# Prova pratica di Calcolatori Elettronici

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

12 febbraio 2025

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

```
struct st1 { int vi[4]; };
struct st2 { char vd[4]; };
class cl {
        long v2[4]; char v1[4]; char v3[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(st1 ss)
{
        for (int i = 0; i < 4; i++) {
                v1[i] = ss.vi[i]; v2[i] = ss.vi[i] * 2;
                v3[i] = 2 * ss.vi[i];
        }
}
cl::cl(st1& s1, int ar2[])
{
        for (int i = 0; i < 4; i++) {
                v1[i] = s1.vi[i]; v2[i] = s1.vi[i] * 4;
                v3[i] = ar2[i];
        }
}
```

2. Due processi possono comunicare tramite una `pipe`, un canale con una estremità di scrittura e una di lettura attraverso il quale viaggia una sequenza di caratteri. I caratteri inviati dall'estremità di scrittura possono essere letti dall'estremità di lettura. Il sistema contiene un numero prefissato `pipe`, numerate da zero a `MAX_PIPE` meno uno. Un processo che voglia leggere o scrivere in una pipe deve prima *aprire* la corrispondente estremità, bloccandosi (se necessario) in attesa che venga aperta anche l'altra estremità. All'apertura, i processi ricevono un *identificatore privato* da zero a `MAX_OPEN_PIPES` meno uno poi lo utilizzano per riferisi all'estremità della pipe durante le successive operazioni di lettura o scrittura. Un

processo può aprire più di una estremità di qualunque pipe, fino ad un massimo di `MAX_OPEN_PIPES`. Le estremità devono essere *chiuse* quando non servono più. Quando un processo termina, il sistema provvede comunque a chiudere tutte le estremità che risultavano ancora aperte dal processo. Le operazioni di lettura o scrittura sono sincrone: lo scrittore di blocca in attesa di aver trasferito tutti i byte, e il lettore si blocca in attesa di averli ricevuti tutti. Se una delle due estremità viene chiusa mentre un processo è in attesa sull'altra, il processo si risveglia dalla primitiva di lettura o scrittura e riceve il valore `false`.

Per realizzare le `pipe` aggiungiamo le seguenti primitive (abortiscono il processo in caso di errore):

- `natl openpipe(natl pipeid, bool writer)` (tipo 0x2a, già realizzata): Apre l'estremità di lettura (se `writer` è `false`) o di scrittura (se `writer` è `true`) della pipe con identificatore `pipeid`. È un errore se la pipe non esiste, o se il processo ha troppe pipe aperte. Restituisce l'identificatore privato della pipe, o `0xFFFFFFFF` se l'estremità della pipe è già aperta (dallo stesso o da un altro processo).
- `bool writepipe(natl slotid, const char *buf, natl n)` (tipo 0x2b, realizzara in parte): Invia `n` caratteri dal buffer `buf` sulla pipe con identificatore privato `slotid`. È un errore se la pipe `slotid` non è l'identificatore valido di una pipe aperta in scrittura, o se il buffer non è accessibile in lettura a tutti i processi. Restituisce `false` se non è stato possibile trasferire tutti i byte.
- `bool readpipe(natl slotid, char *buf, natl n)` (tipo 0x2c, realizzata in parte): Riceve `n` caratteri dal dalla pipe di identificatore privato `slotid` e li scrive nel buffer `buf`. È un errore se la pipe `slotid` non è l'identificatore valido di una pipe aperta in lettura, o se il buffer non è accessibile in scrittura da tutti i processi. Restituisce `false` se non è stato possibile ricevere tutti i byte.
- `void closepipe(natl slotid)` (tipo 0x2d, da realizzare): Chiude una estremità di una pipe. È un errore se `slotid` non è l'indentificatore privato valido di una estremità di pipe ancora aperta.

Per descrivere una `pipe` aggiungiamo al nucleo la seguente struttura dati:

```
struct des_pipe {
    natl writer;
    natl reader;
    natl w_pending;
    const char *w_buf;
    natl r_pending;
    char *r_buf;
};
```

Il campo `writer` identifica il processo che ha aperto l'estremità di scrittura, e il campo `reader` identifica il processo che ha aperto l'estremità di lettura. La pipe è libera se entrambi sono zero. I campi assumono il valore `0xFFFFFFFF` se la corrispondente è estremità è stata chiusa, ma l'altra è ancora aperta. Il campo `w_buf` punta al prossimo byte da trasferire dal buffer dello scrittore, ed è diverso da `nullptr` solo se lo scrittore è bloccato in attesa di completare il trasferimento. Analogamente, il capo `r_buf` punta alla prossima locazione da riempire nel buffer del ricevitore, ed è diverso da `nullptr` solo se il lettore è bloccato in attesa di ricevere tutti i byte. I campi `w_pending` e `r_pending` contengono il numero di byte ancora da scrivere o leggere, rispettivamente.

Inoltre, aggiungiamo il seguente campo ai descrittori di processo:

```
    natl mypipes[MAX_OPEN_PIPES];
```

L'array è indicizzato tramite gli identificatori privati. Ogni entrata contiene `0xFFFFFFFF` se l'identificatore non è usato, oppure contiene l'identificatore della pipe di cui il processo ha aperto una estremità.

Modificare il file `sistema.cpp` in modo da realizzare le parti mancanti.