Implementazione delle primitive di comunicazione

Send/receive sincrona, assumendo che la condivisione avvenga mediante riferimenti logici coincidenti, invece che con capability.

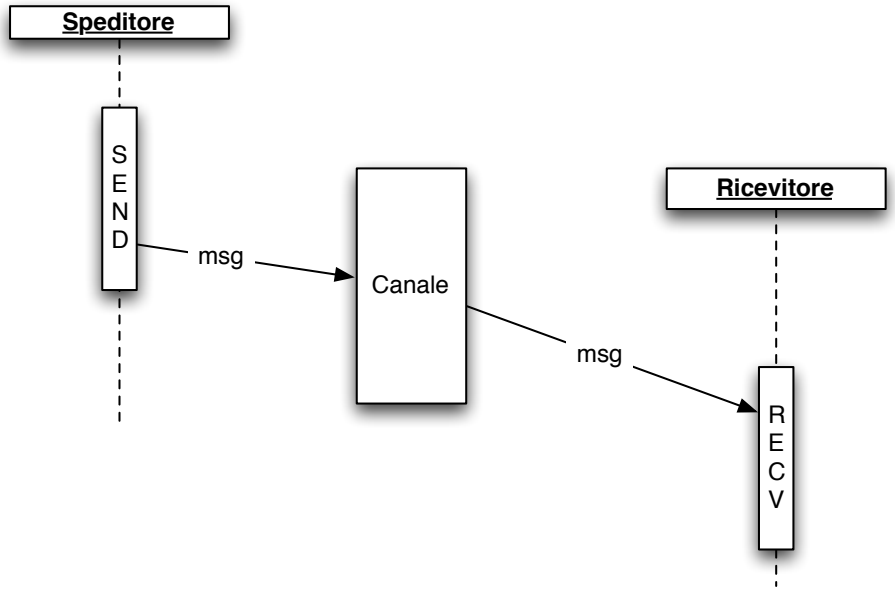
Struttura dati canale

```
{
  boolean wait;   // registra che un partner della comunicazione è in attesa
  boolean vuoto;  // registra che il buffer sia vuoto (inizializzato a true)
  int len;        // lunghezza del messaggio (sizeof(T))
  PTR pcb;        // indirizzo del PCB del processo partner in attesa
  PTR indvtg;     // indirizzo della variabile targa
  T buffer;       // spazio per la memorizzazione del messaggio
                  // (T è il tipo del canale)
}
```

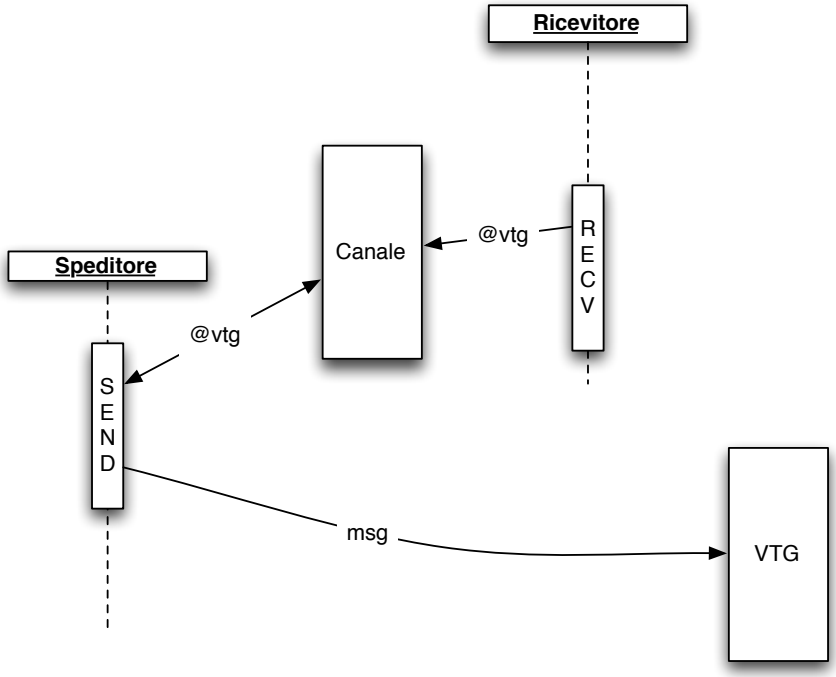
Codice ad alto livello

```
send(ch, msg) {
  if(wait) {
      // il destinatario ha già cominciato la receive ed è in attesa
      // pcb contiene il puntatore al suo PCB per la sveglia
      // vtg contiene l'indirizzo della variabile targa
      wait = false;
      copia(msg,indvtg);      // msg -> (msgVtg)
      vuoto = true;
      sveglia(pcb);
  } else {
      // buffer libero, il destinatario non ha ancora invocato la receive
      // posso copiare il messaggio nel buffer e mettermi in attesa
      wait = true;
      copia(msg,buffer);            // msg -> buffer
      vuoto = false;
      commutaContesto();
  }
}

receive(ch, vtg) {
  if(vuoto) {
      // registro pcb, indirizzo vtg e mi blocco
      wait = true;
      pcb = mioPcb();
      indvtg = indirizzo(vtg);
      commutaContesto();
  } else {
      // c'e' già un messaggio nel canale e il mittente è in attesa
      wait = false;
      vtg = buffer;
      vuoto = true;
      sveglia(pcb);
  }
}
```

