1ª Prova de Sistemas Operacionais – 2011/1

Aluno: .....

1. (3,0) Considere o conjunto de 5 processos (P1,..., P5) com o seguinte comportamento:

Processo	Prioridade	Tempo de Chegada	Tempo de Processamento
P1	5	0.0	6
P2	4	1.0	4
P3	4	2.0	3
P4	3	2.5	3,5
P5	1	4.0	1,5

Represente de forma gráfica o escalonamento dos processos e calcule o tempo médio de espera para cada um dos seguintes algoritmos:
a) Menor Tempo Remanescente Primeiro (Shortest Remaining Time First - SRTF)
b) Escalonamento Circular (Round Robin - RR), com quantum = 1
c) Prioridades (Preemptive Priority) (1 = prioridade mais alta)

2. (1,5) Considere o seguinte programa (assuma que não existem erros na execução das chamadas):

```
int main() {
   int i=0;
      while (i<10) {
      i = i + 2;
      if (fork() != 0) {
            printf (“I’m %d\n”, i);
            i = i - 1;
            exit(0);
      }
      sleep(5);
      }
           return 0;
}
```

a) Explique o seu funcionamento, justificando qual é o resultado da execução do programa (i.e. mostre o que é impresso e o porquê). Assuma que o *quantum* de tempo de cada processo é muito inferior a 5 segundos.
b) Diga se existe a formação de processos *zombies*, e/ou criação de órfãos com adoção por parte do processo *init*.

3. (1,0) Dada a seguinte parte de um programa, quantas vezes serão executados os programas **exame** e **alunos**? Justifique.

```
...
pid = fork();
pid = fork();
for(i=0; i<5; i++)
{
     pid = fork();
     execlp(“exame”, “exame”, 0);
     if (pid == 0)
         break;
}
execlp(“alunos”, “alunos”,”so2”,0);
...
```

4. (1,0) Três processos necessitam sincronizar as suas ações de modo que a seguinte seqüência seja estabelecida:

`P1` $\rightarrow$ `data1` $\rightarrow$ `P2` $\rightarrow$ `data2` $\rightarrow$ `P3`

significando que `P1` produz `data1`, e somente quando `data1` está pronto `P2` pode prosseguir, etc. É garantido que cada processo será executado apenas uma vez (isto é, não existem ciclos). O esqueleto de cada processo é mostrado abaixo (nota: "`do some work`" e "`do more stuff`" são trabalhos independentes dos dados compartilhados).

```
thread1() {
// do some work
....
....
// produce data1
data1 := 1;

//do more stuff
....
}
```

```
thread2() {
//do some work
....
....
//do stuff with data1
//and produce data2
data2 := data2 * data1;
//do more stuff
....
}
```

```
thread3() {
//do some work
....
....
//do stuff with data2
count << data2 <<endl;

// do more stuff
....
}
```

Adicione semáforos e operações sobre os semáforos para garantir a ordem apropriada de execução. Indique o valor inicial de cada semáforo.

5. (2,0) Você foi convocado(a) pela Mãe Natureza para ajudá-la com a reação química da água, que ela não está conseguindo executar direito devido a problemas de sincronização. O problema consiste em reunir, ao mesmo tempo, dois átomos de hidrogênio (H) e um de oxigênio (O) para formar uma molécula de água. Nas tentativas de solução apresentadas por ela (vide abaixo) átomos são vistos como *threads* e cada átomo de H invoca um procedimento `hReady` quando está pronto para reagir, assim como cada átomo de O invoca um procedimento `oReady` quando está pronto para reagir. Para ajudá-la a resolver esse problema, você foi convidado(a) a avaliar os códigos que ela desenvolveu para `hReady` e `oReady` (ela não fez o curso de S.O). Esses procedimentos devem esperar até que pelo menos dois átomos H e um átomo O estejam presentes e, então, uma das *threads* deve chamar `makeWater`, que apenas imprime uma mensagem dizendo que a reação da água foi realizada. Após a chamada `makeWater`, duas instâncias de `hReady` e uma instância de `oReady` devem retornar. A solução deve evitar *starvation* e *busy-waiting*. É assumido que a implementação do semáforo apresenta a ordem FIFO para os *wakeups*, isto é, a *thread* que espera há mais tempo em P() é sempre a próxima *thread* a ser acordada por uma chamada a V(). Você deve apontar (explicar) quais ais são os problemas apresentados pelas soluções abaixo.

```
Semaphore h_wait = 0;
Semaphore o_wait = 0;
int count = 0;

hReady()
{
    count++;
    if(count %2 == 1) {
        P(h_wait);
    } else {
        V(o_wait);
        P(h_wait);
    }

    return;
}
```

```
oReady()
{
    P(o_wait);
    makeWater();
    V(h_wait);
    V(h_wait);

    return;
}
```

```
Semaphore h_wait = 0;
Semaphore o_wait = 0;

hReady()
{
    V(o_wait)
    P(h_wait)

    return;
}
```

```
oReady()
{
    P(o_wait);
    P(o_wait);
    makeWater();
    V(h_wait);
    V(h_wait);

    return;
}
```

6. (1,5) Quais as vantagens do uso de monitores sobre semáforos? Escreva um monitor formado por dois procedimentos *request* e *release*, usados para gerenciar o uso de três unidades um recurso, sendo que pode ser alocada somente uma unidade do recurso a cada *request*.