

\$SPAD/src/lib fnct\_key.c

The Axiom Team

July 29, 2014

**Abstract**

## Contents

<b>1</b>	<b>MAC OSX and BSD port</b>	<b>3</b>
<b>2</b>	<b>License</b>	<b>3</b>

## 1 MAC OSX and BSD port

On the MAC OSX the signal `[[SIGCLD]]` has been renamed to `[[SIGCHLD]]`. In order to handle this change we need to ensure that the platform variable is set properly and that the platform variable is changed everywhere.

— mac os signal rename —

```
#if defined(MACOSXplatform) || defined(BSDplatform)
    bsdSignal(SIGCHLD, null_fnct, RestartSystemCalls);
#else
    bsdSignal(SIGCLD, null_fnct, RestartSystemCalls);
#endif
```

---

## 2 License

```
/*
Copyright (c) 1991-2002, The Numerical Algorithms Group Ltd.
All rights reserved.
```

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical Algorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

The MACOSX platform is broken because no matter what you do it seems to include files from `[[/usr/include/sys]]` ahead of `[[/usr/include]]`. On linux systems these files include themselves which causes an infinite regression of includes that fails. GCC gracefully steps over that problem but the build fails anyway. On MACOSX the `[[/usr/include/sys]]` versions of files are badly broken with respect to the `[[/usr/include]]` versions.

```

— * —

#ifdef MACOSXplatform
#include "/usr/include/unistd.h"
#else
#include <unistd.h>
#endif
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

#include "edible.h"
#include "bsdsignal.h"

#include "bsdsignal.h1"
#include "fnct-key.h1"
#include "prt.h1"
#include "edin.h1"

/** Some constants for function key defs ****/
#define DELAYED 0
#define IMMEDIATE 1
#define SPECIAL 2

/** Here is the structure for storing bound pf-keys          ***/
fkey function_key[13];          /** Strings which replace function
                                keys when a key is hit          ***/

static char *defaulteditor = "clefedit";
char editorfilename[100];

/*

```

```

* The following function environment variable clef editor. The command
* should be the one that the user wishes to have execed
*/

void
set_editor_key(void)
{
    int pid;

    sprintf(editorfilename, "/tmp/clef%d", pid = getpid());

    if (function_key[12].str == NULL) {
        (function_key[12]).type = SPECIAL;
        (function_key[12]).str = defaulteditor;
    }
}

void
define_function_keys(void)
/** This routine is used to find the users function key mappings. It
    simply searches the users HOME directory for a file called ".clef".
    If found it gets the key bindings from within
    *****/
{
    char *HOME, path[1024], string[1024];
    int key;
    int fd;
    char type;

    /** lets initialize the key pointers **/
    for (key = 0; key < 13; key++)
        (function_key[key]).str = NULL;
    /** see if the user has a .clef file      ***/
    HOME = getenv("HOME");
    sprintf(path, "%s/.clef", HOME);
    if ((fd = open(path, O_RDONLY)) == -1) {
        return;
    }
    else {
        /** If so, then get the key bindings **/
        while ((key = get_key(fd, &type))) {
            get_str(fd, string);
            switch (type) {
                case 'D':
                    if (key == 12) {
                        fprintf(stderr,
                            "Clef Error: PF12 can only be of type E in .clef\n");
                    }
                }
            }
        }
    }
}

```

```

        fprintf(stderr, "Line will be ignored\n");
        type = -1;
    }
    else {
        (function_key[key]).type = DELAYED;
    }
    break;
case 'F':
    if (key == 12) {
        fprintf(stderr,
            "Clef Error: PF12 can only be of type E in .clef\n");
        fprintf(stderr, "Line will be ignored\n");
        type = -1;
    }
    else {
        (function_key[key]).type = IMMEDIATE;
    }
    break;
case 'E':
    if (key != 12) {
        fprintf(stderr,
            "Clef Error: PF12 can only be of type E in .clef\n");
        fprintf(stderr, "Line will be ignored\n");
        type = -1;
    }
    else {
        (function_key[key]).type = SPECIAL;
    }
    break;
}
if (type != -1) {
    (function_key[key]).str =
        (char *) malloc(strlen(string) + 1);
    sprintf((function_key[key]).str, "%s", string);
}
}

/*
 * Now set the editor function key
 */
set_editor_key();
}

#define defof(c) ((c == 'F' || c == 'D' || c == 'E')?(1):(0))

int
get_key(int fd, char * ty)
{

