

```

// *****
// MODELLO A SCAMBIO DI MESSAGGI
// *****
// Dichiarazione di un canale per messaggi di tipo intero
//   PORT: Canale 1-N (dichiarato sul ricevente)
port int channel;

// Riferimento al canale da processi esterni
<< nome del processo >>.channel;

// Primitiva send
send << valore >> to << porta >>;

// Primitiva receive
//   - Bloccante se non ci sono messaggi in coda
//   - Restituisce il nome del processo mittente
receive << variabile >> from << porta >>;

// Comando con guardia
//   << guardia >> = << espressione booleana >> << primitiva di ricezione >>
[] << guardia >> -> << istruzione >>;

// Struttura a comando alternativo
if
  [] << guardia1 >> -> << istruzione1 >>;
  //...
  [] << guardiaN >> -> << istruzioneN >>;
fi

// Struttura a comando ripetitivo
do
  [] << guardia1 >> -> << istruzione1 >>;
  //...
  [] << guardiaN >> -> << istruzioneN >>;
od

// Esempio di servitore
process server
{
  port int servizioA;
  port real servizioB;
  Risorsa r;
  int x;
  real y;

  do
    [] (condA); receive(x) from servizioA; ->
    {
      r.A(x);
      // Eventuale restituzione al cliente
    }
    [] (condB); receive(y) from servizioB; ->
    {
      r.B(y);
      // Eventuale restituzione al cliente
    }
  od
}

```

```

// *****
// SEMAFORI NEL MODELLO A SCAMBIO DI MESSAGGI
// *****
// Semaforo
process semaphore
{
    port signal P;
    port signal V;
    int valore = I;
    process proc;
    signal s;
    do
        [] (valore > 0); proc = receive(s) from P; ->
            valore--;
            send(s) to proc.risposta;
        [] proc = receive(s) from V; ->
            valore++;
    od
}

// Chiamata
signal s;
port signal risposta;
send(s) to semaphore.P;
receive(s) from risposta;
send(s) to semaphore.V;

// *****
// REALIZZAZIONE PRIMITIVE ASINCRONE
// *****
// Hp: funzioni per gestire una coda realizzate
void inserisci(messaggio* m, codaMessaggi cm);
messaggio* estrai(codaMessaggi cm);
boolean codaVuota(codaMessaggi cm);

// Descrittore porta di tipo messaggio
typedef struct
{
    codaMessaggi coda;
    desPorta* puntatore;
} desPorta;

typedef desPorta* pPorta;

typedef struct
{
    // ...
    pPorta porteProcesso[NUM_MAX_PORTE];
} desProcesso;

// Bloccaggio su una porta e verifica
void BloccaSu(int iP); // Blocca processoInEsecuzione sulla porta di indice iP (cambia
solo lo stato)
boolean BloccatoSu(desProcesso proc, int iP);

// Verifica la presenza di messaggi o si sospende
void TestaPorta(int iP)

```

```

{
    desProcesso attuale = processoInEsecuzione;
    pPorta p = attuale->porteProcesso[iP];
    if (codaVuota(p->coda))
    {
        bloccaSu(iP);
        AssegnazioneCPU();
    }
}

messaggio* EstraiDaPorta(int iP)
{
    messaggio* m;
    desProcesso attuale = processoInEsecuzione;
    pPorta p = attuale->porteProcesso[iP];
    m = estrai(p->coda);
    return m;
}

void InserisciPorta(messaggio* m, PID proc, int iP)
{
    desProcesso dest = Descrittore(proc);
    pPorta p = dest->porteProcesso[iP];
    inserisci(m, p->coda);
    if (bloccatoSu(dest, iP))
        attiva(dest);
}

void send(T inf, PID proc, int iP)
{
    messaggio* m = new messaggio;
    m->info = inf;
    m->mitt = processoInEsecuzione;
    InserisciPorta(m, proc, iP);
}

void receive(T* inf, PID* proc, int iP)
{
    messaggio* m;
    TestaPorta(iP);
    m = EstraiDaPorta(iP);
    proc = &(m->mittente);
    inf = &(m->info);
}

// Realizzazione messaggi con guardia, verifica su più porte e ricezione qualsiasi
int TestaPorte(int iP[], int n)
{
    pPorta p;
    int ris = -1;
    int indicePorta;
    desProcesso attuale = processoInEsecuzione;
    for (int i = 0; i < n; ++i)
    {
        indicePorta = iP[i];
        p = attuale->porteProcesso[indicePorta];
        if (CodaVuota(p->coda))
            BloccaSu(indicePorta);
    }
}

