# 2024-02-14 - Message Queue (Sincronizzazione)

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;
};
```

```
/// Descrittore di messaggio
struct mq_msg {
    /// contenuto del messaggio
    natq msg;
    /// processi che devono ancora leggere il messaggio
    natq toread;
};

/// Descrittore di MQ
struct mq_des {
    /// numero di processi registrati per la lettura
    natq nreaders;
```

```
    /// array circolare di messaggi
    mq_msg mq[MQ_SIZE];
    /// testa dell'array (punto di estrazione)
    natq head;
    /// coda dell'array (punto di inserimento)
    natq tail;

    /// processi bloccati in scrittura
    des_proc *w_senders;
    /// processi bloccati in lettura
    des_proc *w_readers;
} message_queue;

natq mq_next(natq idx)
{
    return (idx + 1) % MQ_SIZE;
}

bool mq_full()
{
    mq_des *mq = &message_queue;

    return mq_next(mq->tail) == mq->head;
}

natq mq_nmsg()
{
    mq_des *mq = &message_queue;
    natq tail = mq->tail;

    if (tail < mq->head)
        tail += MQ_SIZE;
    return tail - mq->head;
}
```

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:

```
struct des_proc {
    ...
    /// true se il processo è registrato come lettore
    bool mq_reader;
    /// indice del prossimo messagio da leggere
    natq mq_ntr;
    /// messaggio da accodare se il processo è bloccato nella mq_send()
    natq mq_msg;
};

des_proc* crea_processo(void f(natq), natq a, int prio, char liv)
{
    des_proc*   p;
    ...
    p->mq_reader = false;
    p->mq_msg = 0;
    p->mq_ntr = 0;
}
```

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.

```
extern "C" void c_mq_reg()
{
    if (esecuzione->mq_reader) {
        flog(LOG_WARN, "processo gia' registrato come lettore");
        c_abort_p();
        return;
    }

    mq_des *mq = &message_queue;
```

```
    esecuzione->mq_reader = true;
    // il primo messaggio da leggere è il primo che verrà inserito da ora
    esecuzione->mq_ntr = mq->tail;
    mq->nreaders++;
}

extern "C" void c_mq_send(natq msg)
{
    mq_des *mq = &message_queue;

    // lo scrittore si blocca se la coda è piena
    if (mq_full()) {
        esecuzione->mq_msg = msg;
        inserimento_lista(mq->w_senders, esecuzione);
        schedulatore();
        return;
    }

    // altrimenti inserisce in coda un nuovo messaggio
    mq_msg *m = &mq->mq[mq->tail];
    m->msg = msg;
    m->toread = mq->nreaders;
    mq->tail = mq_next(mq->tail);
    // se lo scrittore è anche registrato come lettore, non va contato tra
    // i lettori di questo messaggio
    if (esecuzione->mq_reader)
        m->toread--;
    // se ci sono lettori bloccati, vanno svegliati e gli va recapitato
    // il nuovo messaggio
    if (mq->w_readers) {
        inspronti();
        do {
            des_proc *p = rimozione_lista(mq->w_readers);
            p->contesto[I_RAX] = msg;
            p->mq_ntr = mq_next(p->mq_ntr);
            m->toread--;
            inserimento_lista(pronti, p);
        } while (mq->w_readers);
        schedulatore();
    }
    // se il messaggio è stato letto completamente, lo si estrae dalla coda
    if (!m->toread) {
        mq->head = mq_next(mq->head);
    }
}
```

**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.

```
extern "C" void c_mq_recv()
{
    mq_des *mq = &message_queue;

    if (!esecuzione->mq_reader) {
        flog(LOG_WARN, "processo non registrato come lettore");
        c_abort_p();
        return;
    }

    if (esecuzione->mq_ntr == mq->tail) {
        inserimento_lista(mq->w_readers, esecuzione);
        schedulatore();
        return;
    }

    mq_msg *m = &mq->mq[esecuzione->mq_ntr];
    esecuzione->contesto[I_RAX] = m->msg;
    esecuzione->mq_ntr = mq_next(esecuzione->mq_ntr);
    m->toread--;
    if (!m->toread) {
        inspronti();
        mq->head = mq_next(mq->head);
        if (mq->w_senders) {
            des_proc *p = rimozione_lista(mq->w_senders);
            m = &mq->mq[mq->tail];
            m->msg = p->mq_msg;
            m->toread = mq->nreaders;
            if (p->mq_reader)
                m->toread--;
            mq->tail = mq_next(mq->tail);
            inserimento_lista(pronti, p);
            while (mq->w_readers) {
                p = rimozione_lista(mq->w_readers);
                p->contesto[I_RAX] = m->msg;
                p->mq_ntr = mq_next(p->mq_ntr);
                m->toread--;
                inserimento_lista(pronti, p);
            }
        }
```

```
        schedulatore();
    }
}

void distruggi_processo(des_proc* p)
{
    if (p->mq_reader)
    {
        mq_des *mq = &message_queue;
        while (p->mq_ntr != mq->tail) {
            mq_msg *m = &mq->mq[p->mq_ntr];
            m->toread--;
            if (!m->toread)
                mq->head = mq_next(mq->head);
            p->mq_ntr = mq_next(p->mq_ntr);
        }
        mq->nreaders--;
        while (!mq_full() && mq->w_senders) {
            des_proc *s = rimozione_lista(mq->w_senders);
            mq_msg *m = &mq->mq[mq->tail];
            m->msg = s->mq_msg;
            m->toread = mq->nreaders;
            if (s->mq_reader)
                m->toread--;
            inserimento_lista(pronti, s);
            while (mq->w_readers) {
                s = rimozione_lista(mq->w_readers);
                s->contesto[I_RAX] = m->msg;
                m->toread--;
                s->mq_ntr = mq_next(s->mq_ntr);
                inserimento_lista(pronti, s);
            }
            if (!m->toread)
                mq->head = mq_next(mq->head);
        }
    }

    ...
}
```