Escola de Artes, **Ciências e Humanidades** (EACH)

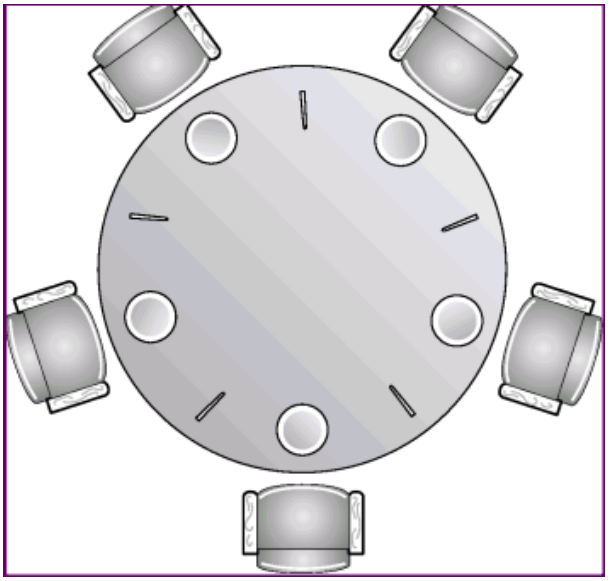
Prática 04 - ACH2044 Sistemas Operacionais
Relatório do experimento

Nome: Guilherme Cavalanti Gomes
NºUSP: 11844788

O PROBLEMA

O objetivo desse EP, é implementar uma solução para o problema do jantar dos filósofos, serão utilizadas POSIX Thread a partir da linguagem C.

O problema consiste em 5 filósofos em uma mesa com 5 garfos, eles pensam e comem, para comer precisam dos 2 garfos adjacentes, e então ao acabar os garfos são liberados e passam a pensar:



IMPLEMENTANDO O ALGORITMO:

A lógica usada nessa solução é utilizar os garfos como semáforos, dessa forma, os processos (filósofos) só poderão executar suas tarefas (comer) quando os garfos os permitirem.

Para isso serão utilizadas 5 variáveis do tipo semáforo num array (garfos), bem como 5 filósofos num array. Cada garfo começa inicialmente como 1.

A função wait faz com que o processo espere até o garfo ser igual a 1, então ela altera o seu valor para 0, já a função signal altera o valor do garfo chamado para 1.

A lógica é simples, cada processo espera pelo seu garfo e pelo garfo a direita, e então come, após dá um sinal para os garfos e passa a pensar, assim eles se alternam constantemente.

O CÓDIGO EM C:

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

//Função de sleep
void sleep(unsigned int mseconds)
{
    clock_t goal = mseconds + clock();
    while (goal > clock())
        ;
}
//semáforos
typedef int semaphore;

semaphore garfos[5];

//filósofos
typedef int filosofo;

filosofo filosofos[5];
// Função de comer
void comer (int threadid){
    int id = (int) threadid;
    printf("Filosofo %i comendo\n",id);
}

// Função de pensar
void pensar (int threadid) {
    int id = (int) threadid;
    printf("Filosofo %d voltou a pensar\n",id);
}

//Wait e Signal
void wait (semaphore* S, int id){
    while(*S <= 0);
    *S = 0;
    return;
}
void signal (semaphore* S){
*S = 1;
}
```

```c
void *processo(void *threadid){
    int id = (int) threadid;
    while (1) {
        wait(&garfos[id], id);
        wait(&garfos[(id+1)%5], id);
        comer(id);
        signal(&garfos[id]);
        signal(&garfos[(id+1)%5]);
        pensar(id);
        sleep(100);
    }
}
void main(){

//Valores inicias para os garfos são 1
for(int i = 0; i<5; i++)garfos[i]=1;

int num; //indetificador da thread
    pthread_t t1, t2, t3, t4, t5;
    int numThread; // armazena o retorno da pthread_create
    //Criando as threads
    num = 0;
    numThread = pthread_create(&t1, NULL, processo, (void *)num);
    num = 1;
    numThread = pthread_create(&t2, NULL, processo, (void *)num);
    num = 2;
    numThread = pthread_create(&t3, NULL, processo, (void *)num);
    num = 3;
    numThread = pthread_create(&t4, NULL, processo, (void *)num);
    num = 4;
    numThread = pthread_create(&t5, NULL, processo, (void *)num);
     pthread_join(t1,NULL); //Espera a t1 acabar
    pthread_join(t2,NULL); // Espera a t2 acabar
     pthread_join(t3,NULL); //Espera a t3 acabar
    pthread_join(t4,NULL); // Espera a t4 acabar
     pthread_join(t5,NULL); //Espera a t5 acabar
}
```

Resultados:

```
C:\Users\W10\Desktop\EP4-OAC>gcc -o programa4 programa4.c -lpthread

C:\Users\W10\Desktop\EP4-OAC>programa4.exe
Filosofo 0 comendo
Filosofo 2 comendo
Filosofo 2 voltou a pensar
Filosofo 0 voltou a pensar
Filosofo 4 comendo
Filosofo 4 voltou a pensar
Filosofo 3 comendo
Filosofo 3 voltou a pensar
Filosofo 1 comendo
Filosofo 1 voltou a pensar
Filosofo 4 comendo
Filosofo 4 voltou a pensar
Filosofo 0 comendo
Filosofo 0 voltou a pensar
Filosofo 3 comendo
Filosofo 3 voltou a pensar
Filosofo 2 comendo
Filosofo 2 voltou a pensar
Filosofo 1 comendo
Filosofo 1 voltou a pensar
Filosofo 4 comendo
Filosofo 4 voltou a pensar
Filosofo 0 comendo
Filosofo 0 voltou a pensar
Filosofo 3 comendo
Filosofo 3 voltou a pensar
Filosofo 2 comendo
Filosofo 2 voltou a pensar
^C
```

Análise:

Pelos resultados é possível concluir que essa solução é funcional, cada filósofo só come quando possui os 2 garfos disponíveis, e ao acabar libera os garfos e passa a pensar. Assim nenhum filósofo irá esperar para sempre, e nunca o mesmo garfo será usado por 2 filósofos diferentes. Também são evitados deadlocks pois os processos estão sempre esperando os 2 garfos estarem livres para começarem a comer, assim nunca haverá uma situação em que cada filósofo segura 1 garfo.