In questo caso, assumiamo che la copia dal mittente al destinatario avvenga mediante passaggio nel buffer di canale quando il mittente esegue la send prima della receive:



e che la copia avvenga direttamente dal mittente alla variabile targa del destinatario quando la receive viene invocata prima della send:



Implementazione in ASM D-RISC

Strutture dati utilizzate

tabella dei canali:

Vettore di maxChan posizioni con base RbaseTabChan.
La posizione i contiene l'indirizzo del canale i-esimo

canale:

offset 0: wait (0 è false, 1 è true)
offset 1: vuoto (0 è false, 1 è true)
offset 2: len (lunghezza in parole del messaggio)
offset 3: pcb (indice nella tabella dei PCB per il PCB del processo in attesa)
offset 4: indvtg (indirizzo logico della variabile targa)
offset 5 e seguenti: buffer (buffer per il messaggio)

tabella dei PCB:

vettore di indirizzi di PCB
la posizione i contiene l'indirizzo del PCB del processo di identificatore i

Send

```
send:  DI
       LD RbasePcb, #offsetTabCh, RbaseTabChan  // indirizzo base della tabella canali
       LD RbaseTabChan, Rchan, RbaseChan        // mi procuro l'indirizzo del canale

       LD RbaseChan, #0, Rwait                  // flag wait
       IF=0 Rwait, else                         // se !wait, ramo else (false = 0)

then:  ST RbaseChan, #0, R0                     // wait = false (0!)

       LD RbaseChan, #4, Rvtg                   // indirizzo della vtg dal canale
       LD RbaseChan, #2, Rlen                   // copia del messaggio da msg a vtg
       CLEAR Ri                                 // inizializzazione i e N
loop:  LD Rmsg, Ri, Rtemp                       // lettura parola del messaggio
       ST Rvtg, Ri, RTemp                       // scrittura parola nella vtg
       INC Ri                                   // i++
       IF< Ri, Rlen, loop                       // se <N cicla

       ADD R0, #1, Rtemp                        // true
       ST RbaseChan, #1, Rtemp                  // vuoto = true

       LD RbaseChan, #3, RinPcbSveglia          // parametro della sveglia processo
       CALL Rsveglia, RretSend                  // chiama sveglia processo
```

```
        GOTO cont                                   // fine ramo then

else:   CLEAR Ri                                    // variabile di iterazione
        LD RbaseChan, #5, Rbuffer                   // indirizzo buffer
        LD RbaseChan, #2, Rlen                      // parole da trasferire

loop1:  LD Rmsg, Ri, Rtemp                          // copia msg ->
        ST Rbuffer, Ri, Rtemp                       //              -> buffer
        INC Ri                                      // i++
        IF< Ri, Rlen, loop1                         // continua

        ST RbaseChan, #3, Rpcb                      // memorizza indirizzo PCB

        ADD R0, #1, Rtemp                           // true
        ST RbaseChan, #0, Rtemp                     // wait = true

        ST RbaseChan, #1, R0                        // vuoto = false

        CALL Rcommuta_contesto, Rret1               // commuta contesto

cont:   GOTO Rret, EI                               // ritorna al chiamante
```

Receive

```
receive: DI
        LD RbasePcb, #offsetTabCh, RbaseTabChan // indirizzo base della tabella canali
        LD RbaseTabChan, Rchan, RbaseChan       // mi procuro l'indirizzo del canale

        LD RbaseChan, #1, Rvuoto                // flag vuoto
        IF=0 Rvuoto, else                       // se !wait, ramo else (false = 0)

then:   ADD R0, #1, Rtrue
        ST RbaseChan, #0, Rtrue                 // wait = true

        ST RbaseChan, #3, Rpcb                  // pcb = mioPcb

        ST RbaseChan, #4, Rvtg                  // indvtg = indirizzo(vtg)

        CALL commuta_contesto, Rret1            // commuta_contesto();
                                                // salva Rret1 come IC per ripristino
        // questo è il punto dove la sveglia processo mi fa ritornare
        GOTO cont

else:   ST RbaseChan, #0, R0                    // wait = false

        // copia messaggio da buffer a variabile targa
        CLEAR Ri
        LD RbaseChan, #2, Rlen
        LD RbaseChan, #5, Rbuffer
loop:   LD Rbuffer, Ri, Rtemp
```

