UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO (ICMC)

BERNARDO MAIA COELHO
Nº USP: 12542481

Professor: Vanderlei Bonato
Disciplina: SSC0140 - Sistemas Operacionais I
Curso: Bacharelado de Ciência de Computação

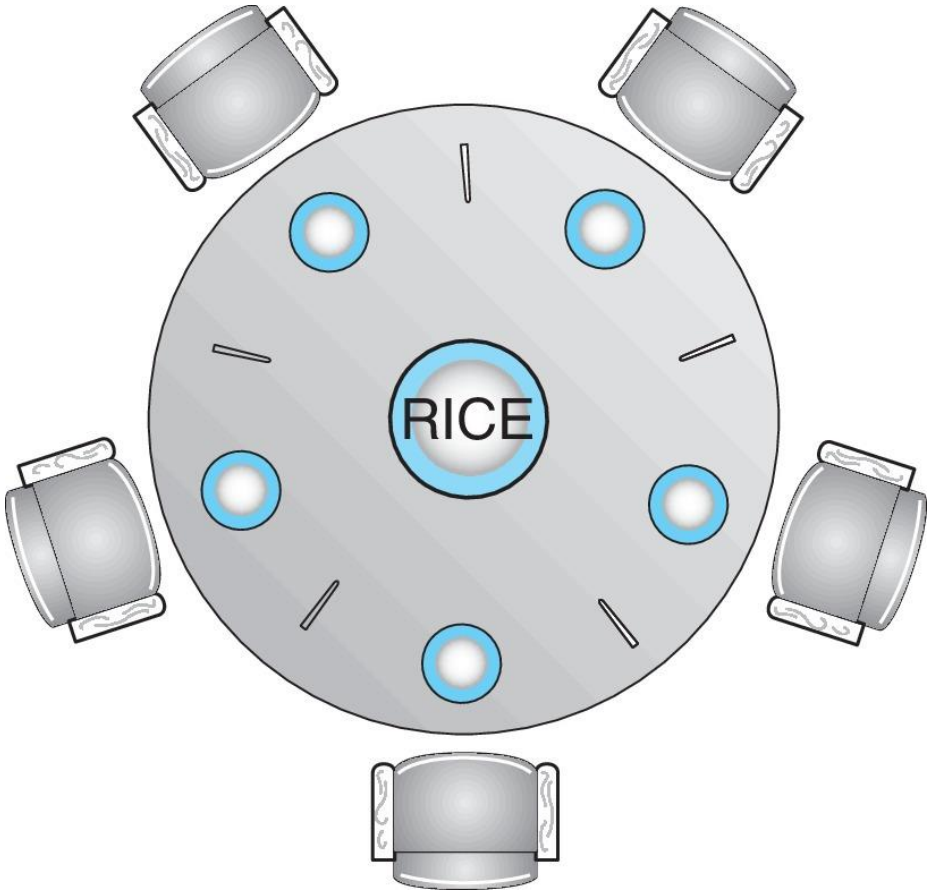
RELATÓRIO DA ATIVIDADE DOS FILÓSOFOS
THE DINING PHILOSOPHERS PROBLEM

06 de novembro de 2022

1. Introdução

Esse projeto busca implementar uma solução na linguagem C, usando o sistema POSIX, para o problema dos filósofos (“The dining philosophers problem”) discutido no livro “Operating System Concepts”. Neste problema, tem-se que N filósofos estão jantando ao redor de uma mesa redonda. À esquerda e à direita de cada filosofo há um palito “hashi” e no centro da mesa há um pote de arroz. Os filósofos precisam de dois palitos para comer do pote de arroz, porém eles devem dividir tais palitos com os seus colegas adjacentes.

Dessa forma, podemos modelar cada filosofo como sendo uma thread e cada palito como um recurso a ser compartilhado pelas threads. Em uma execução ideal, cada filosofo passa uma determinada quantidade de tempo pensando, depois passa determinada quantidade de tempo comendo. É importante impedir que dois filósofos peguem o mesmo palito ao mesmo tempo e impedir que um ou mais filósofos fiquem sem comer (starvation).



