When and why sleep() is needed?

Asked 15 years, 9 months ago Modified 15 years, 9 months ago Viewed 1k times

2

```
cout<<"abcd";

sleep(100);

cout<<'\b';
```

If I want to print the string out and then get back one character , why a sleep() is needed here?

But when using printf in C ,it seems that it is not necessary, why?

```
char* a = "12345";
char* b = "67890";
threadA(){cout<<a;}
threadB(){cout<<b;}

beginthread (threadA);
sleep(100);
beginthread (threadB);
```

In the second pseudo code above ,is it right to use sleep()?

c++ c operating-system

Share Improve this question Follow

asked Mar 18, 2009 at 16:16

MainID

30k ● 19 ● 58 ● 70

5 Sleep() is required when you're really tired. – Perchik Mar 18, 2009 at 16:18

@Perchik : I can't +1 you if you don't post an answer ! ;) – Benoît Mar 18, 2009 at 16:23

@Benoit Didn't want to distract from valid answers. – Perchik Mar 18, 2009 at 16:26

12 Answers

Sorted by: Highest score (default)

For calculating tomorrow date:

25

```
void get_tomorrow_date( struct timeval *date )
{
    sleep( 86400 ); // 60 * 60 * 24
    gettimeofday( date, 0 );
}
```

;)

Share Improve this answer Follow

answered Mar 18, 2009 at 16:23

bayda
13.6k ● 9 ● 41 ● 48

3 Now try to write a calendar application with that :D – utku_karatas Mar 18, 2009 at 16:51

9

There are two subtle issues that you need to understand:

> - Multi-threading
> - I/O and Buffering

I'll try to give you some idea:

> Multi-threading and sleep

Having a `sleep` in a threaded environment makes sense. The `sleep` call makes you wait thereby giving the initial thread some scope to have completed its processing i.e. writing out the string `abcd` to the standard output before the other thread inserts the backspace character. If you didn't wait for the first thread to complete its processing, you'd have written the backspace character first, and then the string `abcd` and wouldn't notice any difference.

> Buffered I/o

I/O typically happens in buffered, non-buffered and semi-buffered states. This can influence how long, if at all, you have to wait for the output to appear on the console.

Your implementation of cout is probably using a buffered model. Try adding a newline or the `endl` at the end of your `cout` statements to print a new line and have it flush immediately, or use `cout << "abcd" << flush;` to flush without printing a new line.

Share

Improve this answer

edited Mar 18, 2009 at 16:48

answered Mar 18, 2009 at 16:25

Follow

Pete Kirkham
49.3k ● 5 ● 94 ● 173

dirkgently
111k ● 16 ● 134 ● 190

There aren't. In the examples you gave, there is no reason for sleep at all. You're running into buffered I/O and you can just use cout.flush() or cout << endl; – Eclipse Mar 18, 2009 at 16:55

@bb: Who are you talking to? Which example are you talking about? – dirkgently Mar 18, 2009 at 18:02

5

In the second case without the sleep there's a *slim* chance that the second thread could start working before the first, resulting in the output "6789012345".

However a "sleep" isn't really the way to handle synchronisation between threads. You'd normally use a semaphore or similar in `threadA()` which `threadB()` has to wait for before doing its work.

Share Improve this answer Follow

answered Mar 18, 2009 at 16:21

Alnitak
340k ● 71 ● 418 ● 502

1 Ugh, yeah... managing threads with sleep causes future programmers to lose sleep. – Kim Reece Mar 18, 2009 at 16:37

2

The reason that the call to sleep makes your code work is because you are using it to turn the potentially parallel execution of the two output stream actions into a single, sequential action. The call to sleep() will allow the scheduler to switch away from the main thread of execution and execute thread A.

If you don't put sleep() in, the order of thread execution is not guaranteed and thread B could well start executing/printing before thread A had a chance to do that.

Share Improve this answer Follow

answered Mar 18, 2009 at 16:23

Timo Geusch
24.3k ● 5 ● 54 ● 71

2

I think you need to understand what `sleep` does in general, and understand why it might exist.

`sleep` does what it sounds like. It instructs the OS to put the requesting task (where a task is a thread of execution) to sleep by removing it from the list of currently running processes and putting it on some sort of wait queue.

Note that there are also times when the OS will put you to sleep whether you like it or not. An example would be any form of blocking I/O, like reading a file from disk. The OS does this so that other tasks may get the CPU's attention while you're off waiting for your data.

One would use `sleep` voluntarily for similar purposes that the OS would. For example, if you have multiple threads and they're waiting on the completion of some computation, you'll probably want to voluntarily relinquish the CPU so that the computation can complete. You may also voluntarily relinquish the CPU so that other threads have a chance to run. For example, if you have a tight loop that's highly CPU-bound, you'll want to `sleep` now and then to give other threads a chance to run.

What it looks like you're doing is sleeping for the sake of something being flushed to stdout so that some other thread won't write to stdout before you. This, however, isn't guaranteed to work. It might work incidentally, but it's certainly not what you'd **want** to do by design. You'd either want to explicitly flush your buffer and not sleep at all, or use some form of synchronization.

As for why `printf` doesn't exhibit those issues... well, it's a crapshoot. Both `printf` and `cout` use some form of buffered output, but the implementation of each may be different.

