UNIVERSIDADE FEDERAL DA PARAÍBA
Centro de Informática

Candidato: Kelvin Williams Neves Bernardo
Email: kelvin-williams@hotmail.com
Github: github.com/kelvin-williams

Guia das tarefas

Este documento é um breve guia e relatório das tarefas realizadas e como elas foram planejadas e implementadas no código, sua leitura não é obrigatória mas acredita-se que seja de ajuda para o(a) avaliador(a).

1. **Localização dos arquivos modificados e/ou criados:**

Tarefa 1:

- nclua/tests/**test-bitwise-operations.lua**
  Arquivo com script em LUA que realiza os testes ASSERT nas funções bitwise da biblioteca bit32 do LUA.

- nclua/tests/**Makefile.am**
  Arquivo com diretivas para que o autotools possa criar o makefile do projeto, precisa ser modificado para que o novo teste seja integrado ao projeto.

Tarefa 2:

- nclua/tests/**test-libcurl-http.c**
  Programa em C que testa as funcionalidades básicas da biblioteca libcurl.

- nclua/tests/**Makefile.am**
  Assim como na Tarefa 1 foi modificado para que o teste fosse integrado ao projeto e também para que a biblioteca Libcurl pudesse ser utilizada pelos testes.

- nclua/build-aux/**libcurl.m4**
  Arquivo com macros e diretivas necessárias para integrar o libcurl ao projeto, foi retirado da pasta da biblioteca no sistema.

- nclua/**configure.ac**
  Arquivo com diretivas de configuração do projeto, foi modificado para que a biblioteca libcurl pudesse ser utilizada no projeto.

- nclua/**Makefile.am**
  Foi modificado para que o libcurl.m4 pudesse ser utilizado na configuração.

Tarefa 3:

- nclua/nclua/event/**hello_c.c**
  Arquivo com funções em C que testam a comunicação C-LUA e LUA-C.
- nclua/nclua/event/**hello.lua**
  Arquivo com funções em LUA que testam a comunicação LUA-C e C-LUA.

- nclua/nclua/event/**Makefile.am**
  Foi modificado para que as funções de hello_c.c pudessem ser utilizadas por módulos em LUA do projeto.

- nclua/tests/**test-hello.lua**
  Script em LUA que testa se as funções de comunicação de hello_c.c e hello.lua estão funcionando corretamente.

- nclua/**hello_c.la** e **hello_c.so**
  Arquivos de configuração criados a partir do hello_c.c pelo libtools.

2. Relatório das Tarefas

2.1 Tarefa 1

Primeiramente foi criado o arquivo **test-bitwise-operations.lua** no local indicado, e depois, antes que o mesmo fosse preenchido ele foi adicionado ao projeto modificando o **Makefile.am** da pasta tests:

```
scripts+= test-event-pointer-api.lua
scripts+= test-event-pointer-check.lua
scripts+= test-event-pointer-filter.lua
# event.init
scripts+= test-event-init-api.lua
scripts+= test-event-init-init.lua
scripts+= test-event-init-load.lua
scripts+= test-event-init-unload.lua
scripts+= test-event-init-post.lua
scripts+= test-event-init-register.lua
scripts+= test-event-init-unregister.lua
scripts+= test-event-init-cycle.lua
scripts+= test-event-init-uptime.lua
scripts+= test-event-init-timer.lua
# libnclua
scripts+= test-libnclua-echo.sh
# bitwise operations
scripts+= test-bitwise-operations.lua
# basic C-Lua communication tests
scripts+= test-hello.lua
check_SCRIPTS= $(scripts)
```

Depois foi preenchido o arquivo de teste criado com os testes ASSERT de cada operação bitwise da biblioteca **bit32** do LUA, como mostrado no exemplo abaixo:

```
-- Testing function bit32.band
-- And of ...011 and ...001, that results in ...001 (1)
ASSERT (bit32.band(3,1) == (1 % 2^32))

-- Testing function bit32.bnot
-- Negating ...0001 results in ...1110 (-2)
ASSERT (bit32.bnot(1) == (-2 % 2^32))
```

Também foi feito um teste ASSERT que resulta em negativo propositalmente:

```
-- Testing function bit32.btest
-- And of ...011 and ...001 equals ...001 (1)
ASSERT (bit32.btest(3,1))
-- And of ...011 and ...100 equals ...000 (0), Expected False
ASSERT (bit32.btest(3,4))
```

Porém ele foi comentado para que todos os outros testes pudessem ser feitos normalmente.

