ResPublish

P. Baillehache

March 20, 2019

Contents

1 Definitions **2**
1.1 TextOMeter 2

2 Interface **3**

3 Code **4**
3.1 respublish.c 4
3.2 respublish-inline.c 6

4 Makefile **7**

5 Unit tests **8**

6 Unit tests output **9**

Introduction

ResPublish is a C library providing structures and functions to log and/or display data in various way during execution of a process.

- TextOMeter: displays textual data in a dedicated Xterm from a running parent process.

It uses the `PBErr` library.

1 Definitions

1.1 TextOMeter

The solution implemented in ResPublish to display text in a dedicated Xterm (while the parent process is attached to another terminal) is resumed below. It is very probably not portable to systems other than the one it has been developped on.

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
// Main function
// Example code to display text in an independant Xterm from a running
// parent process.
// Compile with gcc -o main main.c
int main() {
  // Temporary file to transmit the tty and pid from the Xterm to the
  // parent process
  #define TMPFILENAME "./tmptty"
  // Ensure the temporary file doesn't exist
  remove(TMPFILENAME);
  // Fork between the parent process executing the main algorithm and
  // the child process running the Xterm
  pid_t pid = fork();
  // If we are in the child process
  if (pid == 0) {
    // Open a new Xterm, executing '(echo $$ && tty) > ./tmptty'
    // to save its pid and tty into the temporary file, followed
    // by a bash to avoid the window closing right after the command
    char cmd[] = "/usr/bin/xterm";
    char* argv[] = {
      "xterm",
      "-e",
      "(echo $$ && tty) > " TMPFILENAME " ; bash",
      NULL
      };
    execv(cmd, argv);
    // Never reach here as the child get replaced by the Xterm
  // Else, we are in the parent process
  } else {
    // Wait one second to give time to the Xterm to write its pid and
    // tty into the temporary file
    sleep(1);
    // Open the temporary file
    FILE* fp = fopen(TMPFILENAME, "r");
    // Variable to memorize the tty of the Xterm
    char tty[100] = {'\0'};
    // Read the pid and tty of the Xterm
    fscanf(fp, "%d %s\n", &pid, tty);
    // Close and remove the temporary file
    fclose(fp);
    remove(TMPFILENAME);
    // Display the information about the Xterm
    printf("Xterm attached to tty %s and has pid %d\n", tty, pid);
    // Open the tty
    FILE* ftty = fopen(tty, "w");
    // Simulate a process sending info toward the Xterm and its own
    // console
```

```
    for (int i = 0; i < 10; ++i) {
      fprintf(ftty, "Toward Xterm: %d\n", i);
      printf("Toward parent's terminal: %d\n", i);
      sleep(1);
    }
    // Close the tty
    fclose(ftty);
    // Kill the Xterm
    kill(pid, SIGKILL);
  }
  // Return success code
  return 0;
}
```

2 Interface

```
// ============ RESPUBLISH.H ================

#ifndef RESPUBLISH_H
#define RESPUBLISH_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"

// ---- TextOMeter

// ================= Define ===================

#define TEXTOMETER_TTY_FILENAME "/tmp/textometertty.tmp"
#define TEXTOMETER_TTY_MAXLENGTH 100
#define TEXTOMETER_XTERM_CMD "/usr/bin/xterm"

// ================= Data structure ===================

typedef struct TextOMeter {
  // Title
  char* _title;
  // Width
  int _width;
  // Height
  int _height;
  // PID of the associated xterm
  pid_t _pid;
  // File pointer to the tty of the associated xterm
  char _tty[TEXTOMETER_TTY_MAXLENGTH];
  FILE* _fp;
} TextOMeter;

// ================ Functions declaration ====================

// Create a new TextOMeter of 'width' columns and 'height' lines and
```

```
// 'title' as title of the attached xterm
// May return NULL if the creation of the Xterm failed
TextOMeter* TextOMeterCreate(char* const title,
  const int width, const int height);

// Free the memory used by the TextOMeter 'that'
void TextOMeterFree(TextOMeter** that);

// Clear the content of the TextOMeter 'that'
// Return true if the content could be cleared, false else
#if BUILDMODE != 0
inline
#endif
bool TextOMeterClear(TextOMeter* const that);

