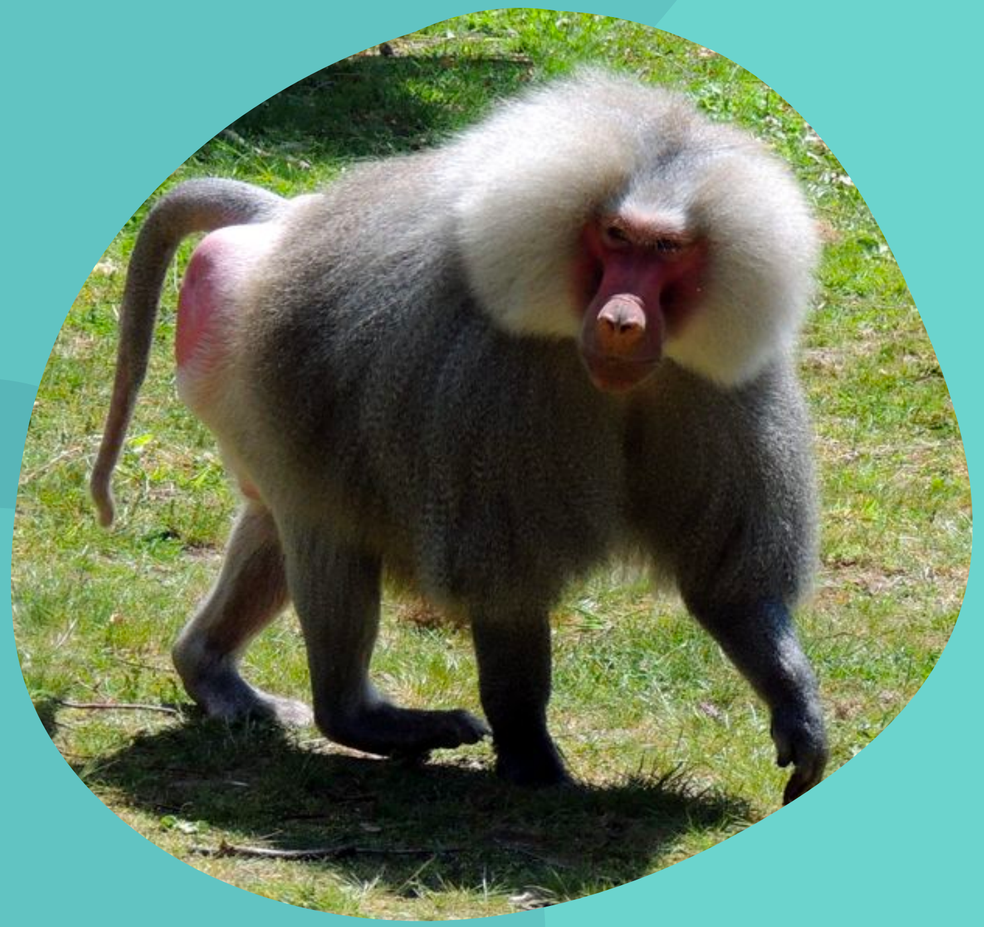
THE BABOON CROSSING PROBLEM

INE5645 - PROGRAMAÇÃO PARALELA & DISTRIBUÍDA - LAB2

VINICIUS ARALDI LUNARDI FARIAS

PROBLEMA

- Adaptado de "Operating Systems: Design and Implementation" por Tanenbaum
- Babúinos estão tentando atravessar um cânion profundo na África do Sul através de uma única corda que liga as duas extremidades do cânion
- Mais de um babuíno pode atravessar o cânion indo na mesma direção, mas se dois babúinos indo em direções opostas se encontrarem no meio da travessia, eles vão brigar e cair no abismo (deadlock)
- A corda só aguenta 5 babuínos atravessando ao mesmo tempo
- Deve ser permitido que todos os babuínos atravessem o cânion, estando de ambos os lados dele (starvation)



PROPOSTA

Elaborar uma solução de sincronização em C/PYTHON (decidir) que solucione o problema e

- evite deadlock (babúinos indo em direção oposta se encontrarem no meio da travessia)
- permita que todos os babúinos atravessem o cânion independente do lado que eles estão (no starvation)
- não ultrapasse o limite de 5 babuínos atravessando ao mesmo tempo na corda, indo na mesma direção

SOLUÇÃO

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

sem_t rope_free;
sem_t direction_free;

sem_t baboon_crossing;
sem_t baboon_crossed;
sem_t rope;
sem_t rope_ready;

int max_baboons_on_rope = 5;
int current_direction;
int left_baboons, right_baboons;
int baboons_crossing = 0;
int baboons_to_cross;
int crossed_right = 0;
int crossed_left = 0;

pthread_mutex_t mutex;

typedef struct baboon_t
{
    int id;
    int direction;
} baboon_b;

void cross(int id, int direction)
{
    sleep(rand() % 5);
    printf("Babuíno %d atravessou a corda na direção %d!\n", id, direction);
}
```

SOLUÇÃO

