

# Библиотека 'lproc\_queue'

## Расширение языка lua 5.3+

3 декабря 2023 г.

# Содержание

<b>1</b>	<b>Назначение</b>	<b>3</b>
<b>2</b>	<b>Сборка</b>	<b>4</b>
2.1	Сборка под Eclipse CDT . . . . .	4
2.2	Сборка под make . . . . .	4
2.3	Результат компиляции . . . . .	4
2.4	Первый запуск . . . . .	4
<b>3</b>	<b>Поддерживаемые платформы</b>	<b>5</b>
3.1	Ubuntu 20.04 . . . . .	5
<b>4</b>	<b>Описание функций</b>	<b>6</b>
4.1	<code>version()</code> - версия библиотеки . . . . .	6
4.2	<code>delay_ms()</code> - задержка в миллисекундах . . . . .	7
4.3	Процессы . . . . .	8
4.3.1	<code>proc_start()</code> - создание и запуск процесса . . . . .	8
4.3.2	<code>proc_exit()</code> - самозавершение дочернего процесса . . . . .	9
4.4	Очереди сообщений . . . . .	10
4.4.1	<code>queue_create()</code> - создание очереди сообщений . . . . .	10
4.4.2	<code>queue_destroy()</code> - уничтожение очереди сообщений . . . . .	11
4.4.3	<code>queue_push()</code> - неблокирующее помещение сообщения в очередь . . . . .	12
4.4.4	<code>queue_nb_pop()</code> - неблокирующее извлечение сообщения из очереди . . . . .	13
4.5	Семафоры . . . . .	15
4.5.1	<code>sem_create()</code> - создание семафора . . . . .	15
4.5.2	<code>sem_destroy()</code> - уничтожение семафора . . . . .	16
4.5.3	<code>sem_post()</code> - неблокирующая установка семафора . . . . .	17
4.5.4	<code>sem_timedwait_ms()</code> - блокирующее ожидание семафора или истечения таймаута . . . . .	18
4.6	<code>debug_mode()</code> - включение режима выдачи отладочных сообщений . . . . .	19

# 1 Назначение

Проект представляет собой библиотеку многопоточности, основанной на библиотеке Linux pthread для Lua 5.3+ (<https://www.lua.org/pil>).

Исходники основаны на проекте из книжки <https://www.lua.org/pil/3/lproc.c>. Недостатком оригинального проекта является то, что межпоточная коммуникация основана на одиночных сообщениях, которые вызывают блокировку, скажем, отправляющего сообщение потока в случае если получатель ещё не прочитал предыдущее сообщение. В результате потоки ждут друг друга и нередко всё самоблокируется.

В данной библиотеке механизм сообщений между потоками полностью заменён. Новый механизм основан на очередях сообщений. Каждая очередь состоит из 256 элементов. Каждый элемент содержит 256 байт. Эти размеры можно переопределить на стадии компиляции. Всего таких очередей может быть сколько угодно. Очереди организованы в кольцевой двусвязанный список.




Благодаря механизму очередей сообщений становится возможным реализация неблокирующих вызовов 'положить в очередь' и 'извлечь из очереди'. Таким образом, потоки выполняются независимо друг от друга и в тоже время обмениваются друг с другом данными.

Исключительные ситуации 'очередь пуста' и 'очередь заполнена' обрабатываются и возвращаются в Lua.

## 2 Сборка

Для сборки нужны пакеты `lua5.3`, `liblua5.3-dev`, `make`, `gcc`.

### 2.1 Сборка под Eclipse CDT

1. Открыть Eclipse CDT.
2. **File** → **Import** → **Existing Project into Workspace**, нажмите 
3. В **Select root directory** укажите путь к файлу `.project` проекта, нажмите 
4. Нажмите . Результат появится папке `./Release*`

### 2.2 Сборка под make

```
cd ~/lproc_queue/lproc_queue
make
```

При необходимости, нужно отредактировать пути к компилятору и lua в файле `makefile`.

### 2.3 Результат компиляции

В процессе компиляции создаётся `lproc_queue.so` lua - библиотека.

