

Pontifícia Universidade Católica de Campinas Faculdade de Engenharia de Computação - FECOMP

Sistemas Operacionais A – Relatório Experimento 4

Beatriz Morelatto Lorente RA: 18071597

Cesar Marrote Manzano RA: 18051755

Fabricio Silva Cardoso RA: 18023481

Pedro Ignácio Trevisan RA: 18016568

**Sumário**

**1.Introdução.......................................................................................................3**

**2.Apresentação dos erros do programa exemplo e suas soluções.............4**

**3.Respostas das perguntas..............................................................................7**

**4.Resultados da execução do programa exemplo**

**5.Resultados da execução do programa modificado**

**6.Análise dos Resultados**

**7.Conclusão**

**Introdução**

O experimento realizado permitiu o entendimento do uso de threads na linguagem de programação C, como forma de se obter concorrência em um programa. O experimento foi divido em duas tarefas.

Na primeira tarefa, foi executado um programa exemplo, ilustrando o problema do produtor consumidor. Havia 10 threads produtoras e 10 consumidoras, que compartilham um buffer, percorrido de maneira circular. As threads produtoras armazenam o valor 10 no buffer, e as consumidoras retiram esse valor. O problema dessa tarefa está em como manipulamos os acessos a esse buffer, de modo que as condições de corrida ocorram o menor número de vezes possível.

Na segunda tarefa, é proposto a solução do problema do jantar das filósofas (adaptação do problema clássico do jantar dos filósofos). No problema, cada filósofa é uma thread, e cada uma irá comer no máximo 365 vezes. No lugar do estado de “pensando”, foi programado uma espera de 25 microssegundos. A tarefa foi importante para observarmos o uso do mutex, para garantir exclusão mútua.

**Apresentação dos erros do programa exemplo e suas soluções**

Ao compilar o programa exemplo foi possível perceber alguns erros de sintáxe e lógica do programa. Os problemas estão listados abaixo, seguidos de suas soluções (as correções estão destacadas em negrito e itálico).

**Problema 1**

#include <pthread.h>

#include <stdio.h>

**Problema corrigido:**

Falta algumas bibliotecas para a compilação do programa.

#include <pthread.h>

#include <stdio.h>

***#include <stdlib.h>***

**Problema 2 e 3 (na função main)**

tc = pthread\_create(&consuers[i], NULL, consume, (void \*) i + 1);

tc = pthread\_create(&consuers[i], NULL, produce, (void \*) i + 1);

**Problema corrigido:**

Era necessário copiar o valor de i + 1 para um vetor. Assim iniciamos declarando dois vetores, um para cada criação da thread.

***int temp1[NUM\_THREADS], temp2[NUM\_THREADS];***

Depois jogamos o valor de i + 1 para a posição correspondente e passamos a posição do vetor como parâmetro da função.

***temp1[i] = i + 1;***

***tc = pthread\_create(&consuers[i], NULL, consume, (void \*)&temp1[i]);***

***temp2[i] = i + 1;***

***tc = pthread\_create(&consuers[i], NULL, produce, (void \*)&temp2[i]);***

Desse jeito foi possível eliminar o ‘warning’ existente ao compilar o programa exemplo.

O problema 2 consiste na falta do uso da função ‘pthread\_join()’, fazendo com que a função main só termine a sua execução depois que todas as threads produtoras e consumidoras terminarem.

for(i = 0; i < NUM\_THREADS; i++){

pthread\_join(producers[i], NULL);

pthread\_join(consumers[i], NULL);

}

**Problema 4 e 5 (na função consume)**

printf(“Soma do que foi consumido pelo Consumidor #%d : %d\n”, threadid, sum);

while(cont\_c > NO\_OF\_ITERATIONS)

**Problemas Corrigidos:**

No primeiro problema a variável ‘threadid’, estava sendo passada como parâmetro para a função, porém não estava sendo convertida em inteiro.

***int \*t\_id = threadid;***

***printf(“Soma do que foi consumido pelo Consumidor #%d : %d\n”, \*t\_id, sum);***

Para o segundo problema foi necessário trocar apenas o sinal de ‘>’ para ‘<’.

***while(cont\_c < NO\_OF\_ITERATIONS)***

**Problema 6 ,7 e 8 (na função produce)**

printf(“Soma do que foi consumido pelo Consumidor #%d : %d\n”, threadid, sum);

if(ret){

cont\_p+-;

sum - = 10;

}

**Problemas corrigidos:**

A correção do problema 6 é similar ao do problema 4.

***int \*t\_id = threadid;***

***printf(“Soma do que foi consumido pelo Consumidor #%d : %d\n”, \*t\_id, sum);***

Para a correção do problema 7 e 8, foi alterado o sinal de ‘-‘ para o sinal de ‘+’.

***if(ret){***

***cont\_p++;***

***sum + = 10;***

***}***

**Problema 9 (na função myremove)**

**Problema 10 (na função myadd)**

**Respostas das perguntas**

**Perguntas do relatório**

**Perguntas do programa**

***Pergunta 1: Porque ‘ret’ não está sendo comparado a algum valor?***

**Resposta:** Pois a função ‘myadd()’ é responsável por validar o parâmetro passado, no caso do programa, o 10. Esse controle é feito a partir do retorno da função, sendo 1 para quando deu certo e 0 para quando deu errado. Se o valor de ret for 1 o programa entra na função, não sendo necessário nenhuma comparação.

***Pergunta 2: Porque não há necessidade de um cast?***

**Resposta:** Pois o retorno da função é do tipo inteiro (0 ou 1), não sendo necessário indicar para a variável um novo tipo.

***Pergunta 3: Para que serve cada um dos argumentos usados com pthread\_create?***

**Resposta:**

• O primeiro parâmetro (&consumers[i]) é usado para guardar as informações sobre a thread criada;

• O segundo parâmetro (NULL) é usado para indicar algumas especificações sobre a thread criada. Quando definido como NULL a thread é criada com os atributos padrões definidos pelo sistema;

• O terceiro parâmetro (consume) é a função que a thread irá executar;

• O quarto parâmetro ((void \*)i+1) são os argumentos de início da thread.

***Pergunta 4: O que ocorre com as threads criadas, se ainda estiverem sendo executadas e a thread que as criou termina através***

***de um pthread\_exit()?***

**Resposta:** Se a thread responsável por criar outros threads for finalizada, as threads criadas também serão finalizadas. Isso também ocorre caso ocorra um erro em alguma thread, assim todas as outras irão dar erro e consequentemente o programa irá finalizar sua execução. Pthread\_exit() é responsável por liberar todos os recursos ligados a thread finalizada.

***Pergunta 5: Idem questão anterior, se o termino se dá através de um exit()?***

**Resposta:** Se o término ocorre pelo exit(), a thread criadora terá um valor de retorno que será passado para as outras threads que ela mesmo criou, não gerando erro e esperando o término das outras threads.

**Resultados da execução do programa exemplo**

**Resultados da execução do programa modificado**

**Análise dos resultados**

**Conclusão**