Prova pratica di Calcolatori Elettronici

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

14 febbraio 2024

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

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

   Realizzare in Assembler GCC le funzioni membro seguenti.

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

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.

   Anche i processi registrati come lettori possono accodare messaggi, ma in quel caso il processo non viene contato tra i processi che devono leggere il messaggio.

   La MQ ha una dimensione finita e viene usata come un array circolare. I processi scrittori che trovano la MQ piena si bloccano in attesa che un messaggio venga completamente letto (liberando dunque una posizione). I processi lettori, invece, si bloccano quando non trovano messaggi che non avevano già letto e si bloccano in attesa di un nuovo messaggio.

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

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

La struttura `mq_msg` descrive un messaggio, con il campo `msg` che è il messaggio vero e proprio, 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_senders` è una coda di processi bloccati in attesa di poter accodare un messaggio; `w_readers` è una coda di processi bloccati in attesa di poter ricevere un messaggio.

Aggiungiamo anche i seguenti campi ai descrittori di processo:

```
    bool mq_reader;
    natq mq_ntr;
    natq mq_msg;
```

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; il campo `mq_msg` è significativo se il processo è bloccato in attesa di poter accodare un messaggio nella MQ, e contiene il messaggio da accodare.

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;
- `void mq_send(natq msg)` (già realizzata): accoda un nuovo messaggio sulla MQ;
- `natq mq_recv()` (da realizzare): restituisce il prossimo messaggio accodato nella MQ e non ancora letto dal processo. È un errore se il processo non è registrato come lettore.

Le primitive abortiscono il processo chiamante in caso di errore e tengono conto 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.