### 2.4 Первый запуск

Тестовый проект лежит в папке `lproc_queue/lproc_queue/tests/probe_01/` рядом с символической ссылкой на библиотеку. Для запуска теста надо перейти в эту папку и ввести команду.

```
lua5.3 ./probe_01.lua
```

## **3 Поддерживаемые платформы**

### **3.1 Ubuntu 20.04**

Проект собирается, но тестировался не в полном объеме...

## 4 Описание функций

### 4.1 `version()` - версия библиотеки

Таблица 1: Функция `version()`

(string version) = version()				
№	Тип	Имя	Описание	По умолчанию
Возвращаемые значения				
1	string	version	Строка с версиями библиотек	

```
local lproc_queue = require('lproc_queue')  
  
local version = lproc_queue.version()    --[[ <--- ]]  
print(version)
```

## 4.2 `delay_ms()` - задержка в миллисекундах

Таблица 2: Функция `delay_ms(integer ms)`

delay_ms()				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	integer	ms	На сколько миллисекунд задержать	

```
local lproc_queue = require('lproc_queue')
...
lproc_queue.delay_ms(300)  --[[ <--- ]]
```

## 4.3 Процессы

### 4.3.1 `proc_start()` - создание и запуск процесса

Таблица 3: Функция `proc_start()`

(boolean ret) = proc_start(string proc_body)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	string	proc_body	Строка текста, содержащая Lua - код дочернего процесса	
Возвращаемые значения				
1	boolean	ret	<code>false</code> - в случае ошибки; <code>true</code> - в случае успеха.	

```
local lproc_queue = require('lproc_queue')

local proc = [[
-- Библиотека 'lproc_queue' автоматически подключается к созданному процессу
while true do
    lproc_queue.delay_ms(500)
    print('from proc')
end
]]

local ret = lproc_queue.proc_start(proc)  --[[ <--- ]]

if( ret == false) then
    print('ERROR')
    os.exit()
end

while true do
    lproc_queue.delay_ms(500)
end
```



### 4.3.2 `proc_exit()` - завершение дочернего процесса

Таблица 4: Функция `proc_exit()`

proc_exit()				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
			Функция вызывается из дочернего процесса, не принимает аргументов	
Возвращаемые значения				
			Не возвращает аргументов	

```
local lproc_queue = require('lproc_queue')

local proc = [[
-- Библиотека 'lproc_queue' автоматически подключается к созданному процессу

    lproc_queue.delay_ms(500)
    print('from proc')

    lproc_queue.proc_exit() --[=[ <--- ]=]
]]

local ret = lproc_queue.proc_start(proc)

if( ret == false) then
    print('ERROR')
end

while true do
    lproc_queue.delay_ms(500)
end
```

## 4.4 Очереди сообщений

### 4.4.1 `queue_create()` - создание очереди сообщений

Таблица 5: Функция `queue_create()`

(integer queue_pointer) = queue_create()				
№	Тип	Имя	Описание	По умолчанию
Возвращаемые значения				
1	integer	queue_pointer	0 - в случае ошибки; указатель на объект очереди - в случае успеха.	

```
local lproc_queue = require('lproc_queue')

local queue__to_proc = lproc_queue.queue_create()  --[[ <--- ]]

if (queue__to_proc == 0) then
    print('ERROR')
end
```

#### 4.4.2 queue\_destroy() - уничтожение очереди сообщений

Таблица 6: Функция queue\_destroy()

(boolean ret) = queue_destroy(integer queue_pointer)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	integer	queue_pointer	Указатель на объект очереди	
Возвращаемые значения				
1	boolean	ret	false - в случае ошибки; true - в случае успеха.	

```
local lproc_queue = require('lproc_queue')

local queue__to_proc = lproc_queue.queue_create()

if (queue__to_proc == 0) then
    print('ERROR')
    os.exit()
end

local ret = lproc_queue.queue_destroy(queue__to_proc)  --[[ <--- ]]

if (ret == false) then
    print('ERROR')
    os.exit()
end
```