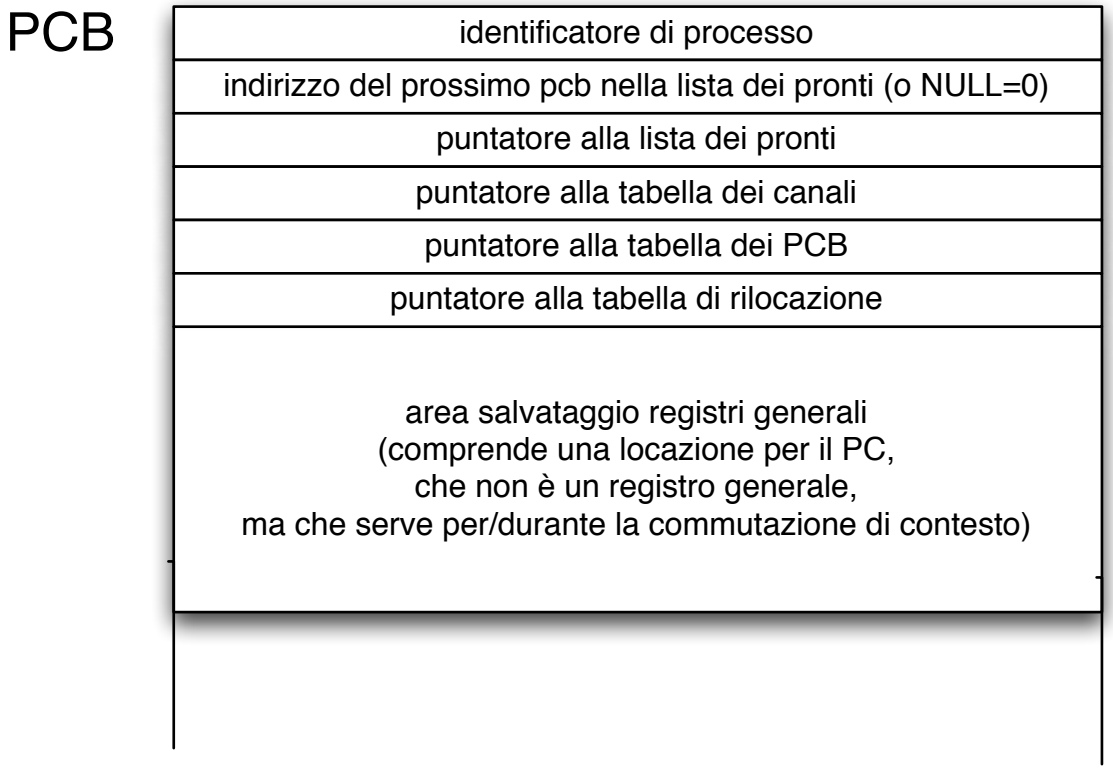
```
        ST Rbuffer, Ri, Rtemp
        INC Ri
        IF< Ri, Rlen, loop

        ST RbaseChan, #1, Rtrue                 // vuoto = true

        LD RbaseChan, #3, RinPcbSveglia         // pcb processo da svegliare
        CALL RsvegliaProcesso, Rret1            // sveglia(pcb);

cont:   GOTO Rret, EI                           // ritorna controllo al chiamante
```

Commuta contesto

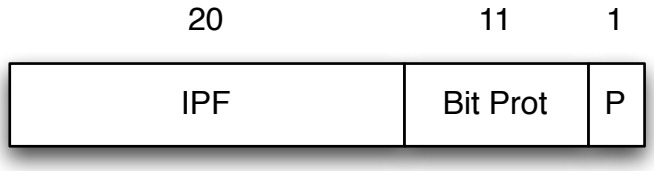
La commuta contesto deve:

1) salvare lo stato del processo corrente nell'area dedicata nel suo PCB
2) salvare in tale area il valore corretto del PC di ritorno
3) accedere alla testa della lista dei pronti
4) trovare il PCB del processo da mandare in esecuzione,
5) modificare la testa della lista dei processi pronti
6) trovare l'indirizzo della tabril, mediante il PCB
7) ripristinare i registri dal PCB appena trovato
8) saltare alla istruzione puntata dal PC trovato nel PCB con la start process

Assumiamo che la lista dei processi pronti sia implementata mediante un campo del PCB che contiene il puntatore al prossimo PCB della lista.
Una locazione PRONTI, accessibile perchè il puntatore logico è presente in tutti i PCB permette di recuperare il primo PCB della lista dei pronti. Un puntatore pari a 0 ha significato di fine lista (NULL)