2.2 Tarefa 2

A primeira coisa a ser feita foi instalar a biblioteca libcurl, executando este comando no terminal:

```
osboxes@osboxes:~$ sudo apt install libcurl3-gnutls
```

Depois foi copiado o arquivo **libcurl.m4**, localizado em: **/usr/share/aclocal**, para a pasta: nclua/build-aux.
E foi modificado o **Makefile.am** principal para que este arquivo pudesse ser utilizado:

```
EXTRA_DIST+=\
  .version\
  README.md\
  bootstrap\
  build-aux/git-log-fix\
  build-aux/git-version-gen\
  build-aux/gitlog-to-changelog\
  build-aux/manywarnings.m4\
  build-aux/perl.m4\
  build-aux/syntax-check-copyright\
  build-aux/syntax-check\
  build-aux/useless-if-before-free\
  build-aux/util.m4\
  build-aux/warnings.m4\
  build-aux/libcurl.m4\
  maint.mk\
  $(NULL)
```

```
MAINTAINERCLEANFILES+=\
  .version\
  ChangeLog\
  INSTALL\
  aclocal.m4\
  build-aux/ar-lib\
  build-aux/compile\
  build-aux/config.guess\
  build-aux/config.sub\
  build-aux/depcomp\
  build-aux/install-sh\
  build-aux/libtool.m4\
  build-aux/ltmain.sh\
  build-aux/ltoptions.m4\
  build-aux/ltsugar.m4\
  build-aux/ltversion.m4\
  build-aux/lt~obsolete.m4\
  build-aux/missing\
  build-aux/mkinstalldirs\
  build-aux/test-driver\
  build-aux/libcurl.m4\
  configure\
  lib/config.h.in\
  lib/config.h.in~\
  $(NULL)
```

Depois também foi modificado o **configure.ac** para que a **libcurl** fosse integrada ao projeto:

```
# Dependencies version.
m4_define([cairo_required_version], [1.10.2])
m4_define([glib_required_version], [2.32.4])
m4_define([gstreamer_required_version], [1.4.0])
m4_define([gtk_required_version], [3.4.2])
m4_define([lua_required_version], [5.2])
m4_define([pango_required_version], [1.30.0])
m4_define([soup_required_version], [2.42])
m4_define([libcurl_required_version], [7.47.0])
AC_SUBST([CAIRO_REQUIRED_VERSION], cairo_required_version)
AC_SUBST([GLIB_REQUIRED_VERSION], glib_required_version)
AC_SUBST([GSTREAMER_REQUIRED_VERSION], gstreamer_required_version)
AC_SUBST([GTK_REQUIRED_VERSION], gtk_required_version)
AC_SUBST([LUA_REQUIRED_VERSION], lua_required_version)
AC_SUBST([PANGO_REQUIRED_VERSION], pango_required_version)
AC_SUBST([SOUP_REQUIRED_VERSION], soup_required_version)
AC_SUBST([LIBCURL_REQUIRED_VERSION], libcurl_required_version)
```

```
# Check for libcurl (optional).
AU_VERSION_BREAK([libcurl], curl_required_version)
AU_CHECK_OPTIONAL_PKG([event-http], [build the http event class], [],
 [LIBCURL], libcurl >= libcurl_required_version,
 [AC_LANG_PROGRAM([[
#include <curl/curl.h>
#if !CURL_AT_LEAST_VERSION  \
    (LIBCURL_REQUIRED_MAJOR,\
     LIBCURL_REQUIRED_MINOR,\
     LIBCURL_REQUIRED_MICRO)
# error "curl is too old"
#endif
 ]])],
 [AC_LANG_PROGRAM([[]], [[curl_global_init(0);]])])
```

Depois de integrado ao projeto, o **libcurl** também precisa ser visível para os testes, para isso foi modificado o **Makefile.am** da pasta tests:

```
AM_CFLAGS= $(WERROR_CFLAGS) $(WARN_CFLAGS)\
  $(CAIRO_CFLAGS) $(GLIB_CFLAGS) $(LUA_CFLAGS) $(LIBCURL_CFLAGS)

AM_LDFLAGS= -static $(CAIRO_LIBS) $(GLIB_LIBS) $(LUA_LIBS) $(LIBCURL_LIBS)
LDADD= $(top_builddir)/lib/libnclua.la

                                    .
                                    .
CLEANFILES+= tests0.def
tests0.def:
        $(AM_V_GEN) (echo EXPORTS;\
        echo luaopen_tests0) >$@

tests0_la_DEPENDENCIES= $(tests0_def_dependency)
tests0_la_CFLAGS= $(AM_CFLAGS)
tests0_la_LDFLAGS= -module $(LT_MODULE_LDFLAGS) -rpath '/force-shared'\
  $(CAIRO_LIBS) $(GLIB_LIBS) $(tests0_export_symbols) $(LIBCURL_CLIBS)
```