```

```

else
{
    ris = indicePorta;
    attuale->stato = << processo attivo >>;
    break;
}
}
if (ris == -1)
    AssegnazioneCPU();
return ris;
}

int ReceiveAny(T* inf, PID* proc, int iP[], int n)
{
    messaggio* m;
    int indicePorta;
    do {
        indicePorta = TestaPorte(iP, n);
    } while (indicePorta == -1);
    m = EstraiDaPorta(indicePorta);
    proc = &(m->mittente);
    inf = &(m->info);
    return indicePorta;
}

// *****
// REALIZZAZIONE PRIMITIVE SINCRONE TRAMITE SEMAFORI
// *****

semaphore M = 0;
semaphore R = 0;
messaggio buffer;

void send(T dato)
{
    messaggio mess;
    mess.info = dato;
    mess.mittente = processoInEsecuzione;
    buffer = mess;
    V(R);
    P(M);
}

void receive(T* dato, PID* mitt)
{
    messaggio mess;
    P(R);
    mess = buffer;
    V(M);
    *dato = mess->info;
    *mitt = mess->mittente;
}

// *****
// REALIZZAZIONE PRIMITIVE SINCRONE TRAMITE PRIMITIVE ASINCRONE
// *****

void send(T inf, PID proc, int iP)
{
    signal s;

```

```

a_send(inf, proc, iP);
a_reveice(s, proc, akPort);
}

void receive(T* inf, PID* proc, int iP)
{
    signal s;
    a_receive(inf, proc, iP);
    a_send(s, proc, akPort);
}

// *****
// REALIZZAZIONE PRIMITIVE SINCRONE DA NUCLEO
// *****
// Verifica la presenza di messaggi o si sospende
void AttendiRicezione()
{
    desProcesso attuale = processoInEsecuzione;
    attuale->stato = << bloccato sulla send >>;
    AssegnazioneCPU();
}

void send(T inf, PID proc, int iP)
{
    messaggio* m = new messaggio;
    m->info = inf;
    m->mitt = processoInEsecuzione;
    InserisciPorta(m, proc, iP);
    AttendiRicezione();
}

void receive(T* inf, PID* proc, int iP)
{
    messaggio* m;
    TestaPorta(iP);
    m = EstraiDaPorta(iP);
    proc = &(m->mittente);
    inf = &(m->info);
}

// *****
// COMUNICAZIONE CON SINCRONIZZAZIONE ESTESA
// *****
accept << servizio >> (in << parametri ingresso >>, out << parametri uscita >>) -> istruzioni;

// *****
// COMUNICAZIONE CON LINGUAGGIO ADA
// *****
task name is
    << dichiarazione entry >>;
end;

task body name is
    << dichiarazioni locali >>;
begin
    << istruzioni task >>;
end;

```

```

entry entryname (<< parametri formali >>);

// Chiamata
call P.entryname (<< parametri effettivi >>);

// Accettazione
accept entrynae (in << ingressi >>, out << uscite >>);
do
    S1;
end;

// Comando con guardia - forma semplice
select
    accept entry1 do
        S1;
    end;

    or

    accept entryN do
        SN;
    end;
end select;

// Comando con guardia - forma completa
select
    when cond1 -> accept entry1 do
        S1;
    end;

    or

    when condN -> accept entryN do
        SN;
    end;
end select;

// Chiamate alternative
select
    << chiamata entry >>;
    else << comandi >>;
end select;

// Chiamate ritardate
select
    << chiamata entry >>;
    or
    delay << intervallo di tempo >>;
end select;

```