timmurphy.org

Software and etc.
stay updated via rss

---

search this site

pthreads in C – a minimal working example

Posted: 4th May 2010 by **Tim** in C
Tags: C, example, multithread, pthreads, threading, threads
65

Pthreads are a simple and effective way of creating a multi-threaded application. This introduction to pthreads shows the basic functionality – executing two tasks in parallel and merging back into a single thread when the work has been

done.

First I'll run through the basics of threading applications with pthreads. Multi-threaded applications allow two or more tasks to be executed concurrently (ie: at the same time). When a thread is created using `pthread_create`, both the original thread and the new thread share the same code base and the same memory – it's just like making two function calls at the same time. Multi-threaded applications come with a whole host of concurrency issues, which will be discussed further in a future post.

All C programs using pthreads need to include the *pthread.h* header file (ie: `#include <pthread.h>`). There are four steps to creating a basic threaded program:

1: Define thread reference variables

The variable type `pthread_t` is a means of referencing threads. There needs to be a `pthread_t` variable in existence for every thread being created. Something like `pthread_t thread0;` will do the trick.

2: Create an entry point for the thread

When creating a thread using pthreads, you need to point it to a function for it to start execution. The function must return `void *` and take a single `void *` argument. For example, if you want the function to take an integer argument, you will need to pass the address of the integer and dereference it later. This may sound complicated but, as is shown below, it's pretty simple. An example function signature would be `void *my_entry_function(void *param);`

3: Create the thread

Once the `pthread_t` variable has been defined and the entry point function created, we can create the thread using `pthread_create`. This method takes four arguments: a pointer to the `pthread_t` variable, any extra attributes (don't worry about this for now – just set it to NULL), a pointer to the function to call (ie: the name of the entry point) and the pointer being passed as the argument to the function. Now there's a lot of pointers in that call, but don't stress – it's not as tricky as it sounds. This call will look something like `pthread_create(&thread0, NULL, my_entry_function, &parameter);`

4: Join everything back up

When the newly-created thread has finished doing it's bits, we need to join everything back up. This is done by the `pthread_join` function which takes two parameters: the `pthread_t` variable used when `pthread_create` was called (not a pointer this time) and a pointer to the return value pointer (don't worry about this for now – just set it to NULL). This call will look something like `pthread_join(thread0, NULL);`

And that's all there is to it. The function used as the thread entry point can call other functions, create variables or do anything any other function can do. It can also use the variables set by the other thread.

When compiling the program, you will also need to add `-lpthread` to the compile command. ie: `gcc program.c -o program -lpthread`

Below is a minimum example of a threaded application. It creates two numbers, x and y, and creates a second thread. The first thread increments y until it has the value of 100, while the second thread increments x until it has the value of 100 at the same time. When this is done, it joins the second thread back with the main program and prints the results. Note how, even though x was changed by the second thread, it has been changed for the main program too!

```
#include <pthread.h>
#include <stdio.h>

/* this function is run by the second thread */
void *inc_x(void *x_void_ptr)
```

```
{

    /* increment x to 100 */
    int *x_ptr = (int *)x_void_ptr;
    while(++(*x_ptr) < 100);

    printf("x increment finished\n");

    /* the function must return something - NULL will do */
    return NULL;

}

int main()
{

    int x = 0, y = 0;

    /* show the initial values of x and y */
    printf("x: %d, y: %d\n", x, y);

    /* this variable is our reference to the second thread */
    pthread_t inc_x_thread;

    /* create a second thread which executes inc_x(&x) */
    if(pthread_create(&inc_x_thread, NULL, inc_x, &x)) {

        fprintf(stderr, "Error creating thread\n");
        return 1;

    }
    /* increment y to 100 in the first thread */
    while(++y < 100);

    printf("y increment finished\n");

    /* wait for the second thread to finish */
    if(pthread_join(inc_x_thread, NULL)) {

        fprintf(stderr, "Error joining thread\n");
        return 2;

    }

    /* show the results - x is now 100 thanks to the second thread */
    printf("x: %d, y: %d\n", x, y);

    return 0;

}
```

This code will print something like:

```
x: 0, y:0
y increment finished
x increment finished
x: 100, y: 100
```

1.  *Nano Thermite 911 False Flag* says:
   February 26, 2016 at 5:47 pm

   This is a WONDERFUL example, but the real problem is that all the INTELLIGENCE is in the “pthread.h” file.

   What is the operation and computer science behind this is what I want to know !!!!!

