Below is the pseudo code of the js_DestroyContext() function:

```
Lock_GC;
    Remove context from list;
    Last = (whether list is empty);
    If (last)
        State = LANDING;
Unlock_GC;

If(last) {
    Wait until gcLevel == 1;
    UnpinAtom();
    Wait until gcPoke=0;
    Js_FreeAtomState();
}
Else {                                  // this part is within the js_GC() function.
    If(state!=UP)
        Return;
    If(gcPoke==false)
        Return;
    gcLevel = 1;      // Real code is more complicated, basic idea is let gcLevel >0;
    js_MarkAtomState();
    gcLevel = 0;
}
```

A legal schedule to prove that this bug is not completely fixed. Here we need three threads, T1 and T2 enter the the js_NewContext and js_DestroyContext function earlier, and then T3 enters the js_NewContext function. I will use semaphore to let this schedule happen.

T1	T2	T3

Enter js_DestroyContext();
Remove context from list;
Last = false;		

	Enter js_DestroyContext();
	Remove context from list;
	Last=true;	

Check whether last == true; No! Check whether state == LAND; No!		
Sem 1		

	Last = true;
	State = LAND;
	
	Wait until gcLevel == 0;
	UnpinPinnedAtom();
	Wait until gcPoke=0;
	Js_FreeAtomState();
	State=DOWN;	

	Sem 2	

		Enter js_NewContext();
		Wait until state==UP or DOWN;
		Append context to list;
		gcPoke=true; // jsapi.c:line 3851

	Sem 3	

Check if gcPoke==true; yes!
gcLevel = 1;
js_MarkAtomState();   // crash!
gcLevel = 0;
gcPoke=0;		

	Sem 4	
		js_InitAtomState;

In order to let this schedule happen, I added four semaphores, Sem 1, sem 2, sem 3, and sem 4 to source code. Please refer to my patch for more details.

Output:

```
heming@rcs:~/bugs/firefox-133773$ sh bug | grep -E
"New|Destroy|sem|Unpin|checks|ContextIterator|return|gcPock|UUU"
js_NewContext tid -138019952 calls JS_APPEND_LINK, now contextnum 1 1
Tid -138019952 sets gcPock to true at func JS_ClearRegExpStatics
js_NewContext tid -146412656 calls JS_APPEND_LINK, now contextnum 2 2
Tid -146412656 sets gcPock to true at func JS_ClearRegExpStatics
Tid -154805360 sets -154805360 to sem 2 tid2
Tid -154805360 sets -154805360 to sem 3 tid1
Tid -154805360 sets -154805360 to sem 4 tid2
Tid -154805360 wait on sem 2
DestroyContext tid -138019952 in critical region, js_ContextNum 2
Tid -138019952 sets -138019952 to sem 1 tid1
Tid -138019952 sets -138019952 to sem 3 tid2
Tid -138019952 sets -138019952 to sem 4 tid1
DestroyContext tid -138019952, gcLevel 0, state 2, last 0
Tid -138019952 sets gcPock to true at func jsRemoveRoot
Tid -138019952 sets gcPock to true at func js_ForceGC
Tid -138019952 post on sem 1
Tid -138019952 wait on sem 3
DestroyContext tid -146412656 in critical region, js_ContextNum 1
Tid -146412656 sets -146412656 to sem 1 tid2
Tid -146412656 sets -146412656 to sem 2 tid1
Tid -146412656 wait on sem 1
Tid -146412656 wakes up on sem 1
DestroyContext tid -146412656, gcLevel 0, state 3, last 1
UnpinAtom, tid -146412656, gcLevel 0, state 3, last 1
Tid -146412656 sets gcPock to true at func jsRemoveRoot
Tid -146412656 sets gcPock to true at func js_ForceGC
Tid -146412656 calls js_ContextIterator() in js_GC()
js_GC tid -146412656, rt state 3, current gcLevel 1, assign gcLevel to 0, return7........
Tid -146412656 post on sem 2
Tid -154805360 wakes up on sem 2
js_NewContext tid -154805360 calls JS_APPEND_LINK, now contextnum 1 1
```

Segmentation fault (core dumped)

GDB back trace:

```
(gdb) bt
#0  0x080819ba in JS_HashTableEnumerateEntries (ht=0x0, f=0x8058f67 <js_atom_marker>,
arg=0xf7c5f21c) at jshash.c:361
#1  0x08059018 in js_MarkAtomState (state=0x80ee4b4, gcflags=0, mark=0x807fa7d
<gc_mark_atom_key_thing>, data=0x80f1e88)
   at jsatom.c:396
#2  0x080804ae in js_GC (cx=0x80f1e88, gcflags=0) at jsgc.c:1216
#3  0x0807ff27 in js_ForceGC (cx=0x80f1e88, gcflags=0) at jsgc.c:1007
#4  0x0805b3a4 in js_DestroyContext (cx=0x80f1e88, gcmode=JS_FORCE_GC) at jscntxt.c:282
#5  0x0804cf65 in JS_DestroyContext (cx=0x80f1e88) at jsapi.c:902
#6  0x08049be7 in RunJavascript ()
```

Conclusion:

From the gdb backtrace we can see that it is because the state->table is NULL, the _HashTableEnumerateEntries triggers a segmentation fault. The reason of segmentation fault is we call js_FreeAtomState first (free the table of JSAtomState), and then we call js_MarkAtomState (access the table of JSAtomState). This symptom is the same as that of the original bug.

Please not e that in my patch, I have commented the “if (!state->table)” check in js_MarkAtomState(). This check is added by the bug #131246, a different bug, and this check will hide the segmentation fault in firefox 133773 bug.

In this bug, we are supposed to avoid the race of js_FreeAtomState vs. js_MarkAtomState, but we failed. I recommend that we reconsider mechanisms to achieve our goals (although the segmentation fault can be hidden by the check added in bug #131246).