Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação
Sistemas Operacionais
Prof.: Sérgio Vale Aguiar Campos – scampos@dcc.ufmg.br
Monitor: Hugo Valentim Barros – hbarros@dcc.ufmg.br

Turma: 2002/02

2ª Lista de Exercícios

1. Recall that a simple definition of fragmentation is “the inability to use memory that is free”. Of course, whenever we do any sort of allocation --- of CPU time quanta, of disk blocks, of cache space --- fragmentation is an issue. For example, if we allocate fixed size units, internal fragmentation is a problem. A more general statement would be “the inability to use X that is free”.

a) Give two examples where an OS intentionally cause internal fragmentation.
b) Does malloc minimize internal fragmentation? Justify.
c) What corresponds to internal fragmentation of a process's time quanta? How does the CPU scheduler minimize it?

2. Dijkstra posed each of the following solutions as a potential software solution to the critical section problem and the explained why they fail. Provide your explanation about why each one fails by describing how it violates one of the following rules for solutions to the critical section problem:

– Mutual Exclusion: only one process at a time should be allowed in its critical section.
– Isolation: If a set of processes tries to enter the critical section, only the process currently competing for the critical section participate in the selection of which process gets to enter the critical section.
– Starvation: Once a process attempts to enter its critical section, it cannot be postponed indefinitely.
– Bounded Wait: After a process requests entry into its critical section, only a bounded number of other process may be allowed to enter their related critical sections before the original process enters its critical section.

a) Proposed Solution 1

```
int turn;
void proc(int i) {
    while (TRUE) {
        <compute>
        while (turn != i);
        <critical section>
        turn = (i+1) % 2;
    }
}

turn = 1;
fork(proc, 0);
fork(proc, 1);
```

b) Proposed Solution 2

```
boolean flag[2];
void proc (int i) {
     while (TRUE) {
           <compute>
           while (flag[(i+1) mod 2]);
           flag[i] = true;
           <critical section>
           flag[i] = FALSE;
     }
}
flag[0] = flag[1] = false;
fork(proc, 0);
fork(proc, 1);
```

c) Proposed Solution 3

```
boolean flag[2];
void proc (int i) {
     while (TRUE) {
           <compute>
           flag[i] = true;
           while (flag[(i+1) mod 2]);
           <critical section>
           flag[i] = FALSE;
     }
}
flag[0] = flag[1] = false;
fork(proc, 0);
fork(proc, 1);
```

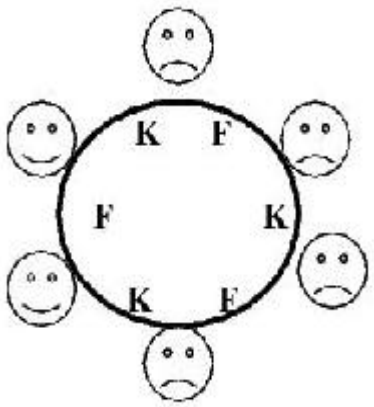
3. The notorius dining gourmand problem is as follows. Six gourmands sit around a table, with a large hunk of well-done roast beef in the middle. Forks (represented as 'F') and knives (represented as 'K') are arranges as shown below. (Gourmands that like their roast beef rare have frowny faces). Each gourmand obeys the following algorithm:

1) Grab the closest knife.

2) Grab the closest fork.

3) Carve and devour a piece of beef.

4) Put down the knife and fork where they used to be.



Can deadlock ever occur? Please be convincingly: either indicate why this algorithm satisfies ou deadlock conditions, or which one(s) it avoids. If you said that the system could deadlock, describe a reasonable deadlock avoidance scheme. If you said the scheme could not deadlock, say which one it uses.

4. Which (if any) of the following will (probably) improve CPU utilization? Explain your answer.

a) Install a faster CPU.
b) Install a bigger paging system.
c) Increase the degree of multiprogramming.
d) Decrease the degree of multiprogramming.
e) Install more main memory.
f) Install a faster hard disk, or multiple controllers with multiple hard disks.
g) Add prepaging to the page fetch algorithms.
h) Increase page size.

5. Describe three circumstances under which blocking I/O should be used. Describe three circumstancesm under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their device is ready?