# Sistemas Operacionais I
# 1º Semestre de 2019
# 1ª Avaliação

Francisco José da Silva e Silva
Departamento de Informática, UFMA

**Responda a apenas 4 das questões abaixo**

**Questão 01** Explique os passos para a execução de uma chamada ao sistema operacional desde a sua solicitação pelo programa do usuário até o recebimento do resultado de sua execução.

**Questão 02** Explique como um sistema operacional de tempo repartido realiza trocas dos processos em execução sem que os mesmos possam monopolizar o uso do processador.

- interrupção, timer, fluxo de exec.

**Questão 03** Por quais estados um processo pode passar desde a sua submissão até o seu término? Quais eventos levam o processo a troca entre estes estados?

- ready, pausa, running

**Questão 04** O código abaixo resolve o problema da exclusão mútua? Se não resolve, ilustre uma situação em que ele falhe. Quais as desvantagens/limitações deste código?

```
while (TRUE) {
    while (turn != 0)       /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)       /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

**Questão 05** Na solução do problema dos produtores e consumidores ilustrado a seguir a inversão da ordem das operações `down()` no consumidor gera qual problema? Ilustre com um exemplo a ocorrência do problema em questão.

```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N;                            /* counts empty buffer slots */
semaphore full = 0;                             /* counts full buffer slots */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE is the constant 1 */
          item = produce_item();                /* generate something to put in buffer */
          down(&empty);                         /* decrement empty count */
          down(&mutex);                         /* enter critical region */
          insert_item(item);                    /* put new item in buffer */
          up(&mutex);                           /* leave critical region */
          up(&full);                            /* increment count of full slots */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* infinite loop */
          down(&full);                          /* decrement full count */
          down(&mutex);                         /* enter critical region */
          item = remove_item();                 /* take item from buffer */
          up(&mutex);                           /* leave critical region */
          up(&empty);                           /* increment count of empty slots */
          consume_item(item);                   /* do something with the item */
     }
}
```

**Questão 06** Explique o que são mutexes e como são utilizados para resolver o problema de exclusão mútua.