


SPONSORED
BY
DIGITALOCEAN
A quick,
easy, and
intuitive
way to
build,
deploy,
manage,
and scale
your apps.

GET \$100 FREE CREDIT

(//srv.buysellads.com/ads/click/x/GTND42JECEADT27WCW74YKQNCTBDVKJNCEBDLZ3JCYVDVK7YF67DT5QKFT7IT2QWCEAITK3LC6
segment=placement:ideonecom;)

HIDE AD • AD VIA BUYSSELLADS (HTTP://SELFserve.BUYSSELLADS.COM/CHECKOUT/NEW/?UTM_SOURCE=IDEONE-COM-02-FLEXBAR&UTM_MEDIUM=AD_VIA_LINK&UTM_CAMPAIGN=IN_UNIT&UTM_TERM=FLEXBAR)

 [new code \(/\)](#)

 [samples \(/samples\)](#)

 [recent codes \(/recent\)](#)

[sign in](#)

(/)

 [fork \(/fork/WEH3Tu\)](/fork/WEH3Tu)



[download \(/plain/WEH3Tu\)](/plain/WEH3Tu)





[copy](#)

<https://ideone.com/WEH3Tu>

language: **C++14 (gcc 8.3)**

created: 2 years ago

visibility:  public (</faq#visibility-of-a-code>)

 [Share or Embed source code](#)

```
<script src="https://ideone.com/e.js/WEH3Tu"
type="text/javascript" ></script>
```

 **sphere online judge**

Learn How to Code

(<http://spoj.com/?>

utm_campaign=permanent&utm_medium=view-right&utm_source=ideone)

Discover > Sphere Engine API
(/sphere-engine)

The brand new service which powers
Ideone!

**Discover > IDE Widget (/sphere-
engine-widget)**

Widget for compiling and running the source
code in a web browser!

```

1. #ifndef NEW_CORO_LIB_H
2. #define NEW_CORO_LIB_H
3. namespace CORO
4. {
5.     using ThreadID = unsigned;
6.     void thd_init();
7.     ThreadID new_thd( void*(*)(void*), void *);
8.     void thread_exit(void *);
9.     int wait_thread(ThreadID id, void **value);
10.    void thd_yield();
11.    void push_value(void*);
12.    void pull_value(void**);
13.    const int WAIT_SUCCESSFUL = 0;
14.    const int NO_THREAD_FOUND = -1;
15.    enum ThreadState : int;
16. }
17. #endif
18.
19. #include <stack>
20. #include <map>
21. #include <queue>
22. #define SIZE 1048576
23. namespace CORO
24. {
25.     int threadCounter = 0;
26.     ThreadID currtid = 0;
27.
28.     enum ThreadState : int
29.     {
30.         newState = 0,
31.         readyState,
32.         runningState,
33.         waitingState,
34.         terminatedState
35.     };
36.
37.     struct TCB
38.     {
39.         TCB()
40.             :tid(threadCounter++),
41.             currtid(currtid),
42.             sp(nullptr),
43.             sbp(nullptr),
44.             paramPtr(nullptr),
45.             retVal(nullptr),
46.             fnPtr(nullptr),
47.             state(newState)
48.         {}
49.         ~TCB()
50.         {
51.             delete [] (char*)sbp;
52.         }
53.         ThreadID tid;
54.         ThreadID currtid = 0;
55.         void *sp;
56.         void *sbp;
57.         void * paramPtr;
58.         void *retVal;
59.         void*(*fnPtr)(void*);
60.         ThreadState state;
61.     };
62.
63.     std::queue<ThreadID> readyThread;
64.     std::map<ThreadID, TCB> allThread;
65.     std::map<ThreadID, ThreadID> waitingThread;
66.     std::stack<ThreadID> newThread;
67.
68.     void thd_init()
69.     {
70.         currtid = new_thd(nullptr, nullptr);
71.
72.         allThread[currtid].currtid = currtid;
73.         allThread[currtid].state = runningState;
74.
75.         newThread.pop();
76.     }

