Universidade Federal de Santa Catarina
Centro de Ciências Tecnológicas
INE5645-05238

RELATÓRIO SOBRE PROBLEMAS DE SINCRONIZAÇÃO

Aluno(a): Bruno Daniel Elias,Eduardo Bruggmann Pedro, Mariany Ferreira da Silva
Professor(a): Odorico Machado Mendizabal

Sobre o problema Cigarette Smokers

O problema dos cigarros tem quatro threads associadas a ele, um agente e três fumantes. Os fumantes ficam em um loop infinito, primeiro esperando pelos ingredientes e depois fazendo e fumando os cigarros. Os ingredientes são, tabaco, papel e fósforos.
O agente tem um suprimento infinito dos três ingredientes e cada fumante tem um suprimento infinito apenas de um ingrediente. O agente escolhe dois diferentes ingredientes aleatórios e os disponibiliza aos fumantes.

Dependendo de quais ingredientes foram disponibilizados o fumante que tiver o ingrediente complementar deve pegá-los, fazer um cigarro e fumar. Cada fumante só pode pegar os ingredientes se tiver o ingrediente complementar consigo.

Pseudo-código para resolver o problema

```
função fumante(item) {
        fazer_cigarro(item)
        fumar_cigarro()
        passar_a_vez()
}

função agente() {
        verificar_items_na_mesa()
        se (mesa == limpa) gerar_item_aleatório_0_2()
        limpar_mesa()
}

função main() {
        while(true) {
                esperar_agente()
                agente()
                se (mesa == cheia) {
                        se (item == tabaco) fumante(tabaco) senão passar_a_vez()
                        se (item == fósforo) fumante(fósforo) senão passar_a_vez()
                        se (item == papel) fumante(papel) senão passar_a_vez()
                }
        }
}
```

Programa em C para resolver o problema

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <omp.h>

void build_and_smoke_cigarette(int thread_number);

int main(int argc, char *argv[]) {

   int paper = 0, matches = 0, tobacco = 0;

    #pragma omp parallel sections default(none) shared(paper, matches,
tobacco)
   {
       // SECTION AGENT
       #pragma omp section
       {
           // -- RAND 2 ITEMS FOREVER
           // ---- WHILE ITEMS IN THE TABLE
           // ------- WAIT
           int r = -1;
           do {
               if(paper + matches + tobacco == 0) {
                   r = rand() % 3;

                   // MATCHES & TOBACCO
                   if(r == 0) {
                       printf("\n[AGENT] GIVE MATCHES & TOBACCO\n");
                       matches = 1;
                       tobacco = 1;
                   }
                   // PAPER & TOBACCO
                   if(r == 1) {
                       printf("\n[AGENT] GIVE PAPER & TOBACCO\n");
                       paper = 1;
                       tobacco = 1;
                   }
```

```
                // MATCHES & PAPER
                if(r == 2) {
                    printf("\n[AGENT] GIVE MATCHES & PAPER\n");
                    matches = 1;
                    paper = 1;
                }
            }
        } while (1);
    }

    // SMOKER PAPER
    #pragma omp section
    {
        // -- TABLE HAS MATCHES & TOBACCO?
        // -- BUILD CIGARETT
        // ---- TIMER 2 SEC
        // -- SMOKE
        // ---- TIMER 2 SEC
        do {
            if(tobacco > 0 && matches > 0) {
                build_and_smoke_cigarette(omp_get_thread_num());

                tobacco = 0;
                matches = 0;
            }
        } while (1);
    }

    // SMOKER MATCHES
    #pragma omp section
    {
        // -- TABLE HAS PAPER & TOBACCO?
        // -- BUILD CIGARETT
        // ---- TIMER 2 SEC
        // -- SMOKE
        // ---- TIMER 2 SEC
        do {
            if(tobacco > 0 && paper > 0) {
                build_and_smoke_cigarette(omp_get_thread_num());

                tobacco = 0;
                paper = 0;
            }
```

```
            } while (1);
        }

        // SMOKER TOBACCO
        #pragma omp section
        {
            // -- TABLE HAS MATCHES & PAPER?
            // -- BUILD CIGARETT
            // ---- TIMER 2 SEC
            // -- SMOKE
            // ---- TIMER 2 SEC
            do {
                if(paper > 0 && matches > 0) {
                    build_and_smoke_cigarette(omp_get_thread_num());

