


[\(/\)](#)[✎ edit](#) [🔗 fork \(/fork/yWsYNr\)](#) [📄 download \(/plain/yWsYNr\)](#)[📄 copy](#)<https://ideone.com/yWsYNr>language: **C++14 (gcc 8.3)**

created: 0 seconds ago

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

Your solution is public and available from recent codes page for other users. You can at any time change visibility of the code by click on icons above. Learn more about visibility of the code (</faq#visibility-of-a-code>). [don't show again](#) | [close](#)

[↪ Share or Embed source code](#)

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

 **sphere online judge**
Learn How to Code

<http://spoj.com/?>[utm_campaign=permanent&utm_medium=view-right&utm_source=ideone\)](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.  /*!
20.  \file new-coro-lib.cpp
21.  \author Kwek Wei Chong
22.  \par email: k.weichong\@digipen.edu
23.  \par DigiPen Login: k.weichong
24.  \par Course: CS180
25.  \par Programming Assignment 2
26.  \date 16/07/2018
27.  \brief
28.      Copyright (C) 2018 DigiPen Institute of Technology.
29.      Reproduction or disclosure of this file or its contents without the
30.      prior written consent of DigiPen Institute of Technology is prohibited.
31.  */
32. /*****/
33. #include <stack>
34. #include <map>
35. #include <queue>
36. #define size 1048576
37.
38. /*****
39.  /*!
40.  \brief
41.      namespace CORO
42.  */
43. /*****/
44. namespace CORO
45. {
46.     int threadCounter = 0;
47.     ThreadID currtid = 0;
48.
49.     enum ThreadState : int
50.     {
51.         newState = 0,
52.         readyState,
53.         runningState,
54.         waitingState,
55.         terminatedState
56.     };
57.
58. /*****
59.  /*!
60.  \brief
61.      Thread Control Block (TCB) struct/class
62.  */

```

```

63.  /*****
64.  struct TCB
65.  {
66.      TCB()
67.          :tid(threadCounter++),
68.          currtid(currtid),
69.          sp(nullptr),
70.          sbp(nullptr),
71.          paramPtr(nullptr),
72.          retVal(nullptr),
73.          fnPtr(nullptr),
74.          state(newState)
75.      {}
76.      ~TCB()
77.      {
78.          delete [] (char*)sbp;
79.      }
80.      ThreadID tid;
81.      ThreadID currtid;
82.      void *sp;
83.      void *sbp;
84.      void *paramPtr;
85.      void *retVal;
86.      void*(*fnPtr)(void*);
87.      ThreadState state;
88.  };
89.
90.  std::queue<ThreadID> readyThread;
91.  std::map<ThreadID, TCB> allThread;
92.  std::map<ThreadID, ThreadID> waitingThread;
93.  std::stack<ThreadID> newThread;
94.
95.  /*****
96.  /*!
97.   \brief thd_init
98.   Initialize a thread to running state.
99.  */
100. /*****
101. void thd_init()
102. {
103.     currtid = new_thd(nullptr, nullptr);
104.
105.     allThread[currtid].currtid = currtid;
106.     allThread[currtid].state = runningState;
107.
108.     newThread.pop();
109. }
110.
111. /*****
112. /*!
113.   \brief new_thd
114.   Create a new thread.
115.
116.   \param thd_function_t
117.   A function pointer
118.
119.   \param param
120.   A param pointer to void
121.
122.   \return newTCB.tid
123.   The tid of the TCB object
124.  */

```

