes atividades: lê a próxima URL da fila de URLs para baixar, baixa a página usando protocolo HTTPS, coloca a página baixada na fila de páginas para analisar e retira a URL da fila de URLs.

C. Consumo

O consumidor realiza as seguintes atividades: lê a próxima página da fila de páginas para analisar, analisa a página de

acordo com as *tags* HTML, armazena os dados obtidos através da análise da página e retira a página da fila de páginas.

D. Pós-processamento

A aplicação calcula o tempo de execução do processo e encerra todas as threads. Depois, o programa imprime os dados obtidos dos perfis de usuários num arquivo JSON.

IV. Análise dos resultados

Ao executarmos a aplicação com diferentes números de produtores e consumidores, obtivemos os seguintes resultados:

TABELA I. TESTES DE DESEMPENHO

Produtores	Consumidores	Páginas	Tempo (s)
1	1	22	72
5	11	22	18
11	1	22	15
22	1	22	7
22	11	22	7

A partir da análise dos resultados dos testes, percebemos que o processo executado de forma sequencial é muito mais lento do que com múltiplas threads. O processo com uso de poucas threads de produção e consumo também se mostrou mais lento do que com uso de muitas threads.

No entanto, percebemos que o aumento no número de threads de consumo não resultou num aumento significativo no desempenho, considerando que a análise das páginas costuma ser feita rapidamente. Já o aumento no número de threads de produção diminuiu consideravelmente o tempo de execução do processo, considerando que o download de páginas da internet é a parte mais lenta do processo devido a limitações da rede (velocidade e latência).

V. Conclusão

Concluímos que o problema do produtor-consumidor pode ser usado para implementar aplicações de análise de páginas da internet. Ao utilizarmos múltiplas threads para baixar as páginas da internet, estamos contornando as dificuldades relacionadas ao desempenho da rede (velocidade e latência). Além disso, a utilização de múltiplas threads para baixar e analisar as páginas da internet faz um melhor uso do processador, principalmente no caso de processadores com múltiplos núcleos.

Referências

[1] "queue — A synchronized queue class", Python 3.6.1 Documentation. Beaverton, OR: Python Software Foundation, 2017. Disponível em: <https://docs.python.org/3/library/queue.html>.

[2] R. Bocchi, "O produtor e o consumidor", Viva o Linux. Nova Friburgo, RJ: 2010. Disponível em: <https://www.vivaolinux.com.br/artigo/O-Produtor-e-o-Consumidor>.