```c
void *thread_rope(void *arg)
{
    printf("Um babuíno entrou na corda\n");
    while (1)
    {
        sem_wait(&baboon_crossing);
        sem_post(&rope);
        sleep(rand() % 4);
        sem_wait(&baboon_crossed);
        sem_post(&rope_ready);
    }
    pthread_exit(0);
}
```

SOLUÇÃO

```
void *thread_cross_rope(void *arg)
{
    baboon_b *b;
    b = (baboon_b *)arg;

    int direction = b->direction;
    int id = b->id;

    pthread_mutex_lock(&mutex);

    if (baboons_crossing > 0)
    {
        if (current_direction == direction)
        {
            if (baboons_crossing < max_baboons_on_rope)
            {
                printf("Corda não está vazia mas ainda não está cheia. Babuíno %d atravessando na direção %d\n", id, direction);
            }
            else
            {
                printf("Corda está cheia. Vamos aguardar liberar\n");
                sem_wait(&rope_free);
                pthread_mutex_unlock(&mutex);
            }
        }

        if (current_direction != direction)
        {
            printf("Babuíno %d tentou atravessar na direção oposta. Vamos aguardar\n", id);
            sem_wait(&direction_free);
            pthread_mutex_unlock(&mutex);
        }
    }
```

SOLUÇÃO

```
else
{
    printf("Corda estava vazia. Babuíno %d atravessando na direção %d\n", id, direction);
    current_direction = direction;
}

baboons_crossing++;
pthread_mutex_unlock(&mutex);
sem_post(&baboon_crossing);
sem_wait(&rope);
cross(id, direction);
sem_post(&baboon_crossed);
sem_wait(&rope_ready);
pthread_mutex_lock(&mutex);
baboons_crossing--;
baboons_to_cross--;

if (baboons_crossing < max_baboons_on_rope)
{
    sem_post(&rope_free);
}
if (baboons_crossing == 0)
{
    sem_post(&direction_free);
}

if (direction == 1)
{
    crossed_left++;
}
else
{
    crossed_right++;
}
pthread_mutex_unlock(&mutex);
printf("Babuíno %d finalizou travessia na direção %d \n", id, direction);

pthread_exit(0);
```

SOLUÇÃO

```c
int main(int argc, char **argv)
{
    if (argc != 2)
    {
        printf("precisa informar o número de threads (babuínos = n-1)\n%s num_threads\n", argv[0]);
        return 1;
    }

    printf("VALORES DAS DIREÇÕES: \n* DIREITA: 0 ESQUERDA: 1 *\n");

    srand(time(NULL));

    int n_threads = atoi(argv[1]);

    baboons_to_cross = n_threads - 1;

    printf("%d babuínos precisam atravessar \n", baboons_to_cross);

    int i = 0;
    pthread_t threads[n_threads];

    sem_init(&baboon_crossed, 0, 0);
    sem_init(&baboon_crossing, 0, 0);
    sem_init(&rope, 0, 0);
    sem_init(&rope_ready, 0, 0);
    sem_init(&rope_free, 0, 0);
    sem_init(&direction_free, 0, 0);

    pthread_mutex_init(&mutex, NULL);
    pthread_create(&threads[i], NULL, thread_rope, NULL);

    for (i = 1; i < n_threads; i++)
    {

        int dir = rand() % 2;
        if (dir == 1)
        {
            left_baboons++;
        }
```

SOLUÇÃO

```
for (i = 1; i < n_threads; i++)
{

    int dir = rand() % 2;
    if (dir == 1)
    {
        left_baboons++;
    }
    else
    {
        right_baboons++;
    }

    baboon_b *b;
    b = malloc(sizeof(baboon_b));
    b->id = i;
    b->direction = dir;

    printf("direção do babuíno %d setada para %d \n", b->id, b->direction);
    pthread_create(&threads[i], NULL, thread_cross_rope, (void *)b);
}

printf("%d babuínos na esquerda\n", left_baboons);
printf("%d babuínos na direita\n", right_baboons);

while (baboons_to_cross > 0)
{

    for (i = 1; i < n_threads; i++)
    {
        pthread_join(threads[i], NULL);
    }
    i = 1;
}

printf("Nao ha mais babuínos para atravessar\n");
printf("%d babuínos atravessaram para esquerda\n", crossed_left);
printf("%d babuínos atravessaram para direita\n", crossed_right);
```

SOLUÇÃO

```
pthread_cancel(threads[0]);

sem_destroy(&baboon_crossed);
sem_destroy(&baboon_crossing);
sem_destroy(&rope);
sem_destroy(&rope_ready);
sem_destroy(&rope_free);
sem_destroy(&direction_free);

pthread_mutex_destroy(&mutex);
```