                    matches = 0;
                    paper = 0;
                }
            } while (1);
        }
    }

    return 0;
}

void build_and_smoke_cigarette(int thread_number) {
    printf("\n[SMOKER] %d building a cigarette\n", thread_number);
    sleep(1);
    printf("\n[SMOKER] %d smoking\n", thread_number);
    sleep(1);
}
```

Sobre o programa criado

O agente citado no problema é implementado com um função while True que verifica se existem itens na mesa, caso não tenha, roda um função rand para disponibilizar de forma aleatória dois itens. Os itens disponibilizados são printados no console.
Cada fumante também roda uma função while True para verificar os itens disponibilizados na mesa pelo agente, caso os itens disponibilizados sejam os que ele precisa, uma função para fazer o cigarro e fumá-lo é chamada.

A função build_and_smoke_cigarette precisa de um número inteiro como parâmetro de entrada e, quando executada, faz prints no console sinalizando que está fazendo o cigarro e posteriormente fumando.

Optamos por fazer uso da API OpenMP para prover paralelismo ao código, sendo assim, o agente e cada um dos fumantes executa em uma thread distinta. Foram utilizadas seções com variáveis compartilhadas, o agente e cada um dos fumantes estão em uma seção paralela separada e as variáveis compartilhadas são paper, matches, tobacco. Cada thread manipula as variáveis compartilhadas e pode modificá-las. O agente modifica o valor das variáveis para 1 de acordo com os itens disponibilizados e cada fumante modifica para o valor 0 quando pega os itens.

Testes do programa e explicações sobre suas saídas

Nesta saída do programa, a thread agente disponibiliza papel e tabaco e a thread fumante que possui os fósforos pega os itens, faz o cigarro e fuma. O valor (zero neste caso) indica o número da thread que foi associada ao fumante durante a execução do programa.

```
[AGENT] GIVE PAPER & TOBACCO

[SMOKER] 0 building a cigarette

[SMOKER] 0 smoking
```

Neste outro exemplo de saída do programa, a thread agente disponibiliza fósforos e tabaco e a thread fumante que possui o papel pega os itens, faz o cigarro e fuma. O valor (cinco neste caso) indica o número da thread que foi associada ao fumante durante a execução do programa.

```
[AGENT] GIVE MATCHES & TOBACCO

[SMOKER] 5 building a cigarette

[SMOKER] 5 smoking
```

Como nos exemplos anteriores, a thread agente disponibiliza fósforos e papel e a thread fumante que possui tabaco pega os itens, faz o cigarro e fuma. O valor (quatro neste caso) indica o número da thread que foi associada ao fumante durante a execução do programa.

```
[AGENT] GIVE MATCHES & PAPER
[SMOKER] 4 building a cigarette
[SMOKER] 4 smoking
```

Com esta saída do programa é possível verificar que cada fumante tem sempre a mesma thread associada à ele, o fumante que tem o valor zero sempre espera o agente disponibilizar papel e tabaco, enquanto o fumante com o valor cinco sempre espera fósforos e tabaco.
Esses valores associados aos fumantes permanecem fixos durante a execução do programa, porém quando o programa é encerrado e posteriormente executado novamente os valores podem mudar, pois threads diferentes podem ser alocadas.

```
[AGENT] GIVE PAPER & TOBACCO
[SMOKER] 0 building a cigarette
[SMOKER] 0 smoking
[AGENT] GIVE PAPER & TOBACCO
[SMOKER] 0 building a cigarette
[SMOKER] 0 smoking
[AGENT] GIVE MATCHES & TOBACCO
[SMOKER] 5 building a cigarette
[SMOKER] 5 smoking
[AGENT] GIVE PAPER & TOBACCO
[SMOKER] 0 building a cigarette
[SMOKER] 0 smoking
[AGENT] GIVE MATCHES & PAPER
[SMOKER] 4 building a cigarette
[SMOKER] 4 smoking
```