```

125.  /*****
126.  ThreadID new_thd( void>(*thd_function_t)(void*), void *param)
127.  {
128.      TCB newTCB;
129.      newTCB.fnPtr = thd_function_t;
130.      newTCB.paramPtr = param;
131.      allThread[newTCB.tid] = newTCB;
132.      newThread.push(newTCB.tid);
133.
134.      return newTCB.tid;
135.  }
136.
137.  /*****
138.  /*!
139.      \brief thread_exit
140.          Terminate a current thread
141.
142.      \param ret_value
143.          The return value of the thread
144.      */
145.  /*****
146.  void thread_exit(void *ret_value)
147.  {
148.      /** Set all waiting thread to ready and terminate thread and set
149.          return value and yield it */
150.      if(waitingThread.find(currtid) != waitingThread.end())
151.      {
152.          readyThread.push(currtid);
153.          allThread[currleid].state = readyState;
154.      }
155.
156.      allThread[currleid].state = terminatedState;
157.      allThread[currleid].retVal = ret_value;
158.
159.      thd_yield();
160.  }
161.
162.  /*****
163.  /*!
164.      \brief wait_thread
165.          Waits for a thread to be completed and obtain the
166.          return value of the thread. id identifies the thread to be waited
167.          upon and value should be changed to the return value of the thread
168.          after wait thread successfully completes.
169.
170.      \param id
171.          Thread to be waited upon.
172.
173.      \param value
174.          The value of the thread. e.g. return value
175.
176.      \return
177.          WAIT_SUCCESSFUL: if successfully waited for the thread
178.          NO_THREAD_FOUND: If no thread is found in waiting queue
179.      */
180.  /*****
181.  int wait_thread(ThreadID id, void **value)
182.  {
183.      /**set thread to wait for current thread
184.          and change the state of the running thread to waiting*/
185.      if(allThread.find(id) != allThread.end())
186.      {

```

```

187.         waitingThread[id] = currtid;
188.         allThread[currtid].state = waitingState;
189.         thd_yield();
190.
191.         waitingThread.erase(id);
192.
193.         if(value)
194.             *value = allThread[id].retVal;
195.
196.         allThread[id].state = terminatedState;
197.         allThread.erase(id);
198.
199.         return WAIT_SUCCESSFUL;
200.     }
201.     else
202.         return NO_THREAD_FOUND;
203. }
204.
205. /*****
206.  */
207.  \brief wait_thread
208.      This function causes the current thread to yield the CPU for another
209.      thread (if any) to be scheduled. Process includes 1) context saving,
210.      calling the scheduler or performing scheduled task, and restoring
211.      context.
212.  */
213. /*****
214. void thd_yield()
215. {
216.     //! context saving
217.     asm volatile("push %%rax \n\t"
218.                 "push %%rbx \n\t"
219.                 "push %%rcx \n\t"
220.                 "push %%rdx \n\t"
221.                 "push %%rsi \n\t"
222.                 "push %%rdi \n\t"
223.                 "push %%rbp \n\t"
224.                 "push %%r8 \n\t"
225.                 "push %%r9 \n\t"
226.                 "push %%r10 \n\t"
227.                 "push %%r11 \n\t"
228.                 "push %%r12 \n\t"
229.                 "push %%r13 \n\t"
230.                 "push %%r14 \n\t"
231.                 "push %%r15 \n\t"
232.                 ::: "rsp");
233.
234.     //! save stack pointer
235.     asm volatile("movq %%rsp, %0\n\t"
236.                 : "+m"
237.                 (allThread[currtid].sp));
238.
239.     if(allThread[currtid].state != terminatedState &&
240.        allThread[currtid].state != waitingState)
241.        allThread[currtid].state = readyState;
242.
243.     if(allThread[currtid].state == readyState)
244.        readyThread.push(currtid);
245.
246.     if(!newThread.empty())
247.     {
248.         currtid = newThread.top();

```