Kind regards,
Nano Thermite 911 Red-Gray chips

2. *Mohan* says:
March 5, 2016 at 10:46 pm

This example I have tried with `x64` configuration but have problem in pthread_join ?
Can u tell me how to run pthread with x64 build?

3. *[ASK] programming - On window x64 build of example crashing at pthread_cancel and pthread_join | Some Piece of Information* says:
March 5, 2016 at 11:00 pm

[…] : Also I have copied example from pthreads in C â€“ a minimal working example and try to run but having same error in pthread_join […]

4. *prabeen* says:
March 28, 2016 at 11:37 pm

Hi Dears,

Please help me to Find the sum of two integer by adding them using multithreading in Cpp.. and the thread fun should return sum result.

5. *sukesh* says:
May 5, 2016 at 6:38 pm

can we send an array to function instead of just one value?

6. *Jeremy* says:
October 24, 2016 at 1:57 am

You can wrap a pointer to your array and its site in a struct and pass a pointer to it:

struct arr_wrapper {
void *array;
size_t len;
}

7. *sathish* says:
March 22, 2017 at 3:34 pm

ok

8. *scan a barcode, play a sound on mac – siktech.wordpress.com* says:
April 7, 2017 at 8:15 pm

[…] http://timmurphy.org/2010/05/04/pthreads-in-c-a-minimal-working-example/ – create a thread in c […]

9.  *Irene* says:
[July 5, 2017 at 5:55 pm]

When I try to create a thread a get an error: 11 is the returned value. How can I solve it? I copied this code and it didn't work

10.  *Richard wicks* says:
[October 31, 2017 at 5:42 am]

To: Nano Thermite

If that is your real name. If you want to know how an OS works under the hood, work with a simple and well written one first. Linux isn't the least bit simple, but Micrium OS is. There's also eCos, but it's more or less defunct.

Linux is enormously complex but well written. Just consider it magic, and leave it at that. It takes years to understand how pthreads work at the nuts and bolts level unless you set out to just study it exclusively for a few months. Implementations of pthreads are also opaque for a reason – to prevent you specifically from messing with what is under the hood.

I wouldn't even argue pthreads are a particularly good implementation of a thread library, but this is my bias from working with vxWorks, pSOS, Micrium, eCos, and probably a 1/2 dozen other OSes I cannot remember. Pthreads is kind of goofy in a lot of ways.

11. *SPO600 Project Stage 3 | Wrap-up – SPO600* says:
April 20, 2018 at 9:08 am

[…] it so that the program could use multithreading in the program. My previous idea was to use the pthread library and extract parts of the main loop, encapsulate it within a function, and replace the […]

12.  *Wes Peters* says:
April 20, 2018 at 11:51 am

I'll agree with what Richard says about learning an OS, on all levels. If you want to learn how a small, functional OS works, get the Labrosse book on uC/OS. There are ports of that OS that will run on a variety of chips, all the way down to tiny 8-bit devices; it's very good, and it's simple enough to really grasp the concepts.

You can't expect someone to teach you something as complicated as 'how pthreads are implemented' in a blog post. Read a book or something.

13.  *Joel* says:
June 14, 2018 at 12:10 am

I have created a working example, but it does not provide any benefit to the runtime, quite the opposite.

If I increment 4 variables inside a loop to 1 billion, the program uses 7% of my CPU and takes 12 seconds.

If I run 3 threads to increment three separate variables, and then increment the fourth variable in the foreground, the program uses 21% of my CPU, but takes 150 seconds.

I expected the latter to take 12 seconds as well. Does anybody think they know why the threading is taking so much longer?

Thank youi
Joel

14.  *Antonio F.* says:
[September 20, 2018 at 10:11 am]

Great example, thanks!

15.  *raya* says:
[October 11, 2018 at 4:27 pm]

not working

« Older Comments

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

newer posts
older posts

Social Connections

Blog Roll

- LaTeX Bold and Italic Font
- LaTeX Horizontal Lines
- LaTeX Line Spacing
- LaTeX Margins
- LaTeX Page Numbering
- LaTeX Therefore Symbol
- My First LaTeX Document
- My Second LaTeX Document
- My Third LaTeX Document

Recent Comments

- Damian Reloaded: Should be: struct
- Kristofer: What you are doing i
- raya: not working
- Antonio F.: Great example, thank
- Sid: Appreciate the tip.
- Lennart: Just like debugger;
- niclas: Thanks, works perfec
- CaoThuy: Thanks your informat
- Y: Very useful! Exactly
- Murali: Thanks. This helped.