Python logic error when deal with re and muti-threading

Bug Description

When use re and multi-threading it will trigger the bug.

Bug type: `Logic Error`

Test Enviroment:

- `Windows 7 SP1 x64 + python 3.4.3`
- `Linux kali 3.14-kali1-amd64 + python 2.7.3`

Bug 0x00

When we deal with the pattern like the following `(.*(.)?)*bcd\\t\\n\\r\\f\\a\\e\\071\\x3b\\$\\\\\?caxyz` the regexp.search() will be hang up.

POC for bug 0x00

```
#!/usr/bin/python

__author__ = 'bee13oy'

import re

source = "(.*(.)?)*bcd\\t\\n\\r\\f\\a\\e\\071\\x3b\\$\\\\\?caxyz"

def run(source):

    print(source)

    regexp = re.compile(r''+source+'')

    sgroup = regexp.search(source)

run(source)
```

Bug 0x00 Analyze

the following code is where the program trapped into an **infinite loop**:

```
LOCAL(Py_ssize_t)
```

```
SRE(match)(SRE_STATE* state, SRE_CODE* pattern, int match_all)

{

    SRE_CHAR* end = (SRE_CHAR *)state->end;

    Py_ssize_t alloc_pos, ctx_pos = -1;

    Py_ssize_t i, ret = 0;

    Py_ssize_t jump;

    unsigned int sigcount=0;

    SRE(match_context)* ctx;

    SRE(match_context)* nextctx;

    TRACE(("|%p|%p|ENTER\n", pattern, state->ptr));

    DATA_ALLOC(SRE(match_context), ctx);

    ctx->last_ctx_pos = -1;

    ctx->jump = JUMP_NONE;

    ctx->pattern = pattern;

    ctx->match_all = match_all;

    ctx_pos = alloc_pos;

    .....

    /* Cycle code which will never return*/

    for (;;) {

    ++sigcount;

    if ((0 == (sigcount & 0xfff)) && PyErr_CheckSignals())

        RETURN_ERROR(SRE_ERROR_INTERRUPTED);


    switch (*ctx->pattern++) {

    case SRE_OP_MARK:
        /* set mark */

        /* <MARK> <gid> */

        TRACE(("|%p|%p|MARK %d\n", ctx->pattern,

               ctx->ptr, ctx->pattern[0]));

    .....

}
```

Bug 0x01

When we use Python multithreading, and use `join(timeout)` to wait until the **thread terminates** or **timed out**.

- when we create a **while cycle** in the sub thread like the following **testcase 1**. It can be exited normal with join(timeout).
- when we write re code like the following **testcase 2**, It will never return and it will be hang up without any response.

Attention: If you want to test in testcase 1, please comment the code related re. If you want to test in testcase 2, please uncomment the while() cycle.

POC for bug 0x01

```
#!/usr/bin/python

__author__ = 'bee13oy'

import re

import os

import threading

timeout = 2

source = "(.*(.)?)*bcd\\t\\n\\r\\f\\a\\e\\071\\x3b\\$\\\\\\?caxyz"

def run(source):

    ##################################

    # testcase 1 (Normal)

    # while(1):
```

```
    #    print("test1")

    ###################################

    # testcase 2 (Bug:never return..)

    print(source)

    regexp = re.compile(r''+source+'')

    sgroup = regexp.search(source)

def handle():

        try:

            t = threading.Thread(target=run,args=(source,))

            t.setDaemon(True)

            t.start()

            t.join(timeout)

            print("finished...\n")

        except:

            print("exception ...\n")

handle()
```

Bug 0x01 Analyze

Bug 0x01 is based on Bug 0x00.

At first, it will run into the sub-thread, but it can't end normally. At this time, join(timeout) will wait for the sub-thread return or timed out, and try to call timed out function in order that main thread can get the control of the program.

The bug is that the sub-thread was into an infinite loop and the main-thread was into an infinite loop too, which causes the program to be hang up.

By analyzing the source code of Python, we found that:

- sub-thread is into an infinite loop
- main-thread is into an infinite loop

sub-thread trapped into an **infinite** loop is described in bug 0x00 Analyze.

the following code is where **main-thread** trapped into an **infinite** loop:

```
static void take_gil(PyThreadState *tstate)

{

    int err;

    if (tstate == NULL)

        Py_FatalError("take_gil: NULL tstate");

    err = errno;

    MUTEX_LOCK(gil_mutex);
    if (!_Py_atomic_load_relaxed(&gil_locked))

        goto _ready;

    /*Cycle code which will never return*/

    while (_Py_atomic_load_relaxed(&gil_locked)) {

        int timed_out = 0;

        unsigned long saved_switchnum;

        saved_switchnum = gil_switch_number;

        COND_TIMED_WAIT(gil_cond, gil_mutex, INTERVAL, timed_out);

        /* If we timed out and no switch occurred in the meantime, it is
time

           to ask the GIL-holding thread to drop it. */

        if (timed_out &&

            _Py_atomic_load_relaxed(&gil_locked) &&

            gil_switch_number == saved_switchnum) {

            SET_GIL_DROP_REQUEST();

        }

    }

    .....
}
```