#### 4.4.3 `queue_push()` - неблокирующее помещение сообщения в очередь

Таблица 7: Функция `queue_push()`

(boolean ret), (integer queue_space) = queue_push(integer queue_pointer, lstring byte_array)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	integer	queue_pointer	Указатель на объект очереди	
2	lstring	byte_array	Массив байт или строка	
Возвращаемые значения				
1	boolean	ret	<code>false</code> - в случае ошибки; <code>true</code> - в случае успеха.	
2	integer	queue_space	Если первый возвращаемый аргумент <code>true</code> , то количество оставшегося свободного места для сообщений в очереди.	

```

local lproc_queue = require('lproc_queue')

local proc_pattern = [[
local channel_in = <channel_in>
local ret, message, queue_count, queue_space

-- Библиотека 'lproc_queue' автоматически подключается к созданному процессу
while true do
    lproc_queue.delay_ms(500)
    print('from proc')
    ret, message, queue_count = lproc_queue.queue_nb_pop(channel_in)
end
]]

local queue__to_proc = lproc_queue.queue_create()
local proc = string.gsub(proc_pattern, '<channel_in>', string.format("0x%016X",
↪ queue__to_proc))

local ret = lproc_queue.proc_start(proc)

if (ret == false) then
    print('ERROR')
    os.exit()
end

while true do
    lproc_queue.delay_ms(321)
    ret, queue_space = lproc_queue.queue_push(queue__to_proc, 'message')  --[[ <--- ]]
    if (ret == false) then
        print('ERROR')
    end
end
end

```

#### 4.4.4 queue\_nb\_pop() - неблокирующее извлечение сообщения из очереди

Таблица 8: Функция queue\_nb\_pop()

(boolean ret), (lstring byte_array), (integer queue_count) = queue_nb_pop(integer queue_pointer)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	integer	queue_pointer	Указатель на объект очереди	
Возвращаемые значения				
1	boolean	ret	false - в случае ошибки; true - в случае успеха.	
2	lstring	byte_array	Если первый возвращаемый аргумент true, то содержимое сообщения в виде массива байт	
3	integer	queue_count	Если первый возвращаемый аргумент true, то количество накопившихся сообщений в очереди.	

```

local lproc_queue = require('lproc_queue')

local proc_pattern = [[
local channel_in = <channel_in>
local ret, message, queue_count, queue_space

-- Библиотека 'lproc_queue' автоматически подключается к созданному процессу
while true do
    lproc_queue.delay_ms(500)
    print('from proc')
    ret, message, queue_count = lproc_queue.queue_nb_pop(channel_in) --[= [ <--- ]=]
    if( ret == false) then
        print('from proc: ERROR')
    else
        print('from proc: meaaage = \''..tostring(message)..'\'')
    end
end
]]

local queue__to_proc = lproc_queue.queue_create()
local proc = string.gsub(proc_pattern, '<channel_in>', string.format("0x%016X",
↪ queue__to_proc))

local ret = lproc_queue.proc_start(proc)

if (ret == false) then
    print('ERROR')
    os.exit()
end

while true do
    lproc_queue.delay_ms(321)
    ret, queue_space = lproc_queue.queue_push(queue__to_proc, 'message')
    if (ret == false) then

```

```
    print('ERROR')  
  end  
end
```

## 4.5 Семафоры

### 4.5.1 `sem_create()` - создание семафора

Таблица 9: Функция `sem_create()`

(integer sem_pointer) = sem_create()				
№	Тип	Имя	Описание	По умолчанию
Возвращаемые значения				
1	integer	sem_pointer	0 - в случае ошибки; указатель на объект семафора - в случае успеха.	

```
local lproc_queue = require('lproc_queue')

local sem__to_proc = lproc_queue.sem_create()  --[[ <--- ]]

if (sem__to_proc == 0) then
    print('ERROR')
end
```

#### 4.5.2 `sem_destroy()` - уничтожение семафора

Таблица 10: Функция `sem_destroy()`

(boolean ret) = sem_destroy(integer sem_pointer)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	integer	sem_pointer	Указатель на объект семафора	
Возвращаемые значения				
1	boolean	ret	<code>false</code> - в случае ошибки; <code>true</code> - в случае успеха.	

```
local lproc_queue = require('lproc_queue')

local sem__to_proc = lproc_queue.sem_create()

if (sem__to_proc == 0) then
    print('ERROR')
    os.exit()
end

local ret = lproc_queue.sem_destroy(sem__to_proc)  --[[ <--- ]]

if (ret == false) then
    print('ERROR')
    os.exit()
end
```



