
	<p align="center"> <b>Universidade Estadual de Maringá</b>  <b>Departamento de Informática</b>  <b>Centro de Tecnologia</b>  <b>Bacharelado em Informática</b> </p>	
<b>Disciplina:</b> 5205 – Programação Concorrente		
<b>Professor:</b> Rodolfo Miranda Pereira		

## Trabalhos Práticos da Disciplina: Paralelização de Problemas de IPC

### 1. Descrição dos problemas

#### a) Problema da Montanha Russa (Complexidade Média)

Existem  $n$  passageiros, que repetidamente aguardam para entrar em um carrinho da montanha russa, fazem o passeio, e voltam a aguardar. Vários passageiros podem entrar no carrinho ao mesmo tempo, pois este tem várias portas. A montanha russa tem somente um carrinho, onde cabem  $C$  passageiros ( $C < n$ ). O carrinho só começa seu percurso se estiver lotado. Sincroniza-se as ações dos processos Passageiro e Carrinho usando semáforos.

```

semaphore passageiro = C;
semaphore carrinho = 0;
semaphore andando = 0;
semaphore mutex = 1;
int Npass = 0;

void passageiro() {
    while (TRUE) {
        down(&passageiro);
        entra_no_carrinho(); /* vários passageiros podem entrar */
                                /* "ao mesmo tempo" */

        down(&mutex);
        Npass++;
        if (Npass == C) {      /* carrinho lotou */
            up(&carrinho);    /* autoriza carrinho a andar */
            down(&andando);  /* espera carrinho parar */
            up(&mutex);
        } else {
            up(&mutex);
            down(&andando); /* espera carrinho lotar, passear e voltar */
        }
    }
}

void carrinho() {
    while (TRUE) {
        down(&carrinho);      /* espera autorização para andar */
        passeia();           /* faz o passeio e volta */
        Npass = 0;           /* esvazia carrinho */
        for (int i = 0; i < C; i++){
            up(&andando);     /* libera passageiro que andou de
                                /* volta à fila */
            up(&passageiro);  /* libera entrada no carrinho */
        }
    }
}

```

## b) Problema do Supermercado (Complexidade Alta)

Considere um supermercado com N caixas de pagamento com um empregado em cada caixa. Enquanto houver clientes na sua fila o empregado atende-os. Se não tiver nenhum cliente para ser atendido na sua fila, o empregado pode atender um cliente de outra fila. Se não existir ninguém para atender em nenhuma das filas, o empregado bloqueia-se à espera de clientes. Quando o cliente chega ele escolhe a fila que tiver menos clientes. Uma vez escolhida o cliente não pode trocar de fila, exceto para ser atendido conforme descrito anteriormente. O número de clientes por fila é ilimitado.

```
semaphore filas [0..N1];
semaphore emp [0..N1];
semaphore mutex = 1;

int cfilas [0..N1];

void cliente(){
    down(&mutex);
    int mf = 0;
    int count = cfilas[0];
    for (int i = 1; i < N; i++){
        if(count==0)
            break;
        if(cfilas[i]<cfilas[mf]){
            mf = i;
            count = cfilas[i];
        }
    }
    up(&emp[mf]);
    cfilas[mf]++;
    up(&mutex);
    down(&filas[mf]);
    serAtendido();
}

void empregado(int fila){
    while(TRUE){
        while(TRUE){
            down(&mutex);
            if(cfilas[fila]>0){
                cfilas[fila];
                up(&filas[fila]);
                up(&mutex);
                down(&emp[fila]);
                atender();
            } else {
                up(&mutex);
                break;
            }
        }
        int j, nf, nvazias = 0;
        for(j = 0; j < N1; j++){
            nf = next(fila,j);
            down(&mutex);
            if(cfila[nf] > 0){
                nvazias = 0;
                cfila[nf];
                up(&mutex);
                up(&fila[nf]);
                down(&emp[nf]);
                atende();
            } else {
                up(&mutex);
                nvazias++;
            }
        }
        down(&mutex);
        if (nvazias == N
            && cfilas(fila) == 0){
            up(&mutex);
            down(&emp[fila]);
        } else {
            up(&mutex);
        }
    }
}
```

