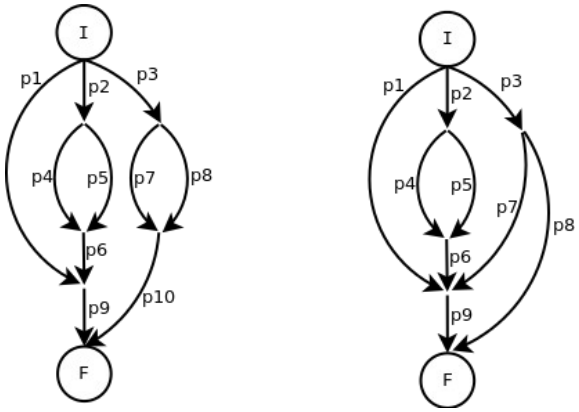
Universidade Federal de Viçosa
Departamento de Informática
INF310 – Programação Concorrente e Distribuída
14/09/2018 – Prova 1 – Valor: 25 pontos

Nome: _____ Matrícula: _____

1. Responda as questões a seguir:
   a) **(2 pontos)** Defina Região Crítica e relacione com Condição de Corrida (ou Corrida crítica) .
   b) **(2 pontos)** Explique o funcionamento da instrução Test and Set (TS).
   c) **(2 pontos)** Por que um arbitrador (ou gerenciador) de interrupções é importante?
   d) **(2 pontos)** Na sala de aula foi visto que processos podem ser classificados como Concorrentes e Paralelos. Qual a diferenças entres essas classificações?

2. Considere os grafos de precedência de processos a seguir.
   a) **(2 pontos)** Qual (ou quais) deles é (são) propriamente aninhado(s)? Justifique utilizando funções S e P para representá-lo(s).



   b) **(3 pontos)** Crie um grafo mostrando a representação das threads descritas pelo código V4 a seguir.

```
V4program
  process P;
  k2, k4: integer;
  { k2:= fork;
    if myself=k2 then
    { nl; write('p2');
      k4:=fork;
      if myself=k4 then {nl; write('p4'); quit};
      nl; write('p3');
      join(k4);
      quit
    };
    nl; write('p1');
    join(k2);
    nl; write('p5')
  }
endprogram
```

   c) **(3 pontos)** É possível representar grafos que não sejam propriamente aninhados através de programas V4? Justifique sua resposta utilizando exemplos.

3. Analise cada um dos três algoritmos de exclusão mútua apresentados a seguir e responda se são válidos. Para cada algoritmo inválido, mostre qual dos princípios (exclusão mútua, ausência de *deadlock* e ausência de atraso desnecessário) é violado e justifique.

a) (3 pontos)

```
...
quer[eu]:=true;
loop
   exit when ¬quer[outro];
   quer[eu]:=false;
   quer[eu]:=true;
endloop;
REGIÃO CRÍTICA;
quer[eu]:=false;
endloop;
...
```

b) (3 pontos)

```
...
quer[eu]:= true;
loop
   exit when ¬quer[outro]
   if vez=outro then
   {  quer[eu]:=false;
      loop
         exit when vez=eu
      endloop
      quer[eu]:=true
   }
endloop
REGIÃO CRÍTICA;
vez:=outro;
quer[eu]:=false;
...
```

c) (3 pontos)

```
...
quer[eu]:=true;
if vez=outro then
   if quer[outro] then
   {  quer[eu]:=false;
      loop
         exit when vez=eu
      endloop
      quer[eu]:=true;
   }
  else vez:=eu;
REGIÃO CRÍTICA ;
quer[eu]:=false;
vez:=outro
...
```

4. **(Questão bônus – OpenMP)** Veja o código serial para o método de ordenação Count Sort.

```
void countSort(int a[], int n) {
   int i, j, count;
   int *temp=malloc(n*sizeof(int));

   for (i=0; i<n; i++) {
      count=0;
      for(j=0; j<n; j++)
         if (a[j] < a[i])
            count++;
         else if (a[j] == a[i] && j<i)
            count++;
      temp[count] = a[i];
   }

   memcpy(a,temp, n*sizeof(int));
   free(temp);
}
```

A ideia básica é de, para cada elemento *a[i]* na lista *a*, contar o número de elementos menores que *a[i]*, armazenando em *count*. Em seguida, o elemento *a[i]* é inserido em uma lista temporária na posição *count*. Se um elemento é igual a *a[i]*, a posição dos dois na lista original é usada como desempate. Ao final, o conteúdo do array temporário (ordenado) é copiado para o array original.

a) **(2 pontos)** Se o loop externo [*for (i=0; i<n; i++)* ...] for paralelizado, quais variáveis devem ser privadas e quais devem ser compartilhadas?

b) **(2 pontos)** Escreva uma versão paralela deste algoritmo de modo que obtenha maior eficiência na ordenação.