

```

/* Execute me to compile with SASC 5.10a
lc -b1 -cflst -d0 -O -v -j73 ispy.c
asm -iINCLUDE: ispy_stubs.asm
blink from lib:c.o ispy.o ispy_stubs.o LIB lib:amiga.lib lib:lcnb.lib lib:debug.lib SC
SD ND DEFINE __main=__tinymain
quit
**
**      ISpy. AmigaMail SetFunction() example.
**
**      Copyright (c) 1991 Commodore-Amiga, Inc.
**      All Rights Reserved
**
*/

#include <exec/types.h>
#include <exec/execbase.h>
#include <exec/memory.h>
#include <exec/semaphores.h>
#include <dos/dos.h>
#include <libraries/gadtools.h>
#include <string.h>
#include <dos.h>

#include <clib/dos_protos.h>
#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>

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

#define ASM __asm __saveds
#define REG(x) register __## x

#ifdef PARALLEL
#define zprintf dprintf
extern VOID dprintf(STRPTR,...);
#else
#define zprintf kprintf
extern VOID kprintf(STRPTR,...);
#endif

#ifdef DEBUG
#define D(x) x
#else
#define D(x) ;
#endif

/* Local protos */
VOID main(VOID);
BOOL InstallWedge(VOID);
BOOL RemoveWedge(VOID);
struct JumpTable *GetJumpTable(UBYTE *);

/* Assembler stubs will return a pointer to the caller's stack in a4.
 * The only system function at this moment using a4 is the workbench.library
 * AddAppIconA() function.
 */
#define ACALLER (0) /* StackPtr[0] = ACaller, StackPtr[1] = saved A6 in stub */
#define CCALLER (2) /* StackPtr[2] = CCaller */

/* The number of 'replacement' functions */
#define NUMBEROFFUNCTIONS (4)

/* prototypes for the functions to be SetFunction()'ed. */

/* intuition.library */
struct Screen *(*ASM oldOpenScreen) (REG(a0) struct NewScreen *,
                                     REG(a6) struct Library *);
struct Window *(*ASM oldOpenWindowTagList) (REG(a0) struct NewWindow *,
                                             REG(a1) struct TagItem *,
                                             REG(a6) struct Library *);
struct Screen *ASM newOpenScreen(REG(a0) struct NewScreen *,
                                 REG(a4) ULONG *,
                                 REG(a6) struct Library *);
struct Window *ASM newOpenWindowTagList(REG(a0) struct NewWindow *,

```

```

                                     REG(a1) struct TagItem *,
                                     REG(a4) ULONG *,
                                     REG(a6) struct Library *);

/* exec.library */
VOID(*ASM oldFreeMem) (REG(a1) VOID *,
                      REG(d0) ULONG,
                      REG(a6) struct Library *);
VOID ASM newFreeMem(REG(a1) VOID *,
                   REG(d0) ULONG,
                   REG(a4) ULONG *,
                   REG(a6) struct Library *);

/* graphics.library */
VOID(*ASM oldSetFont) (REG(a1) struct RastPort *,
                      REG(a0) struct TextFont *,
                      REG(a6) struct Library *);
VOID ASM newSetFont(REG(a1) struct RastPort *,
                   REG(a0) struct TextFont *,
                   REG(a4) ULONG *,
                   REG(a6) struct Library *);

/* Assembler Stubs */
extern OpenScreenStub();
extern OpenWindowTagListStub();
extern FreeMemStub();
extern SetFontStub();

/* The LVO's to use from amiga.lib */
extern LVOOpenScreen;
extern LVOOpenWindowTagList;
extern LVOfreeMem;
extern LVOSetFont;

extern struct ExecBase *SysBase;
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

/* Use a table and an array to make it a little more generic and easier to
 * add functions.