### 4.5.3 `sem_post()` - неблокирующая установка семафора

Таблица 11: Функция `sem_post()`

(boolean ret) = sem_post(integer sem_pointer)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	integer	sem_pointer	Указатель на объект семафора	
Возвращаемые значения				
1	boolean	ret	<code>false</code> - в случае ошибки; <code>true</code> - в случае выставления семафора.	

```
local lproc_queue = require('lproc_queue')

local proc_pattern = []
local semaphore_in = <semaphore_in>
local ret

-- Библиотека 'lproc_queue' автоматически подключается к созданному процессу
lproc_queue.delay_ms(500)

while true do
    print('from proc')
    ret = lproc_queue.sem_timedwait_ms(semaphore_in, 500)
end
]]

local semaphore__to_proc = lproc_queue.sem_create()
local proc = string.gsub(proc_pattern, '<semaphore_in>', string.format("0x%016X",
↪ semaphore__to_proc))

local ret = lproc_queue.proc_start(proc)

if (ret == false) then
    print('ERROR')
    os.exit()
end

while true do
    lproc_queue.delay_ms(321)
    ret = lproc_queue.sem_post(semaphore__to_proc)  --[[ <--- ]]
    if (ret == false) then
        print('ERROR')
    end
end
end
```

#### 4.5.4 `sem_timedwait_ms()` - блокирующее ожидание семафора или истечения таймаута

Таблица 12: Функция `sem_timedwait_ms()`

(boolean ret), (string status) = sem_timedwait_ms(integer sem_pointer, integer timeout_ms)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	integer	sem_pointer	Указатель на объект семафора	
2	integer	timeout_ms	Таймаут ожидания, мС	
Возвращаемые значения				
1	boolean	ret	<code>false</code> - в случае ошибки или таймаута; <code>true</code> - в случае получения семафора.	
2	string	status	Если первый возвращаемый аргумент <code>false</code> , то: <code>timeout</code> - в случае таймаута; <code>error</code> - в случае ошибки.	

```

local lproc_queue = require('lproc_queue')

local proc_pattern = [[
local semaphore_in = <semaphore_in>
local ret

-- Библиотека 'lproc_queue' автоматически подключается к созданному процессу
lproc_queue.delay_ms(500)

while true do
    print('from proc')
    ret = lproc_queue.sem_timedwait_ms(semaphore_in, 500)
end
]]

local semaphore__to_proc = lproc_queue.sem_create()
local proc = string.gsub(proc_pattern, '<semaphore_in>', string.format("0x%016X",
↪ semaphore__to_proc))

local ret = lproc_queue.proc_start(proc)

if (ret == false) then
    print('ERROR')
    os.exit()
end

while true do
    lproc_queue.delay_ms(321)
    ret = lproc_queue.sem_post(semaphore__to_proc)  --[[ <--- ]]
    if (ret == false) then
        print('ERROR')
    end
end
end

```

## 4.6 `debug_mode()` - включение режима выдачи отладочных сообщений

Таблица 13: Функция `debug_mode()`

(string debug_info) = debug_mode(string debug_enable_flags)				
№	Тип	Имя	Описание	По умолчанию
Аргументы				
1	string	debug_enable_flags	<p>Флаги кастомизации отладки. Запись типа <code>0b1000_0000_0000_0000_0000_0000_1111</code> (32-битное число, представленное в битовом формате. Старший бит слева).</p> <p>Младшие биты 3, 2, 1, 0 отвечают за тип отладочных сообщений (записи в коде <code>info_3(/--");</code>, <code>info_2(/--");</code>, <code>info_1(/--");</code>, <code>info_0(/--");</code> соответственно).</p> <p>Все остальные старшие биты отвечают за файлы из которых выводятся отладочные сообщения.</p>	
Возвращаемые значения				
1	string	debug_info	Отчет о кастомизации отладки	

```

local lproc_queue = require('lproc_queue')

local debug_info =
↪ lproc_queue.debug_mode('0b1000_0000_0000_0000_0000_0000_1111') --[[ <--- ]]
print(debug_info)

```