Val C4

1. Análise do Problema

No problema clásssico dos filósofos, cada filósofo precisa de dois garfos (um à sua direita e outro à sua esquerda) para comer.

Este problema já foi resolvido no concurso 4, contudo temos agora a seguinte restrição: **supondo agora que os filosofos são todos destros, eles precisão 1o ter acesso ao garfo da direita para depois ter o garfo da esquerda e assim conseguir comer.**

Agora temos que repensar na forma como vamos alterar o codigo base de forma a não haver *starvation* e *deadlock,* uma vez que:

- ***Starvation*** **:** Como no exercicio dos filosofos, sendo que existem somente 2 filosofos que comem e o resto esperam infinitamente para comer **(considerando que são 5 filosofos)**.
- ***Deadlock*** **:** Se todos os filósofos pegarem primeiro o garfo à direita e esperarem pelo garfo à esquerda, todos esperam infinitamente.

A priori basta pegar no codigo base e impor uma ordem consistente de aquisição dos garfos. Sendo preciso 2 semaforos: 1 para a mão direita e outro para a mão esquerda.

Assim mudamos o codigo base e vamos obrigar cada filósofo a pegar primeiro o garfo à direita e só depois o garfo à esquerda. Evitando o *deadlock* e *starvation*.

2. Implementação

Segue abaixo o código C modificado, onde introduzimos semáforos para os garfos e mudamos a ordem de aquisição, sempre pegamos primeiro o garfo da direita:

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

#define N 5

sem_t forks[N];

void *philosopher(void *num) {
    int i = *(int *)num;
    while (1) {
        printf("Filósofo nº %d pensando\n", i+1);
        sleep(1);
        printf("Filósofo nº %d tem fome\n", i+1);

        // Pega primeiro o garfo da direita
        sem_wait(&forks[i]);
        // Depois pega o garfo da esquerda
        sem_wait(&forks[(i+1)%N]);

        printf("Filósofo nº %d comendo\n", i+1);
        sleep(1);

        // Devolve primeiro o garfo da esquerda
        sem_post(&forks[(i+1)%N]);
        // Depois devolve o garfo da direita
        sem_post(&forks[i]);

        printf("Filósofo nº %d parou de comer\n", i+1);
    }
}
```

```c
int main(void) {
    pthread_t thread_id[N];
    int i;
    int phil[N] = {0,1,2,3,4};

    for (i = 0; i < N; i++) {
        sem_init(&forks[i], 0, 1);
    }

    for (i = 0; i < N; i++) {
        pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);
    }

    // Nunca chega aqui lol
    for (i = 0; i < N; i++) {
        pthread_join(thread_id[i], NULL);
    }
    return 0;
}
```

E segue-se capturas de ecrã da execução do programa de forma a validar como está a funcionar devidamente, logicamente tive que parar a execução do programa uma vez que contem um loop infinito: