fork (/fork/WEH3Tu) download (/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>

/

```
#ifndef NEW_CORO_LIB_H
#define NEW_CORO_LIB_H
namespace CORO
{
using ThreadID = unsigned;
void thd_init();
ThreadID new_thd( void*(*)(void*), void *);
void thread_exit(void *);
int wait_thread(ThreadID id, void **value);
void thd_yield();
void push_value(void*);
void pull_value(void**);
const int WAIT_SUCCESSFUL = 0;
const int NO_THREAD_FOUND = -1;
enum ThreadState : int;
}
#endif

#include <stack>
#include <map>
#include <queue>
#define SIZE    1048576
namespace CORO
{
    int threadCounter = 0;
    ThreadID currtid = 0;

    enum ThreadState : int
    {
        newState = 0,
        readyState,
        runningState,
        waitingState,
        terminatedState
    };

    struct TCB
    {
        TCB()
            :tid(threadCounter++),
             currtid(currtid),
             sp(nullptr),
             sbp(nullptr),
             paramPtr(nullptr),
             retVal(nullptr),
             fnPtr(nullptr),
             state(newState)
        {}
        ~TCB()
        {
            delete [] (char*)sbp;
        }
        ThreadID tid;
        ThreadID currtid = 0;
        void *sp;
        void *sbp;
        void * paramPtr;
        void *retVal;
        void*(*fnPtr)(void*);
        ThreadState state;
    };

    std::queue<ThreadID> readyThread;
    std::map<ThreadID, TCB> allThread;
    std::map<ThreadID, ThreadID> waitingThread;
    std::stack<ThreadID> newThread;

    void thd_init()
    {
        currtid = new_thd(nullptr, nullptr);

        allThread[currtid].currtid = currtid;
        allThread[currtid].state = runningState;

        newThread.pop();
    }

    ThreadID new_thd( void*(*thd_function_t)(void*), void *param)
    {
        TCB newTCB;
        newTCB.fnPtr = thd_function_t;
        newTCB.paramPtr = param;
        allThread[newTCB.tid] = newTCB;
        newThread.push(newTCB.tid);

        return newTCB.tid;
    }

    void thread_exit(void *ret_value)
    {
        auto it = waitingThread.find(currtid);
        if(it != waitingThread.end())
        {
            readyThread.push(it->second);
            allThread[it->second].state = readyState;
        }

        allThread[currtid].state = terminatedState;
        allThread[currtid].retVal = ret_value;

        thd_yield();
    }

    int wait_thread(ThreadID id, void **value)
    {
        if(allThread.find(id) != allThread.end())
        {
            waitingThread[id] = currtid;
            allThread[currtid].state = waitingState;
            thd_yield();

            waitingThread.erase(id);

            if(value)
                *value = allThread[id].retVal;

            allThread[id].state = terminatedState;
            allThread.erase(id);

            return WAIT_SUCCESSFUL;
        }
        else
            return NO_THREAD_FOUND;
    }

    void thd_yield()
    {
        //! context saving
        asm volatile("pushfq"
                     ::: "rsp");

        asm volatile("movq %%rsp, %0\n\t"
                     : "+m"
                     (allThread[currtid].sp));

        if(allThread[currtid].state != terminatedState &&
           allThread[currtid].state != waitingState)
           allThread[currtid].state = readyState;

        if(!newThread.empty())
        {
            if(allThread[currtid].state == readyState)
                readyThread.push(currtid);

            currtid = newThread.top();
            newThread.pop();
            allThread[currtid].state = runningState;

            allThread[currtid].sbp = new char[SIZE];
            allThread[currtid].sp = (char*)allThread[currtid].sbp + SIZE;

            asm volatile("movq %0, %%rsp\n\t"
                          :: "m"
                          (allThread[currtid].sp));

            allThread[currtid].retVal = allThread[currtid].fnPtr(allThread[currtid].paramPtr);
            thread_exit(allThread[currtid].retVal);
        }
        else if(!readyThread.empty())
        {
            if(allThread[currtid].state == readyState)
                readyThread.push(currtid);

            currtid = readyThread.front();
            readyThread.pop();

            currtid = readyThread.front();
            readyThread.pop();

            allThread[currtid].state = runningState;
        }

        asm volatile("movq %0, %%rsp \n\t"
                     :: "m"
                     (allThread[currtid].sp));

        asm volatile("popfq"
                      ::: "rsp");
    }

    void push_value(void *pushed_value)
    {

    }

    void pull_value(void **pulled_value)
    {

    }

}

#include <stdio.h>

void *spin1(void *a)
{
    int i;
    for(i=0; i< 20; i++)
    {
        printf("SPIN1\n");
        if((i+1)%4==0)
            CORO::thd_yield();
    }
    return NULL;
}

void* spin2(void *a)
{
    int i;
    for(i=0; i< 20; i++)
    {
        printf("SPIN2\n");
        if((i+1)%4==0)
            CORO::thd_yield();
    }
    return NULL;
}


int main()
{
    CORO::ThreadID id;
    CORO::thd_init();
    id = CORO::new_thd(spin2, NULL);
    spin1(NULL);
}

```

Success #stdin #stdout 0s 4284KB comments (0)

stdin copy

```
Standard input is empty
```

stdout copy

```
SPIN1
SPIN1
SPIN1
SPIN1
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN2
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
SPIN1
```