Primitive sincrone

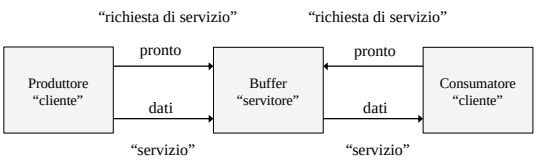
- Processo mittente e processo ricevente *si sincronizzano* al momento della comunicazione.
- Il trasferimento dell'informazione avviene non appena entrambi i processi sono *pronti a comunicare* (*rendez-vous*).
- Il modello di comunicazione adottato prevede che ad un processo siano associati *canali privi di memoria*, uno per ogni tipo di messaggio che il processo può ricevere.

Produttore-consumatore

- 3 Processi: Produttore, Consumatore e Buffer
- Il processo Buffer può contenere fino ad N messaggi.
- Utilizzo delle primitive:

  Send (*mess, proc*); /*send sincrona*/

  Proc= **Receive** (*mess*); /* receive bloccante*/
- Il processo produttore invia due tipi di messaggi, *pronto (*per tenere conto della limitazione di *buffer)* e *dati*.Il processo consumatore invia un messaggio *pronto.*
- Il processo *buffer* ha *due canali*, uno per il tipo di messaggio *pronto* ed uno per il tipo di messaggio *dati.*Essendo la **send** di tipo *sincrono* non è necessario, rispetto al caso di **send** *asincrona*, il messaggio ***ok-to-send***.
- I canali servono al processo *buffer* per sincronizzarsi con i processi produttore e consumatore.

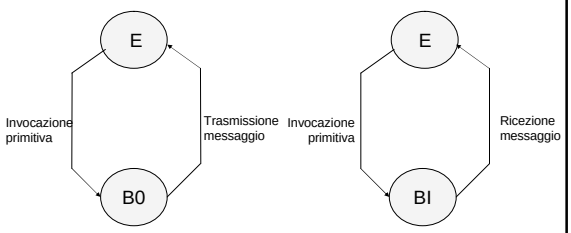



```
void produttore()
{ T dati;   messaggio pronto=...;

  while (1)
  {   <produci dati>;
      Send(pronto,buffer);
      Send(dati, buffer);
  }
}

void consumatore()
{ T dati; messaggio pronto=...;
  processo proc;

  while (1)
  {   Send(pronto,buffer);
      proc=Receive(dati)
      <consuma dati>;
  }
 }
```

```
void buffer_process()
{ queue coda; /*coda di elementi di tipo T*/,
  processo proc;
  boolean cons-pronto, prod-pronto=false;
  T dati;
  messaggio pronto;
  while (1)
  {   proc=Receive(pronto);
      if (proc==produttore)
             if (<coda piena>)
                  prod-pronto=true;
             else
             {    proc=Receive (dati);
                  if (cons-pronto)
                  {     Send(dati,consumatore);
                        cons-pronto=false;
                  }
                  else <inserzione dati in coda>
             }
  /* continua...*/
```

```
    else /* il mittente (proc) e` un consumatore*/
        if (<coda vuota>)
              cons-pronto=true;
        else
        {     <estrazione dati da coda>;
              Send (dati, consumatore);
              if (prod-pronto)
              {     proc=Receive(dati);
                    <inserzione dati in coda>;
                     prod-pronto=false;
              }
         }
    }/* fine while*/
}
```

Stati di un processo che esegue la send e la receive:



E (esecuzione)
BO (bloccato, ma pronto ad inviare un messaggio)

E (esecuzione)
BI (bloccato, ma pronto a ricevere un messaggio)

Implementazione di send/receive

Il *supporto a tempo di esecuzione* ha il compito di implementare i *messaggi di sincronizzazione* necessari per garantire il comportamento sincrono delle primitive.

- La **Send sincrona** viene tradotta nella seguente sequenza di operazioni asincrone:
  1. invio di un segnale di disponibilità a spedire un messaggio;
  2. attesa del segnale di disponibilità (*ok-to-send*) del ricevente (*sincronizzazione*);
  3. invio del messaggio vero e proprio.

Implementazione send sincrona

Esempio:

```
void Send_sincrona(mess msg, proc Dest)
{ mess OK; proc D;
   send(Dest,ready_to_send); /* send asincrona*/
   while (D=receive(&OK)!=Dest)|| OK!= ok_to_send);
   send(Dest, msg);
}

proc Receive(mess *Msg)
{ proc p;
   p=receive(Msg);
   send(proc, ok_to_send);
   p=receive(Msg);
   return p;
}
```