UNIVERSIDADE FEDERAL DE SÃO CARLOS
Bacharelado em Ciência da Computação
Laboratório de Sistemas Operacionais
Professor Gustavo Maciel Dias Vieira
***Campus* Sorocaba**

Projeto 4
Programação Concorrente

Daniel Ramos Miola 438340
Giulianno Raphael Sbrugnera 408093

Sorocaba
2013

1. Introdução

O Projeto 4 tem por objetivo criar um programa concorrente utilizando threads para resolver o problema de diminuir o tempo gasto para quebrar um hash de senha de quatro caracteres.

2. Discussão e Resultados

Ao longo do projeto, houve um total de quatro tarefas a serem realizadas, as quais seus resultados são discutidos abaixo.

1. Baixar, compilar e executar o programa exemplo fornecido. Estudar o seu funcionamento.

A partir do programa fornecido, este foi compilado e executado. O programa em questão utiliza força bruta para quebrar a senha de quatro caracteres. Desta forma, seu desempenho é relativamente baixo, mesmo para uma senha de tamanho pequeno.

2. Medir o tempo que este programa leva para quebrar o *hash* "aaRCVPtrkrWUY".

Os testes foram feitos em um notebook com processador Intel® Core™ i5-3210M CPU @ 2.50GHz×4 no sistema operacional Fedora 19 de 64 bits. O programa fornecido para o projeto levou 43.361 segundos para quebrar a referida *hash*, a qual possui a senha "*raio*".

3. Alterar o programa, adicionando suporte para múltiplas *threads*.

Para alterar o programa e executá-lo em múltiplas *threads* utilizamos a organização produtor/consumidor como consta no código fornecido para o projeto, de modo que o produtor incrementa as senhas e o consumidor testa as senhas geradas.
Foram usadas como comunicação entre as *threads* variáveis globais que guardam as senhas geradas pelo produtor em um vetor que é usado como uma fila circular de 50 posições, além de suas variáveis de controle de posição, variáveis para o controle do semáforo que será usado para o acesso às regiões críticas do código e uma variável que guarda a senha que está sendo incrementada para comparação.

```
// Variaveis globais que facilitam o compartilhamento de dados
// fila circular compartilhada pelas threads
char *shared_passwd[N_ITENS];

// senha que sera testada e incrementada caso necessario
char senha[TAM_SENHA + 1];

// ponteiro para char que guarda o valor da hash fornecida
const char *target;

// definem o ultimo elemento consumido e o ultimo elemento produzido, respectivamente
int inicio = 0, final = 0;

sem_t bloqueio, pos_vazia, pos_ocupada;
```

Imagem 1. Variáveis globais para comunicação entre processos.

Para a execução na máquina de testes utilizamos um produtor que incrementa as senhas e as guarda na fila circular com 50 posições, e 3 consumidores que obtêm essas senhas, geram a hash e comparam com a hash a ser encontrada. Essa abordagem foi utilizada para gerar 4 *threads* que ocupam todas as *threads* do núcleo na máquina em que foram feitos os testes, onde apenas 1 produtor consegue suprir os consumidores já que a tarefa de incrementar a senha a ser testada é bem mais rápida que gerar o *hash* e comparar.

```
pthread_create(&thr_produtor, NULL, produtor, NULL);
pthread_create(&thr_consumidor1, NULL, consumidor, NULL);
pthread_create(&thr_consumidor2, NULL, consumidor, NULL);
pthread_create(&thr_consumidor3, NULL, consumidor, NULL);

pthread_join(thr_produtor, NULL);
pthread_join(thr_consumidor1, NULL);
pthread_join(thr_consumidor2, NULL);
pthread_join(thr_consumidor3, NULL);
```

Imagem 2. Trecho do código responsável por criar as *threads* e esperar pelo término das mesmas.

4. Medir o tempo que o programa *multithreaded* leva para quebrar o *hash* "aaRCVPtrkrWUY".

O tempo obtido com o programa *multithreaded* foi de 21.602 segundos, um tempo duas vezes mais rápido que o *singlethreaded*.

3. Dificuldades encontradas na realização

As principais dificuldades encontradas durante a realização foi utilizar o conceito de semáforos para garantir o uso ininterrupto da memória compartilhada entre as *threads*, ou seja, garantir a exclusão mútua e definir as funções de produtor e consumidor no código, de modo que o projeto inicial contava com apenas duas *threads* que realizavam a tarefa em um tempo similar ao programa exemplo e com os testes foram inseridos mais *threads* consumidoras que diminuíram o tempo de execução consideravelmente.

4. Conclusão

Programas *multithreaded* levam a vantagem de possuir diversos segmentos de seu código sendo executados ao mesmo tempo, minimizando o tempo total de execução gasto. No contexto específico do programa de quebrar senhas, a programação concorrente melhora significativamente a rapidez do algoritmo, por se tratar de um algoritmo que utiliza força bruta para atingir o resultado. Ao mesmo tempo, é necessário garantir exclusão mútua para que a memória compartilhada entre as *threads* não seja modificada de tal forma que o resultado final fique comprometido.

5. Bibliografia

[1] Slides do curso Laboratório de Sistemas Operacionais – UFSCar Sorocaba

[2] Slides do curso Sistemas Operacionais – UFSCar Sorocaba