# ADPmttb 2.0 *seconda parte*

Sul numero scorso, per i soliti problemi di spazio (tiranno), abbiamo pubblicato solo una versione ridotta dell'ADPmttb. Approfittando del fatto che ancora non pubblichiamo i listati di ADPnetwork, sfruttiamo un po' di spazio di questo articolo per «rilasciarvi» altre 5 routine dell'ADPmttb. Si tratta di funzioni per controllare lo stato delle porte create con la funzione NewPort mostrata lo scorso mese. Le prime due, PortWait (da non confondere con la WaitPort di Exec!!!) e MessageWait, permettono di sospendere il processo chiamante per un numero di secondi indicato o fino a quando non viene creata da qualche altro processo una porta, o arriva un messaggio atteso. In tutt'e due i casi il primo parametro è il nome della porta mttb e il secondo parametro è il tempo massimo da aspettare espresso in secondi. Indicando un tempo nullo il timeout è infinito (usare con cautela...).

La funzione CheckPort permette di stabilire se una determinata porta esiste e in caso affermativo quanti messaggi vi sono accodati. L'unico parametro da passare è il nome della porta.

La funzione WipePort serve invece per pulire una porta mttb da messaggi non più desiderati. Il primo parametro è, come al solito, la porta su cui agire, il secondo parametro indica la quantità di messaggi da togliere. Indicando 0 in questo campo la porta è ripulita da tutti i messaggi presenti. In risposta abbiamo il numero di messaggi tolti (o errore se qualcosa non è andato per il giusto verso).

Per finire, la funzione MessageLen permette di conoscere la dimensione del messaggio presente su una porta mttb. Ciò è molto utile quando non si conosce la quantità di memoria da riservare per contenere il messaggio arrivato. Il primo parametro è il modo di funzionamento (MODE_WAIT o MODE_NOWAIT), il secondo parametro è la porta. Indicando MODE_WAIT se la porta è vuota il processo chiamante è sospeso fino a quando non arriva il messaggio; indicando MODE_NOWAIT, nel caso la porta sia vuota, la funzione ritorna la costante predefinita EMPTY_PORT.

Sul prossimo numero cercheremo di mettere le rimanenti funzioni offerte dall'ADPmttb. Arrivederci...

```
/*************************************
*                                    *
*                                    *
*         A D P m t t b   2.0        *
*                                    *
*        MultiTasking  ToolBox       *
*           (seconda parte)          *
*                                    *
*        ----------------------      *
*        (c) 1989  ADPsoftware       *
*        ----------------------      *
*                                    *
*************************************/


int  PortWait(char *,int);
int  MessageWait(char *,int);
int  CheckPort(char *);
int  WipePort(char *, int);
int  MessageLen(UBYTE,char *);

extern struct ExecBase *SysBase;

/*************************************
*                                    *
*       P O R T   W A I T            *
*                                    *
*************************************/

PortWait(porta,sec)
char *porta;

{
if (sec == 0) sec--;
while ((struct MsgPort *)FindPort(porta) == NULL && sec)
      {
      Delay(SysBase->VBlankFrequency);
      sec--;
      }
return(iabs(sec));
}

/*************************************
*                                    *
*   M E S S A G E W A I T            *
*                                    *
*************************************/

MessageWait(porta,sec)
char *porta;

{

struct MsgPort *port;

if (sec == 0) sec--;
Forbid();
   if ((port = (struct MsgPort *)FindPort(porta)) == 0)
      {
      Permit();
      return(NO_PORT);
      }
Permit();

while (port->mp_MsgList.lh_Head->ln_Succ == NULL && sec)
      {
      Delay(SysBase->VBlankFrequency);
      sec--;
      }

return(iabs(sec));
}

/*************************************
*                                    *
*     C H E C K P O R T              *
*                                    *
*************************************/

CheckPort(porta)
char *porta;

{

int count=0;
struct MsgPort *port;
struct Node    *node;

Forbid();
   if ((port = (struct MsgPort *)FindPort(porta)) == 0)
      {
      Permit();
      return(NO_PORT);
      }
   for (node = port->mp_MsgList.lh_Head;
        node->ln_Succ;
        node = node->ln_Succ) count++;
Permit();
if (count) return(count);
else return(EMPTY_PORT);

}

/*************************************
*                                    *
*       W I P E   P O R T            *
*                                    *
*************************************/

WipePort(porta,nmsg)
char *porta;

{
struct adp_message *adpmsg;
struct MsgPort *port;
int i=0;

if (nmsg == 0) nmsg--;
Forbid();
   if ((port = (struct MsgPort *)FindPort(porta)) == 0)
      {
      Permit();
      return(NO_PORT);
      }
   while ((adpmsg = (struct adp_message *)GetMsg(port)) && nmsg--)
         {
         i++;
         if (adpmsg->mode & MODE_SYNC) ReplyMsg(adpmsg);
         else FreeMem( adpmsg, sizeof(struct adp_message) + adpmsg->len );
         }
Permit();
return(i);
}

/*************************************
*                                    *
*    M E S S A G E   L E N           *
*                                    *
*************************************/

MessageLen(mode,porta)
UBYTE mode;
char *porta;

{
struct MsgPort *port;
struct adp_message *temp;
ULONG  succ;
if ((mode & MODE_WAIT || mode & MODE_NOWAIT)==0) return(OP_FAIL);
Forbid();
   if ((port = (struct MsgPort *)FindPort(porta)) == 0)
      {
      Permit();
      return(NO_PORT);
      }
Permit();
if (mode & MODE_WAIT) WaitPort(port);
temp = (struct adp_message *)(port->mp_MsgList.lh_Head);
succ = (ULONG)(((struct Node *)temp)->ln_Succ);
if (succ) return(temp->len);
else return(EMPTY_PORT);
}
```