

Prova pratica di Calcolatori Elettronici

C.d.L. in Ingegneria Informatica, Ordinamento DM 270

9 gennaio 2026

1. Siano date le seguenti dichiarazioni, contenute nel file cc.h:

```
struct st1 { int vi[4]; };
struct st2 { char vd[4]; };
class cl {
    char v1[4]; int v3[4]; long v2[4];
public:
    cl(st1 ss);
    cl(st1& s1, int ar2[]);
    cl elab1(char ar1[], st2 s2);
    void stampa() {
        for (int i = 0; i < 4; i++) cout << (int)v1[i] << ' '; cout << endl;
        for (int i = 0; i < 4; i++) cout << (int)v2[i] << ' '; cout << endl;
        for (int i = 0; i < 4; i++) cout << (int)v3[i] << ' '; cout << endl << endl;
    }
};
```

Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl cl::elab1(char ar1[], st2 s2)
{
    st1 s1;
    for (int i = 0; i < 4; i++) s1.vi[i] = ar1[i] + i;
    cl cla(s1);
    for (int i = 0; i < 4; i++) cla.v3[i] = s2.vd[i];
    return cla;
}
```

2. Un modo per evitare il problema del *blocco critico* nell'utilizzo dei semafori (di mutua esclusione) è di fare in modo che ogni processo acquisisca in una sola operazione indivisibile tutti i semafori di cui ha bisogno. Se qualche semaforo non può essere acquisito, allora non deve esserne acquisito nessuno: il processo si deve bloccare fino a quando tutti diventano disponibili.

Per supportare questo meccanismo aggiungiamo al nucleo la seguente primitiva:

```
void sem_multiwait(natl* sems, natl num);
```

Il parametro `sems` punta ad un array di identificatori di semaforo; il parametro `num` è la dimensione dell'array.

La primitiva `sem_multiwait()` provvede ad acquisire in modo indivisibile tutti i semafori `sems[0], ..., sems[num-1]`. Se tutti i semafori nella lista possono essere acquisiti senza bloccarsi, allora vengono acquisiti tutti. Altrimenti il processo viene sospeso in attesa che tutti diventino acquisibili.

È un errore se `num` è maggiore di `MAX_MULTIWAIT`, se l'array `sems` causa problemi di Cavallo di Troia, o se qualcuno degli identificatori di semaforo non è valido. In caso di errore, la primitiva abortisce il processo senza acquisire alcun semaforo.

Si noti che, nel caso in cui non tutti i semafori siano acquisibili, il processo potrebbe doversi accodare in più di una coda di semaforo, ma il nostro descrittore di processo possiede un solo campo `puntatore` e dunque può accodarsi ad una sola coda. Per risolvere il problema, introduciamo i processi *ombra*. Quando il processo deve accodarsi in più di una coda, crea un numero sufficiente di copie “ombra” del suo descrittore di processo, quindi inserisce ciascuna ombra in una coda diversa. Dal campo `id` dell'ombra è sempre possibile risalire al processo originario.

Al fine di realizzare questo meccanismo, aggiungiamo i seguenti campi al descrittore di processo:

```
natl mw_sems[MAX_MULTIWAIT];  
natl mw_num;  
bool shadow;
```

I primi `mw_num` elementi dell'array `mw_sems` contengono gli identificatori dei semafori passati l'ultima volta che il processo ha invocato `sem_multiwait`. Il campo `shadow` è `true` se e solo se il processo è un processo ombra. Modifichiamo quindi la primitiva `sem_signal` in modo che, quando deve svegliare un processo, risvegli sempre il primo processo (in ordine di priorità) che può acquisire tutti i semafori di cui ha bisogno. Se il processo risvegliato ha bisogno di altri semafori oltre a quello su cui opera la `sem_signal`, la primitiva acquisisce anche questi. La primitiva provvede anche a distruggere i descrittori ombra che non servono più.

Modificare i file `sistema.cpp` e `sistema.s` completando le parti mancanti.

Gestire correttamente eventuali *preemption*.