 */
struct LVOTable
{
    LONG lt_LVO;
    struct Library *lt_LibBase;
    ULONG lt_oldFunction;
    ULONG lt_newFunction;
};

struct LVOTable LVOArray[] =
{
    {&LVOOpenScreen, (struct Library *) &IntuitionBase,
     &oldOpenScreen,
     &OpenScreenStub},
    {&LVOOpenWindowTagList, (struct Library *) &IntuitionBase,
     &oldOpenWindowTagList, &OpenWindowTagListStub},
    {&LVOfreeMem, (struct Library *) &SysBase, &oldFreeMem, &FreeMemStub},
    {&LVOSetFont, (struct Library *) &GfxBase, &oldSetFont, &SetFontStub},
};

struct JumpTable
{
    struct SignalSemaphore jt_Semaphore;
    UWORD pad word;
    struct Task *jt_Owner;
    UBYTE jt_Function[NUMBEROFFUNCTIONS * 6];
};

/* Strings */
/* The name this JumpTable/Semaphore will get. */
static UBYTE *JTName = "ISpy-MiscJumpTable";

static UBYTE *VersTag = "\0$VER: ISpy 37.4 (21.3.91)";
static UBYTE *VerTitle = "ISpy 37.4";
static UBYTE *Copyright = "Copyright (c) 1991 Commodore-Amiga, Inc.\n";
static UBYTE *CBreak = "CTRL-C or BREAK to exit...\n";
VOID
main(VOID)

```

```
{
    Write(Output(), (STRPTR) VerTitle, strlen((STRPTR) VerTitle));
    Write(Output(), Copyright, strlen(Copyright));

    if (SysBase->LibNode.lib_Version > 36)
    {
        if (IntuitionBase = OpenLibrary("intuition.library", 37))
        {
            if (GfxBase = OpenLibrary("graphics.library", 37))
            {
                if (InstallWedge())
                {
                    Write(Output(), CBreak, strlen(CBreak));
                    Wait(SIGBREAKF_CTRL_C);
                    RemoveWedge();
                }
                CloseLibrary(GfxBase);
            }
            else
                Write(Output(), "Couldn't open graphics.library\n", 31);
            CloseLibrary(IntuitionBase);
        }
        else
            Write(Output(), "Couldn't open intuition.library\n", 32);
    }
    else
        Write(Output(), "Requires at least Kickstart 2.04\n", 33);
}

BOOL
InstallWedge(VOID)
{
    struct JumpTable *jumptable;
    ULONG *addressptr;
    UCOUNT i, j;

    Forbid();

    /* Get pointer to JumpTable. Create it if necessary */
    if (jumptable = GetJumpTable(JTName))
    {
        /* Try to get exclusive lock on semaphore, in case it already existed. */
        if (AttemptSemaphore((struct SignalSemaphore *) jumptable))
        {
            /* Make sure nobody else has function addresses in the jumtable */
            if (jumptable->jt_Owner == NULL)
            {
                jumptable->jt_Owner = FindTask(0);
                /* Don't want to disable any longer than necessary */
                Disable();

                for (i = 2, j = 0; i < NUMBEROFFUNCTIONS * 6; i += 6, j++)
                {
                    /* Replace addresses in the jumtable with my own. */
                    addressptr = (ULONG *) ((UBYTE *) jumptable->jt_Function + i);
                    *((ULONG *) LVOArray[j].lt_oldFunction) = (ULONG) * addressptr;
                    *addressptr = (ULONG) LVOArray[j].lt_newFunction;
                    D(zprintf("setting table to Function: 0x%lx\n", *addressptr));
                }
                Enable();
            }
            else
                Write(Output(), "Already running.\n", 16);
            ReleaseSemaphore((struct SignalSemaphore *) jumptable);
        }
        else
            Write(Output(), "Can't lock table.\n", 18);
    }
    else
        Write(Output(), "Can't create jumtable\n", 23);
    Permit();
    return ((BOOL) jumptable);
}

BOOL
RemoveWedge(VOID)
```

```
struct JumpTable *jumptable;
ULONG *addressptr;
UCOUNT i, j;

Forbid();

if (jumptable = GetJumpTable(JTName))
{
    D(zprintf("jumtable @ 0x%lx\n", jumptable));

    /* Check if this task owns this jumtable */
    if (jumptable->jt_Owner == FindTask(0))
    {
        /* Get the semaphore exclusively.
        * Depending on what got SetFunction()'ed this could take some time.
        * Also note that shared locks are used to indicate the code is
        * being executed and that shared locks can jump ahead of queue'ed
        * exclusive locks, adding to the waittime.
