**1. Suponha que, em um sistema, existem diversas aplicações que, em determinados pontos, consomem uma quantidade significativa de memória. Para evitar a exaustão da memória física do sistema, definiu-se que essas aplicações devem ser escritas de modo que apenas uma delas pode executar suas operações que demandam memória de cada vez. Em outras palavras, é necessário garantir a exclusão mútua no uso da memória pelas aplicações. Mostre como as aplicações poderiam ser construídas usando semáforos. Sua solução deve eliminar espera ocupada.**

**Dica: a aplicação pode ser modelada da seguinte forma:**

semaphore S=1;

void aplicacao() {

while (TRUE) {

1 faz\_algo();

2 down(S);

3 come\_memoria(); //região crítica

4 up(S);

}

}

**2. Como no caso anterior, mas agora as aplicações precisam de CPU e memória, e só podem executar quando tiverem ambos os recursos.**

semaphore S=1;

void aplicacao() {

while (TRUE) {

1 faz\_algo();

2 down(S);

3 come\_memoria(); //região crítica -> agora vira come\_mem\_CPU();

4 up(S);

}

}

**3. Suponha que, numa situação semelhante às anteriores, as aplicações podem necessitar de uso exclusivo de dois dos três recursos a seguir: CPU, disco e memória. Um gerente de recursos do SO escolhe, ao acaso, dois desses recursos, e libera a execução de uma aplicação que precise deles. Ao fim do uso, a aplicação devolve os recursos ao gerenciador. Escreva o pseudocódigo das aplicações e do gerenciador usando semáforos. Sua solução deve garantir que uma aplicação só execute quando tiver ambos os recursos necessários, e que ela não retenha um recurso quando o outro recurso do qual ela precisa esteja indisponível.**

Semaphore SC=0, SD=0, SM=0, SG=0

void aplic\_CD(){ //cpu e disco

while(TRUE){

faz\_algo();

//espera cpu e disco

usa\_cpu\_disco();

//libera cpu e disco

}

}

void aplic\_CM(){ //cpu e memória

while(TRUE){

faz\_algo();

//espera cpu+mem

usa\_cpu\_mem();

//libera cpu+mem

}

}

void gerente(){

while(TRUE){

rec=sorteia\_recursos(); //sorteia recursos, cd, dm, dm

if (rec==CD) {up(sc);up(sd)} //libera recursos

else if (rec==cm) {up(sc); up(sm);} //libera recursos

else if (rec==dm) {up(sd; up(sm);} //libera recursos

down (sg); //espera aplicação

}

}

void aplic\_DM(){

while(TRUE){

faz\_algo();

//espera disco+mem

usa\_disco\_mem();

//libera disco+mem

}

}

Tipos de problema de IPC

-Exclusão mútua

-Sincronização

Semáforos contadores -> Sinalização de condição

Quem espera down(s)

Quem sinaliza up(s)

**4. Considere um programa concorrente com três threads, X, Y e Z, mostradas abaixo.**

int n = 1; /\* variável compartilhada entre X, Y e Z \*/

Semaphore S1=0, S2=0

void X() {

n = n \* 16;

up(s1);

}

void Y() {

down(s2);

n = n / 7;

}

void Z() {

down(s1);

n = n + 40;

up(s2);

}

**Mostre como as threads poderiam usar semáforos para garantir que o resultado final de n seja sempre 8.**