```
;/*  Pooltime.c
lc -b1 -cfist -v -y -j73 pooltime
blink FROM LIB:c.o pooltime.o TO pooltime LIB LIB:lc.lib LIB:amiga.lib
quit

This program demonstrates the use of a memory pool to allocate memory
for two timerequest structures.  Before pools, the memory required for
the timerequests was allocated using AllocMem().
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <devices/timer.h>

#include <clib/exec_protos.h>
#include <clib/alib_protos.h>

#include <stdio.h>

#ifdef LATTICE
int CXBRK(void) { return(0); }     /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */
#endif

VOID main(VOID);

void main()
{
struct timerequest *TimerIO[3];
struct MsgPort *TimerMP;
struct Message *TimerMSG;
APTR *TimerPool;            /* pointer to memory pool */
ULONG x,seconds[3]={4,1,2}, microseconds[3]={0,0,0};
int allin = 3;
char *position[]={"last","second","first"};

if (TimerMP = CreateMsgPort())
    {
    if (TimerIO[0] = CreateIORequest(TimerMP,sizeof(struct timerequest)) )
        {
        if (!OpenDevice(TIMERNAME, UNIT_VBLANK,(struct IORequest *) TimerIO[0], 0L))
            {
            /* Set command to TR_ADDREQUEST */
            TimerIO[0]->tr_node.io_Command = TR_ADDREQUEST;

                /* Create the memory pool */
            if (TimerPool = CreatePool(MEMF_FAST,400,200))
                {
                 /* Allocate pooled memory from TimerPool */
                if (TimerIO[1]=(struct timerequest *)
                     AllocPooled(TimerPool,sizeof(struct timerequest)))
                    {
                    /* Allocate pooled memory from TimerPool */
                    if (TimerIO[2]=(struct timerequest *)
                         AllocPooled(TimerPool,sizeof(struct timerequest)))
                        {
                        /* Copy fields from request used to open the timer device */
                        *TimerIO[1] = *TimerIO[0];
                        *TimerIO[2] = *TimerIO[0];

                        /* Initialize other fields */
                        for (x=0;x<3;x++)
                            {
                            TimerIO[x]->tr_time.tv_secs   = seconds[x];
                            TimerIO[x]->tr_time.tv_micro  = microseconds[x];
                            }

                        printf("\n\nSending multiple requests\n\n");

                        /* Send multiple requests asynchronously */
                        SendIO(TimerIO[0]);
                        SendIO(TimerIO[1]);
                        SendIO(TimerIO[2]);

                        /* Now go to sleep with WaitPort() waiting for the requests */
                        while (allin)
                              {
                              WaitPort(TimerMP);
                              TimerMSG=GetMsg(TimerMP);  /* Get the reply message */
                              for (x=0;x<3;x++)
                                if (TimerMSG==(struct Message *)TimerIO[x])
                                    printf("Request %ld finished %s\n",
                                           x,position[--allin]);
                              }
                        }
                    else
                        printf("Error: Could not allocate memory for TimerIO[2]\n");
                    }
                else
                    printf("Error: Could not allocate memory for TimerIO[1]\n");


                /* Delete the entire pool.  You could also free each individual
                 * allocation using FreePooled(), but this is quicker because a pool's
                 * puddles are automatically drained when you delete the pool.
                 */
                DeletePool(TimerPool);
                }
            else
                printf("Error: Could not allocate memory pool\n");

            CloseDevice(TimerIO[0]);
            }
        else
            printf("Error: Could not open %s\n", TIMERNAME);

        DeleteIORequest(TimerIO[0]);
        }
    else
        printf("Error: could not create IORequest\n");

    DeleteMsgPort(TimerMP);
    }
else
    printf("Error: Could not create message port\n");
}
```