        */
        ObtainSemaphore((struct SignalSemaphore *) jumptable);

        Disable();

        /* Restore old pointers in jumtable */
        for (i = 2, j = 0; i < NUMBEROFFUNCTIONS * 6; i += 6, j++)
        {
            addressptr = (ULONG *) ((UBYTE *) jumptable->jt_Function + i);
            *addressptr = *((ULONG *) LVOArray[j].lt_oldFunction);
            D(zprintf("setting table to oldFunction: 0x%lx\n", *addressptr));
        }

        Enable();

        jumptable->jt_Owner = NULL;
        ReleaseSemaphore((struct SignalSemaphore *) jumptable);
    }
}

Permit();

return (TRUE);
}

struct JumpTable *
GetJumpTable(UBYTE * name)
{
    struct JumpTable *jumptable;
    ULONG *addressptr;
    UWORD *jmpinstr;
    UBYTE *jtname;
    UCOUNT i, j;

    /* Not really necessary to forbid again, just to indicate that I don't
    * want another task to create the semaphore while I'm trying to do the
    * same. Here GetJumpTable() is only called from InstallWedge(), so it
    * will just bump the forbid count.
    */
    Forbid();

    if (!(jumptable = (struct JumpTable *) FindSemaphore(name)))
    {
        if (jumptable = AllocMem(sizeof(struct JumpTable), MEMF_PUBLIC | MEMF_CLEAR))
        {
            if (jtname = AllocMem(32, MEMF_PUBLIC | MEMF_CLEAR))
            {
                for (i = 0, j = 0; i < NUMBEROFFUNCTIONS * 6; i += 6, j++)
                {
                    jmpinstr = (UWORD *) ((UBYTE *) jumptable->jt_Function + i);
                    *jmpinstr = 0x4EF9;

                    addressptr = (ULONG *) ((UBYTE *) jumptable->jt_Function + i + 2);
                    *addressptr = (ULONG) SetFunction(
                        (struct Library *) *((ULONG *) LVOArray[j].lt_LibBase)),
                        LVOArray[j].lt_LVO,
```

```
(VOID *) ((UBYTE *) jumptable->jt_Function + i));
    }

    jumptable->jt_Semaphore.ss_Link.ln_Pri = 0;

    strcpy(jtname, name);
    jumptable->jt_Semaphore.ss_Link.ln_Name = jtname;
    AddSemaphore((struct SignalSemaphore *) jumptable);
    /* In 1.2/1.3 AddSemaphore() didn't work properly.
    ** Under 1.2/1.3, change it to:
    ** InitSemaphore(jumptable);
    ** Forbid();
    ** Enqueue(&SysBase->SemaphoreList, jumptable);
    ** Permit();
    */
    }
    else
    {
        FreeMem(jumptable, sizeof(struct JumpTable));
        jumptable = NULL;
    }
}

Permit();

/* Note: if you'd want this to work with 1.3, you wouldn't/couldn't lock the
* semaphore, but instead would have to use a global to in- and decrement.
* When exiting, you'd spin around the global counter, waiting for it to reach
* zero. At that point you'd Disable(), reset the pointers in the jumtable and
* Enable() again.
*/

struct Screen *ASM
newOpenScreen(REG(a0) struct NewScreen * newscreen,
              REG(a4) ULONG * stackptr,
              REG(a6) struct Library * base)
{
    struct SignalSemaphore *jt;
    struct Screen *screen = NULL;

    /* Find the semaphore to lock shared, indicating the routine is being run. */
    /* For speed reasons you may want to cache the pointer to the semaphore
    ** in a global variable */
    if (jt = FindSemaphore(JTName))
    {
        /* Lock shared in 2.0. In 1.3 you'd increment a global counter */
        ObtainSemaphoreShared(jt);
        /* Simple test for valid argument. Could check all the fields too. */
        if (newscreen != NULL)
        {
            screen = (*oldOpenScreen) (newscreen, base);
        }
        else
        {
            ULONG ACaller = stackptr[ACALLER];
            ULONG CCaller = stackptr[CCALLER];

            Forbid();
            /* To make sure the output isn't garbled */
            zprintf("OpenScreen(NULL) by '%s' (at 0x%lx) from A:0x%lx C:0x%lx\n",
                SysBase->ThisTask->tc_Node.ln_Name,
                SysBase->ThisTask,
                ACaller,
                CCaller,
                stackptr);
            Permit();
        }
    }
}
```

```
/* Release shared lock. In 1.3 you'd decrement the global pointer */
ReleaseSemaphore(jt);
}

