2025-01-09 - Message queue

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;
};

/// Unica istanza di mq_des
mq_des 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;
}
```

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:

```
struct des_proc {
    ...

    /// true se il processo è registrato come lettore
    bool mq_reader;
    /// indice del prossimo messagio da leggere
    natq mq_ntr;
    /// @brief puntatore al buffer/messaggio se il processo è bloccato nella MQ, nullptr al
    char *mq_buf;
    /// dimensione del buffer/messaggio puntato da @ref mq_buf
    natq mq_buflen;
};

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

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)
            {
                inserimento_lista(pronti, m->sender);
                m->sender = nullptr;
                mq->head = mq_next(mq->head);
            }
            p->mq_ntr = mq_next(p->mq_ntr);
        }
        mq->nreaders--;
    }
    ...
}
```

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.

```
void mq_msgpush(des_proc *receiver)
{
    if (esecuzione->mq_buflen > receiver->mq_buflen)
        return;
    char *src = esecuzione->mq_buf,
         *dst = receiver->mq_buf;
    natq  rem = esecuzione->mq_buflen;
```

```
    while (rem) {
        vaddr vdst = int_cast<vaddr>(dst);
        natq tocopy = DIM_PAGINA - (vdst % DIM_PAGINA);
        if (rem < tocopy)
            tocopy = rem;
        char *pdst = ptr_cast<char>(trasforma(receiver->cr3, vdst));
        memcpy(pdst, src, tocopy);
        dst += tocopy;
        src += tocopy;
        rem -= tocopy;
    }
}
```

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.

```
void mq_msgpull(des_proc *sender)
{
    if (sender->mq_buflen > esecuzione->mq_buflen)
        return;
    char *src = sender->mq_buf,
         *dst = esecuzione->mq_buf;
    natq  rem = sender->mq_buflen;
    while (rem)
    {
        vaddr vsrc = int_cast<vaddr>(src);
        natq tocopy = DIM_PAGINA - (vsrc % DIM_PAGINA);
        if (rem < tocopy)
            tocopy = rem;
        char *psrc = ptr_cast<char>(trasforma(sender->cr3, vsrc));
        memcpy(dst, psrc, tocopy);
        dst += tocopy;
        src += tocopy;
        rem -= tocopy;
    }
}

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(char *msg, natq len)
{
    mq_des *mq = &message_queue;

    if (esecuzione->mq_reader) {
        flog(LOG_WARN, "mq_send: il processo e' un reader");
        c_abort_p();
        return;
    }

    if (!c_access(int_cast<vaddr>(msg), len, false, false)) {
        flog(LOG_WARN, "mq_send: parametri non validi");
        c_abort_p();
        return;
    }

    // fallimento se la coda è piena
    if (mq_full()) {
        esecuzione->contesto[I_RAX] = false;
        return;
    }

    // altrimenti inserisce in coda un nuovo messaggio
    mq_msg *m = &mq->mq[mq->tail];
    esecuzione->mq_buf = msg;
    esecuzione->mq_buflen = len;
    m->sender = esecuzione;
    m->toread = mq->nreaders;
    mq->tail = mq_next(mq->tail);
    // se ci sono lettori bloccati, vanno svegliati e gli va recapitato
    // il nuovo messaggio
    if (mq->nwaiting) {
        if (mq->nwaiting == mq->nreaders)
            inspronti();
```

```
            do {
                des_proc *p = rimozione_lista(mq->w_readers);
                mq_msgpush(p);
                p->contesto[I_RAX ] = esecuzione->mq_buflen;
                p->mq_ntr = mq_next(p->mq_ntr);
                m->toread--;
                mq->nwaiting--;
                inserimento_lista(pronti, p);
            } while (mq->nwaiting);
            schedulatore();
        }
        // se il messaggio è stato letto completamente, lo si estrae dalla coda
        if (!m->toread) {
            esecuzione->mq_buf = nullptr;
            esecuzione->mq_buflen = 0;
            m->sender = nullptr;
            mq->head = mq_next(mq->head);
        }
    }

    extern "C" void c_mq_recv(char *buf, natq size)
    {
        mq_des *mq = &message_queue;

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

        if (!c_access(int_cast<vaddr>(buf), size, true, false)) {
            flog(LOG_WARN, "mq_recv: parametro buf non valido");
            c_abort_p();
            return;
        }

        esecuzione->mq_buf = buf;
        esecuzione->mq_buflen = size;

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

```
    mq_msg *m = &mq->mq[esecuzione->mq_ntr];
    des_proc *sender = m->sender;
    esecuzione->contesto[I_RAX] = sender->mq_buflen;
    mq_msgpull(sender);
    esecuzione->mq_ntr = mq_next(esecuzione->mq_ntr);
    m->toread--;
    if (!m->toread) {
        sender->mq_buf = nullptr;
        sender->mq_buflen = 0;
        m->sender = nullptr;
        mq->head = mq_next(mq->head);
        inspronti();
        inserimento_lista(pronti, sender);
        schedulatore();
    }
}
```