2. Orientações para a execução do código.

A implementação está disponível em: https://github.com/bTheMage/HungryPhilosophers

2.1. Em uma máquina com um sistema GNU-Linux da distribuição Debian ou similar, ou com apt instalado, garanta que os pacotes estão atualizados com:

```
sudo apt update && apt upgrade -y
```

2.2. Em seguida, garanta o compilador gcc esteja atualizado com:

```
sudo apt install build-essential -y
```

2.3. Em seguida, garanta que a biblioteca POSIX Thread (pthread.h) esteja instalada com:

```
sudo apt install libpthread-stubs0-dev
```

2.4. Caso não tenha git instalado, a ferramenta apt pode ser usada novamente:

```
sudo apt install git -y
```

2.5. Em qualquer pasta que desejar, clone o repositório com:

```
git clone https://github.com/bTheMage/HungryPhilosophers
```

2.6. Para compilar o programa, use:

```
make all
```

Ou

```
gcc -pthread main.c -o main
```

2.7. Para executar o programa, use:

```
make run
```

Ou

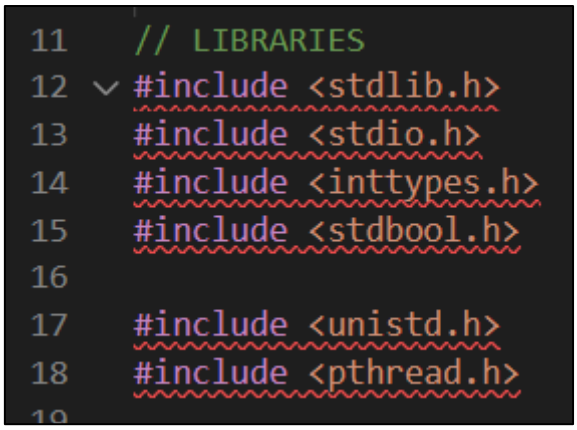
```
./main
```

2.8. Observe o programa executando e imprimindo na tela o estado de cada filosofo pelo tempo que desejar e, quando estiver satisfeito, pressione ctrl + c para encerrar a execução.

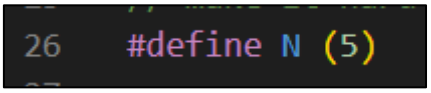
3. O código e explicações

Devido à ausência de objetos e métodos na linguagem c, e como o algoritmo não muito grande, variáveis e estados relacionadas ao monitor foram implementadas como variáveis globais e métodos foram implementados como funções.

As bibliotecas utilizadas estão listadas logo abaixo. Destaca-se que as ferramentas providas por “inttypes.h” e “stdbool.h” não são necessárias, mas facilitam a legibilidade do código.



A constante N representa a quantidade de filósofos presentes. Ela pode ser modificada antes da compilação, porém recomenda-se mantê-la em um valor não muito elevado, para facilitar a visualização do funcionamento do código e sua execução.



A macro “PANIC” serve para interromper o programa em caso de erro grave e “RIGHT_PHI” e “LEFT_PHI” calculam os ids dos filósofos adjacentes a algum id de acordo com o valor de N.

```
30  // MACROS
31  // Use this to stop execution.
32  #define PANIC(MSG) {fprintf (stderr, MSG); exit(EXIT_FAILURE);}
33
34  // OBS: "phi" will be used as an abbreviation for "philosopher" from this point on
35  // The following macros will return the index (ids) for the philosophers on the
36  // left and right of a given philosopher passed to the macro as an id (index).
37  #define LEFT_PHI(PHI_ID) ((PHI_ID + 1) % N)
38  #define RIGHT_PHI(PHI_ID) ((PHI_ID + (N - 1)) % N)
```

Os estados em que um filosofo pode se encontrar são definidos em um enum e as seguintes variáveis globais são declaradas.

```
42  // TYPES
43  // Let THINKING, HUNGRY and EATING be the 3 possible states for a philosopher
44  typedef enum {THINKING = 0, HUNGRY = 1, EATING = 2} PHI_STATE;
45
46
47
48  // GLOBAL VARIABLES
49  // A vector of the philosophers states.
50  PHI_STATE phi_states [N];
51
52  // A vector os mutexes and conditional variables to emulate monitors.
53  pthread_mutex_t phi_mutex[N];
54  pthread_cond_t phi_cond [N];
55
56  // A vector of thread ids whick will emulate the philosophers trying to access
57  // shared resources (chopsticks).
58  pthread_t phi_threads[N];
59
```