### c) Problema do Pombo (Complexidade Baixa)

Considere a seguinte situação. Um pombo correio leva mensagens entre os sites A e B, mas só quando o número de mensagens acumuladas chega a 20. Inicialmente, o pombo fica em A, esperando que existam 20 mensagens para carregar, e dormindo enquanto não houver. Quando as mensagens chegam a 20, o pombo deve levar exatamente (nenhuma a mais nem a menos) 20 mensagens de A para B, e em seguida voltar para A. Caso existam outras 20 mensagens, ele parte imediatamente; caso contrário, ele dorme de novo até que existam as 20 mensagens. As mensagens são escritas em um post-it pelos usuários; cada usuário, quando tem uma mensagem pronta, cola sua mensagem na mochila do pombo. Caso o pombo tenha partido, ele deve esperar o seu retorno p/ colar a mensagem na mochila. O vigésimo usuário deve acordar o pombo caso ele esteja dormindo. Cada usuário tem seu bloquinho inesgotável de post-it e continuamente prepara uma mensagem e a leva ao pombo.

```
#define N 20

int contaPostIt = 0;
semaforo mutex = 1; //controlar acesso a variavel contaPostIt
semaforo cheia = 0; //usado p/ fazer o pombo dormir enquanto n ha 20msg
semaforo enchendo = N; //usado p/ fazer usuarios dormirem enquanto pombo trabalha

void usuario() {
    while(TRUE) {
        down(&enchendo);
        down(&mutex);
        colaPostIt_na_mochila();
        contaPostIt++;
        if (contaPostIt == N)
            up(&cheia);
        up(&mutex);
    }
}

void pombo() {
    while(TRUE) {
        down(&cheia);
        down(&mutex);
        leva_mochila_ate_B_e_volta();
        contaPostIt = 0;
        for (int i = 0; i < N; i++)
            up(&enchendo);
        up(&mutex);
    }
}
```

## 2. O que deve ser feito

Os problemas supracitados foram pensados para mostrar na prática questões de IPC (*Inter-Process Communication*) que envolvem disputa por recursos mutuamente exclusivos.

Escolham um desses problemas.

Os alunos deverão desenvolver duas implementações para o mesmo problema, cada implementação referente a um trabalho:

- **Primeiro Trabalho:** Implementação utilizando memória compartilhada, ou seja, usando a biblioteca pthreads.
- **Segundo Trabalho:** Implementação utilizando memória distribuída, ou seja, usando a biblioteca MPI.

A solução deve fazer uso de programação concorrente.

## 3. O que deve ser entregue para cada trabalho

Deve-se entregar um relatório escrito, no formato de artigo da SBC com no máximo 10 páginas, contendo:

- Introdução;
- Descrição detalhada do problema;
- Como o problema foi modelado/implementado com memória compartilhada/distribuída;
- Análise dos resultados;
- Conclusões; e
- Referências Bibliográficas.

A análise de desempenho deve ser feita utilizando as métricas aprendidas em aula (Speedup Teórico, Lei de Amdahl, etc).

- Note que para algumas métricas você deverá saber o tempo de execução do programa sequencial, logo, vocês também deverão implementar uma versão sequencial para resolver o problema;

Além dos relatórios, deve-se entregar os códigos fonte dos trabalhos, com as devidas instruções para sua execução.

#### 4. Como os trabalhos serão avaliados

Cada trabalho será referente a uma avaliação distinta:

- O relatório e a implementação referentes à memória compartilhada formarão a nota do Primeiro Trabalho;
- O relatório e a implementação referentes à memória distribuída formarão a nota do Segundo Trabalho.

A nota de cada trabalho somará no máximo 10,0 e será avaliada de acordo os seguintes critérios:

- a) Implementação.
  - I. Funcionamento correto. (20%)
  - II. Código Limpo (20%).
  - III. Dificuldade do problema escolhido (20%).
- b) Relatório.
  - IV. Organização do texto e ortografia (20%).
  - V. Análise dos resultados (20%).

O critério III descrito como “dificuldade do problema escolhido” será avaliado de acordo com a complexidade informada na descrição dos problemas da seção 1:

- Complexidade Baixa: 0.0;
- Complexidade Média: 1.0;
- Complexidade Alta: 2.0;

#### 5. Considerações Gerais

- Datas de Entrega e Apresentação:
  - Primeiro Trabalho: 08/11/2016
  - Segundo Trabalho: 20/12/2016
- As implementações devem ser feitas na Linguagem C.
- O trabalho pode ser feito em grupos de até dois alunos.
- O trabalho deve ser entregue no moodle (<http://moodlep.uem.br/course/view.php?id=770>)
  - Chave de inscrição: PCINFO2016
  - Não imprima o trabalho, entregue apenas no moodle.
- Não esqueça de identificar a dupla que fez o trabalho ao envia-lo no moodle.
- No dia da entrega cinco duplas serão sorteadas para apresentar o trabalho.
  - A apresentação não vale nota, porém, alunos que não comparecerem no dia da apresentação ou, caso sorteados, não souberem explicar sobre seu trabalho, terão suas notas zeradas.
  - A apresentação consiste em mostrar e explicar o funcionamento do código, bem como a análise de desempenho realizada.
- Cópias de qualquer tipo anularão o trabalho.
  - Esta regra não se aplica aos pseudocódigos apresentados na primeira seção, ou seja, sintam-se livre para usa-los como base (mas não necessariamente se retenham a eles).