return (screen);
}

struct Window *ASM
newOpenWindowTagList(REG(a0) struct NewWindow * newwindow,
                    REG(a1) struct TagItem * tags,
                    REG(a4) ULONG * stackptr,
                    REG(a6) struct Library * base)
{
    struct SignalSemaphore *jt;
    struct Window *window = NULL;

    if (jt = FindSemaphore(JTName))
    {
        ObtainSemaphoreShared(jt);
        if (newwindow != NULL || tags != NULL)
        {
            window = (*oldOpenWindowTagList) (newwindow, tags, base);
        }
        else
        {
            ULONG ACaller = stackptr[ACALLER];
            ULONG CCaller = stackptr[CCALLER];

            Forbid();
            zprintf("OpenWindowTagList(NULL,NULL) by '%s' (at 0x%lx) from A:0x%lx C:0x%lx\n",
                SysBase->ThisTask->tc_Node.ln_Name,
                SysBase->ThisTask,
                ACaller,
                CCaller,
                stackptr);
            Permit();
        }
        ReleaseSemaphore(jt);
    }

    return (window);
}

VOID ASM
newFreeMem(REG(a1) VOID * memptr,
           REG(d0) ULONG size,
           REG(a4) ULONG * stackptr,
           REG(a6) struct Library * base)
{
    struct SignalSemaphore *jt;

    if (jt = FindSemaphore(JTName))
    {
        ObtainSemaphoreShared(jt);
        if (memptr != NULL && size != 0L)
        {
            (*oldFreeMem) (memptr, size, base);
        }
        else
        {
            ULONG ACaller = stackptr[ACALLER];
            ULONG CCaller = stackptr[CCALLER];

            Forbid();
            zprintf("FreeMem(0x%lx,%ld) by '%s' (at 0x%lx) from A:0x%lx C:0x%lx\n",
                memptr, size,
                SysBase->ThisTask->tc_Node.ln_Name,
                SysBase->ThisTask,
                ACaller,
                CCaller,
                stackptr);
            Permit();
        }
    }
}
```

```

    }
    ReleaseSemaphore(jt);
}

VOID ASM
newSetFont(REG(a1) struct RastPort * rp,
           REG(a0) struct TextFont * font,
           REG(a4) ULONG * stackptr,
           REG(a6) struct Library * base)
{
    struct SignalSemaphore *jt;

    if (jt = FindSemaphore(JTName))
    {
        ObtainSemaphoreShared(jt);
        if (rp != NULL && font != NULL)
        {
            (*oldSetFont) (rp, font, base);
        }
        else
        {
            ULONG          ACaller = stackptr[ACALLER];
            ULONG          CCaller = stackptr[CCALLER];

            Forbid();
            zprintf("SetFont(0x%lx,0x%lx) by '%s' (at 0x%lx) from A:0x%lx C:0x%lx\n",
SP:0x%lx\n",
                    rp, font,
                    SysBase->ThisTask->tc_Node.ln_Name,
                    SysBase->ThisTask,
                    ACaller,
                    CCaller,
                    stackptr);
            Permit();
        }
        ReleaseSemaphore(jt);
    }
}

```

```

**
**      ISpy stubs

```

```

**
**      Copyright (c) 1991 Commodore-Amiga, Inc.
**      All Rights Reserved
**
**
**      INCLUDE "exec/types.i"
**
**      SECTION CODE
**
**      XREF      _SysBase
**
**      XREF      _LVOForbid
**      XREF      _LVOPermit
**
**      XREF      _newOpenScreen
**      XREF      _newOpenWindowTagList
**      XREF      _newFreeMem
**      XREF      _newSetFont
**
**      XDEF      _OpenScreenStub
**      XDEF      _OpenWindowTagListStub
**      XDEF      _FreeMemStub
**      XDEF      _SetFontStub
**
**      _OpenScreenStub:
**      movem.l    a4,-(sp)
**      lea        4(sp),a4
**      jsr        _newOpenScreen
**      movem.l    (sp)+,a4
**      rts
**
**      _OpenWindowTagListStub:
**      movem.l    a4,-(sp)
**      lea        4(sp),a4
**      jsr        _newOpenWindowTagList
**      movem.l    (sp)+,a4
**      rts
**
**      _FreeMemStub:
**      movem.l    a4,-(sp)
**      lea        4(sp),a4
**      jsr        _newFreeMem
**      movem.l    (sp)+,a4
**      rts
**
**      _SetFontStub:
**      movem.l    a4,-(sp)
**      lea        4(sp),a4
**      jsr        _newSetFont
**      movem.l    (sp)+,a4
**      rts
**
**      END

```