Na função “main”, há uma fase de inicialização de variáveis, seguida pela criação das threads. Note que o id dos filósofos é convertido e passado como ponteiro.

```c
// MAIN FUNCTION
int main (int argc, char *argv[]) {
    // LOCAL VARIABLES AND CONSTANTS
    // This variable serves to convert and display the state of a philosopher.
    const char phi_states_names[3][10] =
        #if (N <= 7)
            {"THINKING ", "HUNGRY   ", "EATING   "}
        #else
            {"T ", "H ", "E "}
        #endif
    ;

    // This variable is just a clock that counts how many times we display
    // something on the terminal.
    int clk = 0;
    printf("%d) \t", clk++);


    // CREATING THREADS
    for (int i = 0; i < N; i++) {
        // Creating the thread
        if ( pthread_create (&phi_threads[i], NULL, runner, (void*)((intptr_t)i)) )
            PANIC("Failed to create a thread!");

        // Just displaying the philosopher id.
        #if (N <= 7)
            printf("%d       \t", i);
        #else
            printf("%d \t", i);
        #endif
    }
    printf("\n");
```

Por fim, entra-se em um loop infinito em que se imprime na tela os estados dos filósofos com determinada frequência. As linhas de código subsequentes não são executadas, porém foram adicionadas para exemplificar uma sequência de finalização de um programa como este.

A função “runner” é executada por cada thread e tudo o que ela faz é chamar a função “pickup” e “putdown” para pegar e soltar os palitos, além de dormir por um determinado tempo, isso em loop infinito.

A função “pickup” muda o estado do filosofo para faminto (“HUGRY”) e checa se os palitos adjacentes estão disponíveis (o que é equivalente a checar se os filósofos adjacentes estão comendo, no estado “EATING”). Se estiverem indisponíveis, o filosofo se coloca numa lista de espera e fica aguardando.

```c
// A philosopher's though: Lets pickup a chop stick!
void pickup (int phi_id) {
    // I AM HUNGRY!!!
    phi_states[phi_id] = HUNGRY;

    // CAN I EAT NOW!?
    test(phi_id);

    // WAITING TO EAT
    if (phi_states[phi_id] != EATING) {
        // Obs: Since C does not provide a monitor, we use a condition variable
        // with a POSIX mutex lock.
        pthread_mutex_lock(&phi_mutex[phi_id]);
        while (phi_states[phi_id] != EATING)
            pthread_cond_wait(&phi_cond[phi_id], &phi_mutex[phi_id]);
        pthread_mutex_unlock(&phi_mutex[phi_id]);
    }
}
```

A função putdown muda o estado do filosofo para pensado (“THINKING”) e sinaliza que os palitos estão disponíveis (acordado as threads adjecentes).

```c
189   // Lets putdown the chopsticks
190   void putdown(int phi_id)
191   {
192       // I'm full, so back to thinking!
193       phi_states[phi_id] = THINKING;
194
195       // Lets wake our neighbor
196       test(RIGHT_PHI(phi_id));
197       test(LEFT_PHI(phi_id));
198   }
199
```

A função test, usada nas funções anteriores, serve para checar a disponibilidade dos palitos e muda o estado para “EATING”. Ela também sinaliza à thread a disponibilidade dos palitos.

```
168  // Testing if i can eat
169  void test(int phi_id)
170  {
171      bool right_phi_isnt_eating = (phi_states[RIGHT_PHI(phi_id)] != EATING);
172      bool left_phi_isnt_eating = (phi_states[LEFT_PHI(phi_id)] != EATING);
173      bool we_are_hungry = (phi_states[phi_id] == HUNGRY);
174
175      bool we_should_eat_now = left_phi_isnt_eating && right_phi_isnt_eating &&
176          we_are_hungry;
177
178      if (we_should_eat_now) {
179          // indicate that I'm eating
180          phi_states[phi_id] = EATING;
181
182          // signal() has no effect during Pickup(),
183          // but is important to wake up waiting
184          // hungry philosophers during Putdown()
185          pthread_cond_signal(&phi_cond[phi_id]);
186      }
187  }
```

O código não foi mostrado em sua plenitude nesse tópico, então recomenda-se que se cheque o repositório git. Ademais, pode haver pequenas mudanças como em comentários ou no número das linhas no repositório.

4. Resultados

A execução do programa mostra que os filósofos conseguem revezar o acesso ao palitos com sucesso e sem starvation. Indícios de starvation foram observados quando a frequência de impressão dos estados estava em sincronia com a frequência de espera (sleep) das thread. Porém, ao colocar tal frequência fora de fase, tais indícios cessaram. Portanto, em execuções futuras do programa, recomenda-se que a impressão dos resultados seja testada em múltiplas fases e frequências.

A imagem abaixo é advinda de uma impressão com um período de 0.090909 segundos.