```

249.         newThread.pop();
250.         allThread[currtid].state = runningState;
251.
252.         //! stack allocation
253.         allThread[currtid].sbp = new char[size];
254.         allThread[currtid].sp = (char*)allThread[currtid].sbp + size;
255.
256.         //! load rsp
257.         asm volatile("movq %0, %%rsp\n\t"
258.                     :: "m"
259.                     (allThread[currtid].sp));
260.
261.         //! run and exit the thread
262.         allThread[currtid].retVal = allThread[currtid].fnPtr(allThread[currtid].p
aramPtr);
263.         thread_exit(allThread[currtid].retVal);
264.     }
265.     else
266.     {
267.         currtid = readyThread.front();
268.         readyThread.pop();
269.
270.         allThread[currtid].state = runningState;
271.     }
272.
273.     //! load stack pointer and pop it
274.     asm volatile("movq %0, %%rsp \n\t"
275.                 :: "m"
276.                 (allThread[currtid].sp));
277.
278.     asm volatile("pop %%r15 \n\t"
279.                 "pop %%r14 \n\t"
280.                 "pop %%r13 \n\t"
281.                 "pop %%r12 \n\t"
282.                 "pop %%r11 \n\t"
283.                 "pop %%r10 \n\t"
284.                 "pop %%r9 \n\t"
285.                 "pop %%r8 \n\t"
286.                 "pop %%rbp \n\t"
287.                 "pop %%rdi \n\t"
288.                 "pop %%rsi \n\t"
289.                 "pop %%rdx \n\t"
290.                 "pop %%rcx \n\t"
291.                 "pop %%rbx \n\t"
292.                 "pop %%rax \n\t"
293.                 ::: "rsp");
294. }
295.
296. void push_value(void *pushed_value)
297. {
298.
299. }
300.
301. void pull_value(void **pulled_value)
302. {
303.
304. }
305.
306. }
307.
308. #include <stdio.h>
309.


```

```
310. void *spin1(void *a)
311. {
312.     int i;
313.     for(i=0; i< 20; i++)
314.     {
315.         printf("SPIN1\n");
316.         if((i+1)%4==0)
317.             CORO::thd_yield();
318.     }
319.     return NULL;
320. }
321.
322. void* spin2(void *a)
323. {
324.     int i;
325.     for(i=0; i< 20; i++)
326.     {
327.         printf("SPIN2\n");
328.         if((i+1)%4==0)
329.             CORO::thd_yield();
330.     }
331.     return NULL;
332. }
333.
334.
335. int main()
336. {
337.     CORO::ThreadID id;
338.     CORO::thd_init();
339.     id = CORO::new_thd(spin2, NULL);
340.     spin1(NULL);
341. }
342.
```


Success #stdin #stdout 0s 4368KB


 comments (0)

 stdin

 copy

Standard input is empty

 stdout

 copy

SPIN1
SPIN1
SPIN1
SPIN1
SPIN2
SPIN2
SPIN2
SPIN2
SPIN1
SPIN1
SPIN1
SPIN1
SPIN2
SPIN2
SPIN2
SPIN2
SPIN1
SPIN1
SPIN1
SPIN1
SPIN2
SPIN2
SPIN2
SPIN2
SPIN1
SPIN1
SPIN1
SPIN1
SPIN2
SPIN2
SPIN2
SPIN2
SPIN1
SPIN1
SPIN1
SPIN1
SPIN2
SPIN2
SPIN2
SPIN2
SPIN1
SPIN1
SPIN1
SPIN1
SPIN2
SPIN2
SPIN2
SPIN2

Sphere Research Labs (http://sphere-research.com?utm_campaign=permanent&utm_medium=footer&utm_source=ideone). Ideone is powered by
Sphere Engine™ (http://sphere-engine.com/?utm_campaign=permanent&utm_medium=footer&utm_source=ideone)
[home \(/\)](#) [api \(https://sphere-engine.com/?utm_campaign=permanent&utm_medium=sphereengine&utm_source=ideone\)](https://sphere-engine.com/?utm_campaign=permanent&utm_medium=sphereengine&utm_source=ideone) [language](#) [faq \(/faq\)](#)
[credits \(/credits\)](#) [desktop](#) [mobile \(/switch/mobile/l3lxc1locg==\)](#)
[terms of service \(/legal-tos\)](#) [privacy policy \(/legal-pp\)](#) [gdpr info \(/legal-gdpr\)](#)
[Feedback & Bugs \(/ideone/Tools/bug/form/1/link/yWsYNr/compiler/44\)](#)

popular languages:

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