Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programação Concorrente

Relatório de Entrega de Atividades

Aluno(s): Amanda Oliveira Alves e Fillype Alves do Nascimento

Matrícula: 15/0116276 e 16/0070431

Atividade: Aula Prática 04 - Dormir e Acordar

1.1.1

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.1.1.c
// atividade: 1.1.1

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

pthread_mutex_t possedalista;
int assinaturas=0, *ass;

void *aluno(void *id) {
    int *i = (int *) id;

    pthread_mutex_lock(&possedalista);

    *(ass+assinaturas) = *i;
    assinaturas++;

    sleep(1);

    printf("Aluno %d assinou a lista.\n", *i);

    pthread_mutex_unlock(&possedalista);
}
```

```
int main() {
    int n;
    scanf("%d",&n);

    int tmp[n];
    ass = tmp;

    pthread_t t[n];

    pthread_mutex_init(&possedalista, 0);
    for (int i = 0; i < n; i++) {
        int *id;
        id = malloc(sizeof(int));
        *id = i;
        pthread_create(&t[i], NULL, aluno, (int *) id);

    }

    for (int i = 0; i < n; i++) {
        pthread_join(t[i], NULL);
    }

    printf("A ordem de assinatura da lista foi: ");
    for(int i=0; i<n ;i++){
        printf("%d ", *(ass+i));
    }
    printf("\n");

    pthread_mutex_destroy(&possedalista);
    return 0;
}
```

2.1.1

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 2.1.1.c
// ativnade: 2.1.1

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define NUMTHREADS 90
#define SALA1 10
#define SALA2 6
#define SALA3 18


sem_t semaforo;
void *turista(void *args){
  int n = *((int *)args);
  sem_wait(&semaforo);
  sleep(2);
  printf("Sala %d completa, começando a explicação.\n", n);
  sem_post(&semaforo);
}

int main(){
  pthread_t t[NUMTHREADS];
  int *n;
  sem_init(&semaforo, 0, SALA1);
  for (int i = 0; i < NUMTHREADS; i++) {
    n = (int *) malloc(sizeof(int));
    *n = 1;
    pthread_create(&t[i], NULL, turista, (void *)n);
  }
  for (int i = 0; i < NUMTHREADS; i++) {
    pthread_join(t[i], NULL);
  }
  sem_destroy(&semaforo);

  sem_init(&semaforo, 0, SALA2);
  for (int i = 0; i < NUMTHREADS; i++) {
    n = (int *) malloc(sizeof(int));
    *n = 2;
```

```
    pthread_create(&t[i], NULL, turista, (void *)n);
  }
  for (int i = 0; i < NUMTHREADS; i++) {
    pthread_join(t[i], NULL);
  }
  sem_destroy(&semaforo);

  sem_init(&semaforo, 0, SALA3);
  for (int i = 0; i < NUMTHREADS; i++) {
    n = (int *) malloc(sizeof(int));
    *n = 3;
    pthread_create(&t[i], NULL, turista, (void *)n);
  }
  for (int i = 0; i < NUMTHREADS; i++) {
    pthread_join(t[i], NULL);
  }
  sem_destroy(&semaforo);
  return 0;
}
```

3.1.1

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 3.1.1.c
// atividade: 3.1.1

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

int contador = 0;
pthread_cond_t condespera;
pthread_mutex_t mutex;
void *carteiro(void *empty){
    pthread_mutex_lock(&mutex);
    printf("Sou o carteiro, daqui a 1 segundo eu volto\n");
    sleep(1);
    pthread_cond_signal(&condespera);
    pthread_mutex_unlock(&mutex);
}

void *cliente(void *args){
    int id = *((int *)args);
```

```
    pthread_mutex_lock(&mutex);
    printf("Sou o cliente %d\n", id);
    contador = contador + 1;
    if ((contador % 2) == 0){
         pthread_cond_wait(&condespera, &mutex);
    }
    pthread_mutex_unlock(&mutex);
}

int main(){
    pthread_t l, e[100];
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&condespera, NULL);
    int *id;
    pthread_t t[100];

    for (int i = 0; i < 30; i++){
        for(int j = 0; i<100; i++){
            *id = j;
            pthread_create(&e[i], NULL, cliente, (void *)id);
        }
    }

    pthread_create(&l, NULL, carteiro, NULL);
    sleep(2);
    for (int i = 0; i < 30; i++) {
        for(int j = 0; i<100; i++){
            pthread_join(e[i],NULL);
        }

    }

    pthread_join(l, NULL);
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&condespera);
    return 0;
}
```