```

```

77.
78. ThreadID new_thd( void*(*thd_function_t)(void*), void *param)
79. {
80.     TCB newTCB;
81.     newTCB.fnPtr = thd_function_t;
82.     newTCB.paramPtr = param;
83.     allThread[newTCB.tid] = newTCB;
84.     newThread.push(newTCB.tid);
85.
86.     return newTCB.tid;
87. }
88.
89. void thread_exit(void *ret_value)
90. {
91.     auto it = waitingThread.find(currtid);
92.     if(it != waitingThread.end())
93.     {
94.         readyThread.push(it->second);
95.         allThread[it->second].state = readyState;
96.     }
97.
98.     allThread[currtid].state = terminatedState;
99.     allThread[currtid].retVal = ret_value;
100.
101.     thd_yield();
102. }
103.
104. int wait_thread(ThreadID id, void **value)
105. {
106.     if(allThread.find(id) != allThread.end())
107.     {
108.         waitingThread[id] = currtid;
109.         allThread[currtid].state = waitingState;
110.         thd_yield();
111.
112.         waitingThread.erase(id);
113.
114.         if(value)
115.             *value = allThread[id].retVal;
116.
117.         allThread[id].state = terminatedState;
118.         allThread.erase(id);
119.
120.         return WAIT_SUCCESSFUL;
121.     }
122.     else
123.         return NO_THREAD_FOUND;
124. }
125.
126. void thd_yield()
127. {
128.     //! context saving
129.     asm volatile("pushfq"
130.                 ::: "rsp");
131.
132.     asm volatile("movq %%rsp, %0\n\t"
133.                 : "+m"
134.                 (allThread[currtid].sp));
135.
136.     if(allThread[currtid].state != terminatedState &&
137.        allThread[currtid].state != waitingState)
138.         allThread[currtid].state = readyState;
139.
140.     if(!newThread.empty())
141.     {
142.         if(allThread[currtid].state == readyState)
143.             readyThread.push(currtid);
144.
145.         currtid = newThread.top();
146.         newThread.pop();
147.         allThread[currtid].state = runningState;
148.
149.         allThread[currtid].sbp = new char[SIZE];
150.         allThread[currtid].sp = (char*)allThread[currtid].sbp + SIZE;
151.
152.         asm volatile("movq %0, %%rsp\n\t"

```

```

153.         :: "m"
154.         (allThread[currtid].sp));
155.
156.         allThread[currtid].retVal = allThread[currtid].fnPtr(allThread[currtid].p
aramPtr);
157.         thread_exit(allThread[currtid].retVal);
158.     }
159.     else if(!readyThread.empty())
160.     {
161.         if(allThread[currtid].state == readyState)
162.             readyThread.push(currtid);
163.
164.         currtid = readyThread.front();
165.         readyThread.pop();
166.
167.         currtid = readyThread.front();
168.         readyThread.pop();
169.
170.         allThread[currtid].state = runningState;
171.     }
172.
173.     asm volatile("movq %0, %%rsp \n\t"
174.         :: "m"
175.         (allThread[currtid].sp));
176.
177.     asm volatile("popfq"
178.         ::: "rsp");
179. }
180.
181. void push_value(void *pushed_value)
182. {
183.
184. }
185.
186. void pull_value(void **pulled_value)
187. {
188.
189. }
190.
191. }
192.
193. #include <stdio.h>
194.
195. void *spin1(void *a)
196. {
197.     int i;
198.     for(i=0; i< 20; i++)
199.     {
200.         printf("SPIN1\n");
201.         if((i+1)%4==0)
202.             CORO::thd_yield();
203.     }
204.     return NULL;
205. }
206.
207. void* spin2(void *a)
208. {
209.     int i;
210.     for(i=0; i< 20; i++)
211.     {
212.         printf("SPIN2\n");
213.         if((i+1)%4==0)
214.             CORO::thd_yield();
215.     }
216.     return NULL;
217. }
218.
219.
220. int main()
221. {
222.     CORO::ThreadID id;
223.     CORO::thd_init();
224.     id = CORO::new_thd(spin2, NULL);
225.     spin1(NULL);
226. }
227.

```

🗨️ comments (0)

 copy

⚙️ stdout

 copy

SPIN1

popular languages:

bash (/l/bash) pascal (/l/pascal) c (/l/c) perl (/l/perl) c# (/l/c-sharp) php (/l/php) c++ (/l/cpp) python (/l/pascal) c++14 (/l/cpp14) python3 (/l/python-3)
haskell (/l/haskell) ruby (/l/ruby) java (/l/java) sqlite (/l/sqlite) objective-c (/l/objective-c) swift (/l/swift) vb.net (/l/vb-net)
list of all supported programming languages (/languages)