// Print the string 'str' on the TextOMeter 'that'
// Return true if the content could be printed, false else
#if BUILDMODE != 0
inline
#endif
bool TextOMeterPrint(TextOMeter* const that, const char* const str);

// Flush the stream of the TextOMeter 'that'
#if BUILDMODE != 0
inline
#endif
void TextOMeterFlush(TextOMeter* const that);

// ================ Inliner ====================

#if BUILDMODE != 0
#include "respublish-inline.c"
#endif


#endif
```

3 Code

3.1 respublish.c

```
// ============ RESPUBLISH.C ================

// ================= Include =================

#include "respublish.h"
#if BUILDMODE == 0
#include "respublish-inline.c"
#endif

// ================ Define ====================

// ================ Functions implementation ====================

// Create a new TextOMeter of 'width' columns and 'height' lines and
// 'title' as title of the attached xterm
// May return NULL if the creation of the Xterm failed
TextOMeter* TextOMeterCreate(char* const title,
```

```
  const int width, const int height) {
#if BUILDMODE == 0
  if (title == NULL) {
    ResPublishErr->_type = PBErrTypeNullPointer;
    sprintf(ResPublishErr->_msg, "'title' is null");
    PBErrCatch(ResPublishErr);
  }
  if (width <= 0) {
    ResPublishErr->_type = PBErrTypeInvalidArg;
    sprintf(ResPublishErr->_msg, "'width' is invalid (0<%d)", width);
    PBErrCatch(ResPublishErr);
  }
  if (height <= 0) {
    ResPublishErr->_type = PBErrTypeInvalidArg;
    sprintf(ResPublishErr->_msg, "'height' is invalid (0<%d)", height);
    PBErrCatch(ResPublishErr);
  }
#endif
  // Declare the new TextOMeter
  TextOMeter* that = NULL;
  // Ensure the temporary file to get the tty and pid doesn't exists
  remove(TEXTOMETER_TTY_FILENAME);
  // Fork to create the Xterm
  pid_t pid = 0;
  if ((pid = fork()) == 0) {
    // Create the Xterm
    char cmd[] = TEXTOMETER_XTERM_CMD;
    char geometry[100];
    sprintf(geometry, "%dx%d", width, height);
    char* argv[] = {
      "xterm",
      "-xrm",
      "'XTerm.vt100.allowTitleOps: false'",
      "-T",
      title,
      "-geometry",
      geometry,
      "-e",
      "(echo $$ && tty) > " TEXTOMETER_TTY_FILENAME " ; bash",
      NULL
      };
    if (execv(cmd, argv) == -1) {
      fprintf(stderr, "TextOMeter '%s' couldn't create the Xterm, "
        "execv failed (%d)\n", title, errno);
    }
  } else {
    // Wait for the tty and pid from the Xterm
    FILE *fp = NULL;
    sleep(1);
    int wait = 10000;
    do {
      fp = fopen(TEXTOMETER_TTY_FILENAME, "r");
      wait--;
    } while (fp == NULL && wait > 0);
    if (fp == NULL) {
      fprintf(stderr,
        "TextOMeter '%s' couldn't read the tty and pid\n", title);
    } else {
      // Read the tty and pid from the Xterm
      char tty[100];
      if (fscanf(fp, "%d %s\n", &pid, tty) == EOF) {
        fprintf(stderr,
```

```
          "TextOMeter '%s' couldn't read the tty and pid\n", title);
      } else {
        // Allocate memory for the new TextOMeter
        that = PBErrMalloc(ResPublishErr, sizeof(TextOMeter));
        // Set properties
        that->_width = width;
        that->_height = height;
        that->_title = strdup(title);
        that->_pid = pid;
        strcpy(that->_tty, tty);
        // Open the tty to send message to the Xterm
        that->_fp = fopen(that->_tty, "w");
      }
      // Close the temporary file and delete it
      fclose(fp);
      remove(TEXTOMETER_TTY_FILENAME);
    }
  }
  // Return the new TextOMeter
  return that;
}

