```
;/* SIGF_Single.c - Execute me to compile me with SAS C 5.10b
LC -b1 -cfist -v -y -j73 SIGF_Single.c
Blink FROM LIB:c.o,SIGF_Single.o TO SIGF_Single LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit ;*/

/* This example program illustrates simple usage of the SIGF_SINGLE */
/* signal for "single shot" signalling.  This signal is one of the  */
/* system private signals, but applications can use it in certain   */
/* cases, but only if used carefully.  Specifically, applications   */
/* should use it only to Wait() on, and using only that signal      */
/* (applications cannot Wait() on other signals in the same         */
/* Wait()).  Not following these rules can cause serious system     */
/* problems. For more details, see the Amiga Mail article,          */
/* "Signalling with SIGF_SINGLE" from the September/October 1992    */
/* issue of Amiga Mail.                                             */

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/dosextens.h>
#include <dos/dostags.h>

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

static UBYTE *VersTag = "$VER: SIGF_Single 38.7 (21.9.92)";

void childprocesscode(void);      /* prototype for our childprocess routine. */

struct Process *mainprocess = NULL, *childprocess = NULL;
UBYTE childprocessname[] = "RKM_signal_childprocess";

BPTR output;

void main(int argc, char **argv)
{
    if (output = Open("*", MODE_OLDFILE))       /* Open the console for the  */
                                                /*            child process. */
    {
        mainprocess = (struct Process *)FindTask(NULL);/* childprocess can   */
                                                       /* access this global.*/

        if (childprocess = CreateNewProcTags(
                    NP_Entry,       childprocesscode,  /* The child process  */
                    NP_StackSize,   2000L,             /*              code. */
                    NP_Name,        childprocessname,
                    NP_Output,      output,
                    NP_FreeSeglist, FALSE,
                    NP_CloseOutput, TRUE,
                    TAG_DONE,       0L))

        {                                  /* I split the PutStr() into two */
            PutStr("Main Process: Created a child process"); /* to fit code  */
            PutStr(" and waiting on SIGF_SINGLE.\n");        /* on an 8.5 by */
                                                             /* 11 page.     */

            Flush(Output());    /* Make sure the PutStr() above appears      */
                                /* in the console window before the child    */
                                /* process start PutStr()ing to the console. */

            SetSignal(0L, SIGF_SINGLE);        /* Use SIGF_SINGLE only after */
                                               /* clearing it.               */
            Wait(SIGF_SINGLE);     /* Only use SIGF_SINGLE for Wait()ing and */
                                   /* Wait on that signal alone!             */

            PutStr("Main Process: Received signal from child.\n");
        }
        else
            PutStr("Main Process: Can’t create child process. Exiting.\n");
    }
    else
        PutStr("Main Process: Can’t open CON:.  Exiting.\n");
}

void childprocesscode(void)     /* This function is what CreateNewProcTags() */
{                               /* loads as the child process.  This child   */
                                /* signals the parent using SIGF_SINGLE.     */
    ULONG x;

    PutStr("Child Process: I’m alive and starting a 5 second TimeDelay()");
    Flush(Output());

    for (x = 0; x < 5; x++)
    {
        TimeDelay(UNIT_VBLANK, 1, 0);     /* Delay for 5 seconds, printing a */
        FPuts(Output(), " .");            /* dot during each second.         */
        Flush(Output());
    }

    TimeDelay(UNIT_VBLANK, 1, 0);

    PutStr(" Finished.\n");
    PutStr("Child Process: Signalling main process and exiting.  Bye.\n");
    Flush(Output());

    Signal((struct Task *)mainprocess, SIGF_SINGLE);    /* Finished waiting, */
                                                        /* signal the main   */
                                                        /* process and exit  */
                                                        /* child process.    */
}
```