Assumiamo che una entry per la tab ril contenga (assumiamo che le pagine siano da 4K, e che quindi il numero di pagina (logio o fisico) richieda 32 - 12 = 20 bit per la sua rappresentazione):



```
commuta_contesto:                   // parametri Rpcb indirizzo del PCB -> wait
                                    // Rret indirizzo di ritorno della procedura (!)
    // salvataggio stato
    LD Rpcb, #offsetAreaSalvataggio, Rsalva
    ST Rsalva, #1, R1
    ST Rsalva, #2, R2
    ...
    ST Rsalva,#59, R59              // assumendo che gli altri reg siano gestiti a parte
                                    // cioè inizializzati con i puntatori giusti una volta
                                    // per tutte nell'area di salvataggio registri del PCB

    // salvataggio IC nell'area relativa al salvataggio registri
    ST Rsalva, #offsetIC, Rret      // è stata chiamata con param Rpcb e Rret

    // PCB del processo da mandare in esecuzione dalla lista dei pronti
    // se la lista è con puntatori logici al prossimo pcb, la variabile pronti è
    // già il puntatore al PCB del processo da mandare in esecuzione
    LD Rpcb, #offsetPronti, Rpronti         // testa della lista dei pronti
    LD Rpcb, #offsetListaPronti, Rpcb       // assumiamo che non sia mai vuota
                                            // eventualmente c'è il processo vuoto
    // aggiustamento della testa della lista dei processi pronti
    LD Rpcb, #offsetNextListaPronti, RnuovaTestaPronti
    ST Rpronti, #0, RnuovaTestaPronti

    // indirizzo della tabril del nuovo processo da comunicare alla MMU
    // mediante una START_PROCESS

    LD Rpcb, #offestIndLogTabRil, Rtabril   // indirizzo tabril da PCB del nuovo proc
    // trovo indirizzo fisico da mandare a MMU
    SHR 12, Rtabril, Rtemp                  // shift a ds di 12 pos, risultato in Rtemp
    LD Rtabril, Rtemp, RentryTabRil
    SHR 12, RentryTabRil, RentryTabRil      // questi due shift
    SHL 12, RentryTabRil, Rtabril2MMU       // cancellano i bit bassi della entry
                                            // lasciando nel reg l'indirizzo fisico
                                            // (si assume che sia allineato alla pagina!)

    // ripristino registri
    LD Rpcb, #offsetAreaSalvataggio, Rsalva     // indirizzo area salvataggio
    LD Rsalva , #1, R1
    LD Rsalva, #2, R2
    ...
    LD Rsalva, #62, R62

    // recupera il valore corretto di IC
    LD Rsalva, #offsetIC, Ric   // registro cui si salta alla fine per riprendere l'esecuzione
    LD Rsalva,#63, R63          // Rsalva sarà R63 così è scritto quando non serve più

    // fai ripartire il processo
    START_PROCESS Rtabril2MMU, Ric
```

La parte bordata è relativa al recupero dell'indirizzo fisico della tabella di rilocazione da mandare alla MMU mediante la START_PROCESS, nell'ipotesi che la MMU riceva appunto un indirizzo fisico. In questo caso si è assunto che le tabril dei vari processi siano condivise, come la tabella dei canali o dei PCB, e pertanto, conoscendone l'indirizzo logico, possiamo accedervi anche se in quel momento è in uso la tabella di rilocazioni di un altro processo. Se invece volessimo utilizzare nella START_PROCESS un indirizzo logico, questo codice potrebbe essere sostituito dal seguente codice:

```
// mi procuro l'indirizzo logico della Tabril del nuovo processo
// da passare con la START_PROCESS alla MMU
LD Rpcb, #offestIndLogTabRil, Rtabril2MMU
```

Sveglia processo

Parametri della sveglia processo:
Rpcb: indirizzo del PCB del processo da svegliare
Rret: indirizzo di ritorno della procedura

Assunzione:
nella lista dei pronti c'e' un id (indice del PCB nella tabella dei processi)
l'id è accessibile da PCB (è un campo del PCB)

Sorrimento della lista dei pronti per trovare l'ultima posizione

punt = pronti;
while(pronti != NULL) { punt = pronti; pronti = next(pronti); }

```
sveglia:LD Rpcb, #offsetListaPronti, Rpronti          // testa lista pronti
        // si scorre la lista fino a che non si arriva in fondo

        // punt=pronti; while(pronti != 0) { punt = pronti; pronti = pronti.next; }
        MOV Rpronti, Rpunt
loop:   IF=0 Rpronti, cont
        MOV Rpronti, Rpunt
        LD Rpronti, #offsetNextListaPronti, Rpronti
        goto loop

cont:   // link del PCB alla lista
        ST Rpcb, #offsetNextListaPronti, R0      // puntatore null, sono l'ultimo
        ST Rpunt, #offsetNextListaPronti, Rpcb // metto il PCB in lista

        // adesso ho finito posso restituire il controllo al chiamante
        GOTO Rret
```