// Free the memory used by the TextOMeter 'that'
void TextOMeterFree(TextOMeter** that) {
  // Check argument
  if (that == NULL || *that == NULL)
    // Nothing to do
    return;
  // Close the file pointer to the tty
  if ((*that)->_tty != NULL)
    fclose((*that)->_fp);
  // Kill the terminal
  if (kill((*that)->_pid, SIGTERM) == -1) {
    fprintf(stderr,
      "Couldn't kill the TextOMeter '%s'\n", (*that)->_title);
  }
  // Free memory
  free((*that)->_title);
  free(*that);
  *that = NULL;
}
```

3.2 respublish-inline.c

```
// ============ RESPUBLISH-INLINE.C ================

// ================ Functions implementation ====================

// Clear the content of the TextOMeter 'that'
// Return true if the content could be cleared, false else
#if BUILDMODE != 0
inline
#endif
bool TextOMeterClear(TextOMeter* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ResPublishErr->_type = PBErrTypeNullPointer;
    sprintf(ResPublishErr->_msg, "'that' is null");
    PBErrCatch(ResPublishErr);
```

```
  }
#endif
  // Print on the tty as many line as the height of the Xterm
  for (int i = that->_height; i--;) {
    if (fprintf(that->_fp, "\n") < 0)
      return false;
    fflush(that->_fp);
  }
  return true;
}

// Print the string 'str' on the TextOMeter 'that'
// Return true if the content could be printed, false else
#if BUILDMODE != 0
inline
#endif
bool TextOMeterPrint(TextOMeter* const that, const char* const str) {
#if BUILDMODE == 0
  if (that == NULL) {
    ResPublishErr->_type = PBErrTypeNullPointer;
    sprintf(ResPublishErr->_msg, "'that' is null");
    PBErrCatch(ResPublishErr);
  }
  if (str == NULL) {
    ResPublishErr->_type = PBErrTypeNullPointer;
    sprintf(ResPublishErr->_msg, "'str' is null");
    PBErrCatch(ResPublishErr);
  }
#endif
  // Print the string on the tty
  if (fprintf(that->_fp, "%s", str) < 0)
    return false;
  return true;
}

// Flush the stream of the TextOMeter 'that'
#if BUILDMODE != 0
inline
#endif
void TextOMeterFlush(TextOMeter* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ResPublishErr->_type = PBErrTypeNullPointer;
    sprintf(ResPublishErr->_msg, "'that' is null");
    PBErrCatch(ResPublishErr);
  }
#endif
  fflush(that->_fp);
}
```

4 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0
```

```
all: pbmake_wget main xterm

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=respublish
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$

xterm: xterm.c
gcc -o xterm xterm.c
```

5 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include "respublish.h"

void UnitTestTextOMeter() {
  TextOMeter* meterA = TextOMeterCreate("UnitTestA", 40, 20);
  TextOMeter* meterB = TextOMeterCreate("UnitTestB", 40, 20);
  if (meterA == NULL || meterB == NULL) {
    ResPublishErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ResPublishErr->_msg, "TextOMeterCreate NOK");
    PBErrCatch(ResPublishErr);
  }
  if (!TextOMeterClear(meterA) || !TextOMeterClear(meterB)) {
    ResPublishErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ResPublishErr->_msg, "TextOMeterClear NOK");
    PBErrCatch(ResPublishErr);
  }
  sleep(2);
  if (!TextOMeterPrint(meterA, "Message from UnitTestTextOMeterA") ||
    !TextOMeterPrint(meterB, "Message from UnitTestTextOMeterB")) {
    ResPublishErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ResPublishErr->_msg, "TextOMeterClear NOK");
    PBErrCatch(ResPublishErr);
  }
  TextOMeterFlush(meterA);
  TextOMeterFlush(meterB);
  sleep(2);
  TextOMeterFree(&meterA);
  TextOMeterFree(&meterB);
```

```
  if (meterA != NULL || meterB != NULL) {
    ResPublishErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ResPublishErr->_msg, "TextOMeterFree NOK");
    PBErrCatch(ResPublishErr);
  }
  printf("UnitTestTextOMeter OK\n");
}

void UnitTestAll() {
  UnitTestTextOMeter();
  printf("UnitTestAll OK\n");
}

int main() {
  UnitTestAll();
  // Return success code
  return 0;
}
```

6 Unit tests output

```
UnitTestTextOMeter OK
UnitTestAll OK
```