E assim foi possível criar o arquivo de teste **test-libcurl-http.c**, o qual utilizou um código de teste exemplo para testar as funcionalidades do **curl**, as modificações necessárias foram feitas para que os ASSERTS funcionassem:

```
int main(void)
{
  /*Example HTTP test from the CURL website*/
  TEST_BEGIN
  {

  CURL *curl;
  CURLcode res;

  /*If curl_global_init doesn't return zero something went wrong and the other curl functions can't be used */
  ASSERT (curl_global_init(CURL_GLOBAL_DEFAULT) == 0);

  /* If curl_easy_init returns NULL it means that something went wrong*/
  ASSERT ((curl = curl_easy_init()) != NULL);
```

O **Makefile.am** da pasta tests foi modificado para que **test-libcurl-http.c** pudesse ser executado:

```
# Test programs.
programs=
programs+= test-libnclua-open
programs+= test-libnclua-open-x
programs+= test-libnclua-close
programs+= test-libnclua-debug-get-surface
programs+= test-libnclua-debug-dump-surface
programs+= test-libnclua-paint
programs+= test-libnclua-resize
programs+= test-libnclua-cycle
programs+= test-libncluaw-open
programs+= test-libncluaw-at-panic
programs+= test-libncluaw-at-panic-x
programs+= test-libncluaw-event-clone
programs+= test-libncluaw-event-clone-x
programs+= test-libncluaw-event-free-x
programs+= test-libncluaw-debug-get-surface
programs+= test-libncluaw-debug-dump-surface
programs+= test-libncluaw-cycle
programs+= test-libcurl-http
check_PROGRAMS= $(programs)
```

2.3 Tarefa 3

Primeiramente foi criado o arquivo **hello_c.c** que possui as funções que serão utilizadas para se comunicar com o LUA, e também o wrapper para que as funções pudessem ser utilizadas pelo módulo LUA:

```
/* This function returns the string "Hello" to the Lua stack*/
static int helloFunc(lua_State * L){

        lua_pushstring(L, "Hello");
        printf("C module sent Hello message\n");

        return 1;
}
```

.
.

```c
static const struct luaL_Reg hello_funcs[] = {
  {"_hello", helloFunc},
  {"_listen", listenForAnswer},
  {NULL, NULL}
};

int luaopen_nclua_event_hello_c(lua_State * L);

int luaopen_nclua_event_hello_c(lua_State * L){

  G_TYPE_INIT_WRAPPER ();
  lua_newtable (L);
        luax_newmetatable(L,"nclua.event.hello_c");
        luaL_setfuncs(L, hello_funcs,0);
        return 1;
}
```

Para que este arquivo pudesse ser referenciado por arquivos LUA do projeto foi necessária a modificação do **Makefile.am** da pasta event:

```
# hello
eventlib_LTLIBRARIES+= hello_c.la
if OS_WIN32
hello_c_def_dependency= hello_c.def
def_dependency+= $(hello_c_def_dependency)
hello_c_export_symbols= -export-symbols $(hello_c_def_dependency)\
  -no-undefined
endif
hello_c_la_DEPENDENCIES= $(hello_c_def_dependency)
hello_c_la_CFLAGS= $(AM_CFLAGS) $(GLIB_CFLAGS) $(GIO_CFLAGS)
hello_c_la_LDFLAGS= $(AM_LDFLAGS) $(GLIB_LIBS) $(GIO_LIBS)\
  $(hello_c_export_symbols)
hello_c_la_SOURCES=\
  hello_c.c\
  $(NULL)
```

Depois de realizar estas modificações e recompilar o projeto, foram gerados automaticamente mais dois arquivos na pasta event: **hello_c.la** e **hello_c.so**, eles são utilizados para realizar a linkagem com os outros módulos do projeto após a compilação.

Com as funções de comunicação em C prontas, o próximo passo foi criar o arquivo **hello.lua**, que invoca as funções de **hello_c.c** e envia uma resposta de volta para ele. Também foi necessário modificar o **Makefile.am** da pasta event para que hello.lua pudesse ser utilizado no projeto:

```
dist_eventdata_DATA=\
  check.lua\
  engine.lua\
  init.lua\
  key.lua\
  ncl.lua\
  pointer.lua\
  queue.lua\
  user.lua\
  hello.lua\
  $(NULL)
```

O próximo passo é criar o arquivo de teste **test-hello.lua** para avaliar se as funções de comunicação dos dois módulos estão funcionando corretamente e imprimir as saídas de cada um, o resultado foi o seguinte:

```
C module sent Hello message
Lua module received:    Hello   , asnwering...
C module confirming, message Received : I'm here.
PASS test-hello.lua (exit status: 0)
```

Portanto foi comprovado que os módulos conseguiram se comunicar corretamente.

3. Conclusão

Todas as tarefas foram concluídas inteiramente e com os resultados esperados, portanto este documento se encerra aqui.