Relatório do trabalho prático da matéria de sistemas operacionais (SOP)

Guilherme M. Utiama[1], **Peter Brendel**[1]

[1]Ciência da computação – Universidade do Estado de Santa Catarina (UDESC)
Santa Catarina – SC – Brazil

utiama.guilherme@gmail.com, peter.brendel@edu.udesc.br

1. Introdução

Este projeto foi desenvolvido com base nos documentos e códigos apresentados pelo professor Rafael Obelheiro, utilizando a linguagem C++ e as funções fornecidas pela biblioteca pthreads. O problema sugerido é criar um apurador de votos usando threads. Os dados de entrada são fornecidos através arquivos texto. Nestes arquivos as informações são: quantidade de threads a serem criadas, numero e nome do candidato e um arquivo com os votos. No trabalho, é dever dos programadores cuidar da validade das entradas, isto é, o número do candidato poderia não ser válido. Com isso, o código foi desenvolvido para coletar as informações e mostrar estatísticamente a porcentagem de votos de cada candidato, assim como os votos inválidos.

2. Controle de concorrência

A concorrência ocorre na fila de votos, problema tratada com mutex, garantindo a exclusão mútua na inserção e remoção de dados. Neste programa a variável mutex é representada por **mtx**. Outro problema está presente, a espera ocupada da thread responsável por ler os dados da fila. Caso a fila esteja vazia, a thread não deve ocupar poder de processamento, isto foi resolvido através de uma variável de condição denominada **emptyq**. Quando a fila está vazia, a thread dorme e aguarda através da função *pthread_cond_wait*() a sinalização da variável.
Duas barreiras foram utilizadas para manter o controle de execução do programa. A primeira é **barrier_init**, responsável por aguardar a preparação de todas as threads e a outra **barrier_end** é responsável por aguardar a finalização de todas as threads.
Ao ultrapassar uma barreira, as threads sinalizam os apuradores de voto que não existem mais votos para serem adicionados a fila, desta maneira, quando atingem o fim podem finalizar a execução ao invés de aguardar novas entradas.

3. Conclusão

Ao fim do desenvolvimento deste software foi possível identificar as características da programação multithreading, é preciso ter muito cuidado ao tratar erros para evitar a ocorrência de *deadlocks* e esperas ocupadas. Foi notável também o fato da programação paralela fornecer mais desempenho quando comparado com a execução de um programa serial.