```

```

/*
 * Determines the key number being mapped, and whether it is immediate or
 * delay. It returns the key value, and modifies the parameter type
 */
char keynum[1024];
int nr;

nr = read(fd, keynum, 3);
if (nr != -1 && nr != 0) {
    if (!defof(keynum[0])) {
        return 0;
    }
    else {
        *ty = keynum[0];
        keynum[3] = '\0';
        return (atoi(&keynum[1]));
    }
}
else
    return 0;
}

int
get_str(int fd, char * string)
{
    /** Gets the key mapping being bound **/
    char c;
    int count = 0;
    char *trace = string;

    read(fd, &c, 1);
    while (c == ' ')
        read(fd, &c, 1);
    while (c != '\n') {
        count++;
        *trace++ = c;
        if (read(fd, &c, 1) == 0)
            break;
    }
    *trace = '\0';
    return count;
}

void
null_fnct(int sig)
{
    return;
}

```

```

void
handle_function_key(int key,int  chann)
{
    /** this procedure simply adds the string specified by the function key
        to the buffer                                     *****/
    int count, fd;
    int amount = strlen(function_key[key].str);
    int id;
    int save_echo;

    /** This procedure takes the character at in_buff[num_proc] and adds
        it to the buffer. It first checks to see if we should be inserting
        or overwriting, and then does the appropriate thing      *****/

    switch ((function_key[key]).type) {
    case IMMEDIATE:
        if (INS_MODE) {
            forwardcopy(&buff[curr_pntr + amount],
                        &buff[curr_pntr],
                        buff_pntr - curr_pntr);
            forwardflag_cpy(&buff_flag[curr_pntr + amount],
                           &buff_flag[curr_pntr],
                           buff_pntr - curr_pntr);
            for (count = 0; count < amount; count++) {
                buff[curr_pntr + count] = (function_key[key].str)[count];
                buff_flag[curr_pntr + count] = '1';
            }
            ins_print(curr_pntr, amount + 1);
            buff_pntr = buff_pntr + amount;
        }
        else {
            for (count = 0; count < amount; count++) {
                buff[curr_pntr + count] = (function_key[key].str)[count];
                buff_flag[curr_pntr + count] = '1';
                myputchar((function_key[key].str)[count]);
            }
        }
        num_proc = num_proc + 6;
        curr_pntr = curr_pntr + amount;
        buff_pntr = buff_pntr + amount;
        send_function_to_child();
        break;
    case DELAYED:
        if (INS_MODE) {
            forwardcopy(&buff[curr_pntr + amount],
                        &buff[curr_pntr],
                        buff_pntr - curr_pntr);
            forwardflag_cpy(&buff_flag[curr_pntr + amount],
                           &buff_flag[curr_pntr],
                           buff_pntr - curr_pntr);

```



```

        for (count = 0; count < amount; count++) {
            buff[curr_pntr + count] = (function_key[key].str)[count];
            buff_flag[curr_pntr + count] = '1';
        }
        ins_print(curr_pntr, amount + 1);
        buff_pntr = buff_pntr + amount;
    }
    else {
        for (count = 0; count < amount; count++) {
            buff[curr_pntr + count] = (function_key[key].str)[count];
            buff_flag[curr_pntr + count] = '1';
            myputchar((function_key[key].str)[count]);
        }
    }
    num_proc = num_proc + 6;
    curr_pntr = curr_pntr + amount;
    buff_pntr = buff_pntr + amount;
    fflush(stdout);
    break;
case SPECIAL:
    /* fprintf(stderr, "Here I am \n"); */
    if (access(editorfilename, F_OK) < 0) {
        fd = open(editorfilename, O_RDWR | O_CREAT, 0666);
        write(fd, buff, buff_pntr);
        back_up(buff_pntr);
        close(fd);
    }
    else {
        if (buff_pntr > 0) {
            fd = open(editorfilename, O_RDWR | O_TRUNC);
            write(fd, buff, buff_pntr);
            back_up(buff_pntr);
            close(fd);
        }
    }
}
\getchunk{mac os signal rename}
switch (id = fork()) {
    case -1:
        perror("Special key");
        break;
    case 0:
        execlp((function_key[12]).str,
            (function_key[12]).str,
            editorfilename, NULL);
        perror("Returned from exec");
        exit(0);
}
while (wait((int *) 0) < 0);
/** now I should read that file and send all it stuff thru the

```

```

        reader                                     *****/
fd = open(editorfilename, O_RDWR);
if (fd == -1) {
    perror("Opening temp file");
    exit(-1);
}
num_proc += 6;

/** reinitialize the buffer */
init_flag(buff_flag, buff_pntr);
init_buff(buff, buff_pntr);
/** reinitialize my buffer pointers */
buff_pntr = curr_pntr = 0;
/** reset the ring pointer */
current = NULL;
save_echo = ECHOIT;
ECHOIT = 0;
while ((num_read = read(fd, in_buff, MAXLINE))) {
    do_reading();
}
close(fd);
break;
}
return;
}

```

---

## References

- [1] nothing