Relatório EP 4 · SO

Prof. Dra. Gisele da Silva Craveiro

Mikael Gi Sung Shin, 10843441

Código e Explicação:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define tam 5

pthread_t filosofo[tam];
pthread_mutex_t garfo[tam];
```

1) Para a implementação da solução do problema do "Jantar dos Filósofos" em C, precisaremos incluir as bibliotecas descritas acima e definir o "tam" como 5, pois este problema se trata de 5 threads/filósofos e 5 recursos/garfos (utilizaremo-os como mutexes, um mutex para cada recurso, pois ele só poderá ser liberados pelo processo que o adquiriu). Além disso, foi necessário declarar esses 2 arranjos globais (das threads e dos mutexes) para serem utilizados em todos os métodos da classe.

```c
int main()
{

    int arg[tam];

    for(int i = 0; i < tam; i++){

        pthread_mutex_init(&(garfo[i]), NULL);
    }

    for(int i = 0; i < tam; i++){

        arg[i] = i;
        pthread_create(&filosofo[i], NULL, acaoFilosofo, (void*)&arg[i]);
    }

    for(int i = 0; i < tam; i++){

        pthread_join(filosofo[i], NULL);
    }

    return 0;
}
```

2) O método main() faz as inicializações e criações dos mutexes e das threads, e comanda as threads executarem o método acaoFilosofo(...).

```
10
11  void garfosIndisponiveis(int *nFilosofo)
12  {
13      pthread_mutex_lock(&(garfo[*nFilosofo]));
14      pthread_mutex_lock(&(garfo[(*nFilosofo + 1) % 5]));
15  }
16
17  void garfosDisponiveis(int *nFilosofo)
18  {
19      pthread_mutex_unlock(&(garfo[*nFilosofo]));
20      pthread_mutex_unlock(&(garfo[(*nFilosofo + 1) % 5]));
21  }
22
```

3) Quando um único recurso está sendo alocado para o processo 'x', um outro processo 'y' não poderá recorre-lo. Deve-se, pois, esperar tal recurso estar disponível para o uso.
É essa lógica que foi implementado nos dois métodos acima. O garfosIndisponiveis(...) será chamado quando o filósofo/thread decidir comer e utilizar os dois garfos que estão à sua disposição, e então o mutex será atualizado. O garfosDisponiveis(...) retrata o cenário contrário, ele será chamado quando um filósofo para de comer, tornando os garfos disponíveis novamente.
obs: Para o filósofo de índice 1, estão em sua disposição os garfos de índice 1 e 2. Para o filósofo de índice 2, os garfos de índice 2 e 3. (...). Para o filósofo de índice 4, os garfos 4 e 0.

```
23  void *acaoFilosofo(void *arg){
24
25      int *nFilosofo = (int *) arg;
26      printf("Filosofo #%d: Entrou no método.\t", *nFilosofo);
27
28      while(1){
29
30          // bloco para pensar = não usa o recurso (garfo)
31          int tempoPensando = (rand()%16);
32          printf("Filosofo #%d: Pensando por %ds.\n", *nFilosofo, tempoPensando);
33          sleep(tempoPensando);
34
35          // bloco para pensar = usa o recurso (garfo)
36          garfosIndisponiveis(nFilosofo);
37          int tempoComendo = (rand()%16);
38          printf("Filosofo #%d: Comendo por %ds.\n", *nFilosofo, tempoComendo);
39          sleep(tempoComendo);
40          garfosDisponiveis(nFilosofo);
41      }
42  }
43
```

4) Primeiramente, é impressa uma mensagem de sucesso à entrada da thread #n no método. Em seguida, os threads caem num loop infinito, onde será executada a ação de pensar e comer. No bloco "pensar", delimitado pelos comentários, foi criada uma variável que recebe um número aleatório menor que 16 e que servirá como tempo de execução desta ação. Depois, passa-se este valor como parâmetro da função "sleep(...)" que simulará o tempo gasto da ação "pensar" por parte dos filósofos/threads.

A mesma lógica foi implementada para o bloco “comer”, recebe um número randômico, printa um texto referindo-se a execução desta ação e passa-se como parâmetro da função “sleep(...)”, o número recebido. Porém, foi incluído os métodos para disponibilizar e indisponibilizar os recursos/garfos, quando os filósofos começam a comer e param de comer. Caso haja um filósofo que esteja comendo, os outros dois filósofos que se encontram nos lados, não poderão comer, uma vez que o garfo já está sendo usado. Isto é, as threads não conseguem entrar na região crítica, pois não há recurso em sua disposição.

Saída:

```
Filosofo #0: Entrou no método.  Filosofo #0: Pensando por 7s.
Filosofo #1: Entrou no método.  Filosofo #1: Pensando por 6s.
Filosofo #2: Entrou no método.  Filosofo #2: Pensando por 9s.
Filosofo #3: Entrou no método.  Filosofo #3: Pensando por 3s.
Filosofo #4: Entrou no método.  Filosofo #4: Pensando por 1s.
Filosofo #4: Comendo por 15s.
Filosofo #1: Comendo por 10s.
Filosofo #1: Pensando por 12s.
Filosofo #4: Pensando por 9s.
Filosofo #0: Comendo por 13s.
Filosofo #3: Comendo por 10s.
Filosofo #3: Pensando por 11s.
Filosofo #2: Comendo por 2s.
Filosofo #2: Pensando por 11s.
Filosofo #0: Pensando por 3s.
Filosofo #1: Comendo por 6s.
Filosofo #4: Comendo por 12s.
Filosofo #1: Pensando por 2s.
Filosofo #1: Comendo por 4s.
Filosofo #4: Pensando por 8s.
Filosofo #3: Comendo por 11s.
Filosofo #1: Pensando por 8s.
Filosofo #0: Comendo por 7s.
Filosofo #0: Pensando por 13s.
```

Pelos comentários/explicações feitos anteriormente, a regra é clara: os filósofos que se encontram diretamente ao lado do filósofo que já está comendo não poderão comer. Podemos ver pelos outputs acima, que a implementação obedece esta regra.

obs: dois filósofos podem comer ao mesmo tempo, contanto que a regra se aplique. Existem 3 exemplos que compõem essa situação.

Instruções para compilação:

1) Usar alguma IDE que suporte a linguagem C, e executar (“run”) o código descrito acima. (obs: utilizei o compilador online https://www.onlinegdb.com/online_c_compiler).
2) Executar as seguintes linhas de código em algum terminal, dentro do diretório do programa:
   `gcc “nomedoarquivo”.c -o “nomedoarquivo`
   `./“nomedoarquivo”`

Referências para o desenvolvimento do código:

1) https://www.youtube.com/watch?v=G0ZCndqb0xk&t=75s&ab_channel=Lu%C3%ADsPauloBravin
2) https://www.youtube.com/watch?v=JaXr15a5cPA&list=PLxI8Can9yAHeK7GUEGxMsqoPRmJKwI9Jw&index=12&ab_channel=UNIVESP
3) http://wiki.icmc.usp.br/images/7/76/Aula06_2.pdf
4) Fundamentos de Sistemas Operacionais de Abraham Silberschatz.