In summary, it's probably best to remember the following:

- When you want to synchronize, use synchronization primitives.
- When you want to give someone else a chance to run, sleep.
- The OS is better at deciding whether an I/O operation is blocking or not.

Share Improve this answer Follow

answered Mar 18, 2009 at 17:22

FreeMemory

8,604 ● 7 ● 38 ● 50

1

if you're having problems seeing the "abcd" being printed, it's because you're not giving cout an endline character to flush the buffer.

if you put

```
cout << "abcd" << endl;
```

you would be able to see the characters, then it would beep. no sleep necessary.

Share Improve this answer Follow

answered Mar 18, 2009 at 16:24

helloandre
10.7k ● 9 ● 49 ● 64

1

```
while( true )
{
   msgStack.Lock();
   process( msgStack.pop_msg());
   msgStack.Unlock();
   sleep(0);
}
```

Share Improve this answer Follow

answered Mar 18, 2009 at 16:38

dark
89 ● 3

1

sleep in the first example is just to print message a little before you will see "backspace" action. In the second example sleep "can" help. But it is weird. You won't be able to synchronize console outs with sleep in some more complex case.

Share

Improve this answer

Follow

edited Mar 18, 2009 at 16:53

answered Mar 18, 2009 at 16:21

Mykola Golubyev
59.7k ● 15 ● 93 ● 102

\b = backspace, \a = alert(bell) – dirkgently Mar 18, 2009 at 16:27

@Mykola: But why I can print message before the "backspace" action when using sleep? – MainID Mar 18, 2009 at 17:01

@Jinx: Why not? You always could. Pause just for "notice something" effect (as for me). Nothing more. – Mykola Golubyev Mar 18, 2009 at 17:25

1

In the code that launches two threads:

```
beginthread (threadA);
sleep(100);
beginthread (threadB);
```

the sleep() waits for 100 ms and then continues. The programmer probably did this in order to give threadA a chance to start up before launching threadB. If you must wait for threadA to be initialized and running before starting threadB, then you need a mechanism that waits for threadA to start, but this is the wrong way to do it.

100 is a magic cookie, chosen arbitrarily, probably accompanying a thought like "it should never take threadA more than 100 ms to start up." Assumptions like this are faulty because you have no way of knowing how long it will take for threadA to start. If the machine is busy or if the implementation of threadA changes it could easily take longer than 100 ms for the thread to launch, run its startup code, and get to it's main loop (if it is that kind of thread).

Instead of sleeping for some arbitrary amount of time, threadA needs to tell the main thread when it is up & running. One common way of doing this is by signaling an event.

Sample code that illustrates how to do this:

```
#include "stdafx.h"
#include <windows.h>
#include <process.h>


struct ThreadParam
{
    HANDLE running_;
    HANDLE die_;
};

DWORD WINAPI threadA(void* pv)
{
    ThreadParam* param = reinterpret_cast<ThreadParam*>(pv);
    if( !param )
        return 1;

    // do some initialization
    //  :   :

    SetEvent(param->running_);
    WaitForSingleObject(param->die_, INFINITE);
    return 0;
}

DWORD WINAPI threadB(void* pv)
{
    ThreadParam* param = reinterpret_cast<ThreadParam*>(pv);
    if( !param )
        return 1;

    // do some initialization
    //  :   :

    SetEvent(param->running_);
    WaitForSingleObject(param->die_, INFINITE);
    return 0;
}

int main(int argc, char** argv)
{
    ThreadParam
        paramA = {CreateEvent(0, 1, 0, 0), CreateEvent(0, 1, 0, 0) },
```

```
        paramB = {CreateEvent(0, 1, 0, 0), CreateEvent(0, 1, 0, 0) };

    DWORD idA = 0, idB = 0;
    // start thread A, wait for it to initialize
    HANDLE a = CreateThread(0, 0, threadA, (void*)&paramA, 0, &idA);
    WaitForSingleObject(paramA.running_, INFINITE);
    // start thread B, wait for it to initi
    HANDLE b = CreateThread(0, 0, threadB, (void*)&paramB, 0, &idB);
    WaitForSingleObject(paramB.running_, INFINITE);
    // tell both threads to die
    SetEvent(paramA.die_);
    SetEvent(paramB.die_);
    CloseHandle(a);
    CloseHandle(b);
    return 0;
}
```

Share Improve this answer Follow

answered Mar 18, 2009 at 17:29

John Dibling

101k ● 32 ● 191 ● 331

0

It's not needed - what output do you get if you omit it?

Share Improve this answer Follow

answered Mar 18, 2009 at 16:19

anon

0

The only thing sleep does is pauses execution on the calling thread for the specified number of milliseconds. It in no way will affect the outcome of any printing you might do.

Share Improve this answer Follow

answered Mar 18, 2009 at 16:20

Jim H.

5,579 ● 1 ● 25 ● 23

0

Sleep can be used to avoid a certain thread/process (yeah, i know they are different things) hogging the processor.

On the other hand, printf is thread safe. Cout is not. That may explain differences in their behaviour.

Share Improve this answer Follow

answered Mar 18, 2009 at 16:21

Tom

45.1k 30 141 171

"printf is thread safe" -- this is possibly false. It is both thread safe and not thread safe. It is entirely implementation dependent, and it's not guaranteed to be safe. It depends on whose implementation of printf you're using. – FreeMemory Mar 18, 2009 at 17:24