Prova pratica di Calcolatori Elettronici

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

9 gennaio 2025

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

```
struct st1 { char vc[4]; }; struct st2 { int vd[4]; };
class cl
{       st1 s; long v[4];
public:
        cl(char c, st2 s2);
        void elab1(st1& s1, st2& s2);
        void stampa()
        {       int i;
                for (i=0;i<4;i++) cout << s.vc[i] << ' '; cout << endl;
                for (i=0;i<4;i++) cout << v[i] << ' '; cout << endl << endl;
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
void cl::elab1(st1& s1, st2& s2) {
        cl cla('a', s2);
        for (int i = 0; i < 4; i++) {
                if (s.vc[i] < s1.vc[i]) {
                        s.vc[i] = cla.s.vc[i];
                        v[i] = cla.v[i];
                }
        }
}
```

2. Aggiungiamo al nucleo il meccanismo della *message queue (MQ)*.

   Un qualunque processo può accodare un nuovo messaggio sulla MQ. I processi che vogliono leggere i messaggi accodati nella MQ devono prima registrarsi. Registrandosi acquisiscono il diritto di leggere tutti i messaggi accodati da quel momento in poi. Ogni messaggio accodato deve essere letto da tutti i processi che risultavano registrati nel momento in cui il messaggio era stato accodato. A quel punto diciamo che il messaggio è letto *completamente* e può essere rimosso dalla coda.

   I processi registrati come lettori non possono accodare messaggi.

   La MQ ha una dimensione finita e viene usata come un array circolare. I processi scrittori che trovano la MQ piena ricevono un errore. I processi lettori si bloccano quando non trovano messaggi che non avevano già letto e si bloccano in attesa di un nuovo messaggio. I processi scrittori si bloccano fino a quando il loro messaggio non è stato letto completamente.

   Per realizzare il meccanismo appena descritto introduciamo le seguenti strutture dati:

```
struct mq_msg {
    des_proc *sender;
    natq toread;
};
struct mq_des {
    natq nreaders;
    mq_msg mq[MQ_SIZE];
    natq head;
    natq tail;
    des_proc *w_readers;
    natq nwaiting;
};
```

La struttura `mq_msg` descrive un messaggio, con il campo `sender` che punta al processo che lo ha inviato (i dettagli del messaggio si trovano nel descrittore del processo, si veda più avanti), mentre il campo `toread` conta il numero di processi che hanno diritto a leggerlo e ancora non l'hanno fatto. La struttura `mq_des` descrive la MQ. Il campo `nreaders` conta il numero di processi attivi registrati per la lettura; il campo `mq` è l'array circolare di messaggi; `head` è l'indice della testa dell'array (posizione dell'ultimo messaggio non ancora completamente letto) mentre `tail` è l'indice della coda dell'array (posizione in cui andrà inserito il prossimo messaggio); `w_readers` è una coda di processi bloccati in attesa di poter ricevere un messaggio, mentre `nwaiting` conta i processi accodati su `w_readers`.

Aggiungiamo anche i seguenti campi ai descrittori di processo:

```
bool mq_reader;
natq mq_ntr;
char *mq_buf;
natq mq_buflen;
```

Il campo `mq_reader` vale true se il processo è registrato come lettore della MQ; il campo `mq_ntr` è significativo per i processi registrati come lettori, e contiene l'indice nella MQ del prossimo messaggio che il processo deve leggere; i campi `mq_buf` e `mq_buflen` hanno un significato diverso per i processi lettori e scrittori: nei processi scrittori `mq_buf` punta al messaggio da inviare, di lunghezza `mq_buflen`; nei processi lettori `mq_buf` punta al buffer, di capienza `mq_buflen`, in cui il lettore vuole ricevere il messaggio. In entrambi i casi i buffer possono trovarsi nella memoria utente/privata dei rispettivi processi.

Aggiungiamo inoltre le seguenti primitive:

- `void mq_reg()` (già realizzata): registra il processo come lettore della MQ; è un errore se il processo è già registrato;
- `bool mq_send(char *msg, natq len)` (realizzata in parte): accoda un nuovo messaggio sulla MQ; è un errore se il processo è registrato come lettore; restituisce `false` se non è stato possibile accodare il messaggio perché la MQ era piena;
- `natq mq_recv(char *buf, natq size)` (realizzata in parte): restituisce il prossimo messaggio accodato nella MQ e non ancora letto dal processo. È un errore se il processo non è registrato come lettore. Se il messaggio è troppo grande per il buffer, viene considerato come letto, ma il buffer non viene modificato. In ogni caso, restituisce la lunghezza del messaggio.

Le primitive abortiscono il processo chiamante in caso di errore e tengono conto dei problemi di Cavallo di Troia e della priorità tra i processi.

Attenzione: quando un processo registrato come lettore termina, tutti i messaggi già accodati nella MQ e non ancora letti dal processo vanno gestiti opportunamente (di fatto come se li avesse letti); inoltre, il processo non deve essere più contato tra i processi registrati.

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