How to Terminate a Thread in C Program (pthread_exit Example)

by Himanshu Arora on April 13, 2012

In the part-II (Thread creation and Identification) of the Linux Thread series, we discussed about thread IDs, how to compare two thread IDs and how to create a thread.

In this article we will mainly focus on how a thread is terminated.

Linux Threads Series: part 1, part 2, part 3 (this article).

C Thread Example Program

If we take the same example as discussed in part-II of this series :

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_t tid[2];

void* doSomeThing(void *arg)
{
    unsigned long i = 0;
    pthread_t id = pthread_self();

    if(pthread_equal(id,tid[0]))
    {
```

```
        printf("\n First thread processing\n");
    }
    else
    {
        printf("\n Second thread processing\n");
    }

    for(i=0; i<(0xFFFFFFFF);i++);

    return NULL;
}

int main(void)
{
    int i = 0;
    int err;

    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
        if (err != 0)
            printf("\ncan't create thread :[%s]", strerror(err));
        else
            printf("\n Thread created successfully\n");

        i++;
    }

    sleep(5);
    return 0;
}
```

Did you observe the ‘sleep()’ function being used? Did you get a question about why sleep() is being used? Well if you did then you are at the correct place to get the answer and if you did not then also its going to be a good read ahead.

If I remove the sleep() function from the code above and then try to compile and run it, I see the following output :

```
$ ./threads
Thread created successfully
First thread processing
Thread created successfully
```

But if I run it with sleep() enabled then I see the output as :

```
$ ./threads
Thread created successfully
First thread processing
Thread created successfully
Second thread processing
```

So we see that the log ‘Second thread processing’ is missing in case we remove the sleep() function.

So, why does this happen? Well, this happened because just before the second thread is about to be scheduled, the parent thread (from which the two threads were created) completed its execution. This means that the default thread in which the main() function was running got completed and hence the process terminated as main() returned.

Thread Termination

As already discussed above that each program starts with at least one thread which is the thread in which main() function is executed. So maximum lifetime of every thread executing in the program is that of the main thread. So, if we want that the main thread should wait until all the other threads are finished then there is a function pthread_join().

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **rval_ptr);
```

The function above makes sure that its parent thread does not terminate until it is done. This function is called from within the parent thread and the first argument is the thread ID of the thread to wait on and the second argument is the return value of the thread on which we want to the parent thread to wait. If we are not interested in the return value then we can set this pointer to be NULL.

If we classify on a broader level, then we see that a thread can terminate in three ways :

1. If the thread returns from its start routine.
2. If it is canceled by some other thread. The function used here is pthread_cancel().
3. If its calls pthread_exit() function from within itself.

The focus here would be on pthread_exit(). Its prototype is as follows :

```
#include <pthread.h>
void pthread_exit(void *rval_ptr);
```

So we see that this function accepts only one argument, which is the return from the thread that calls this function. This return value is accessed by the parent thread which is waiting for this thread to terminate. The return value of the thread terminated by pthread_exit() function is accessible in the second argument of the pthread_join which just explained above.

C Thread Termination Example

Lets take an example where we use the above discussed functions :

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_t tid[2];
int ret1,ret2;

void* doSomeThing(void *arg)
{
    unsigned long i = 0;
    pthread_t id = pthread_self();

    for(i=0; i<(0xFFFFFFFF);i++);

    if(pthread_equal(id,tid[0]))
    {
        printf("\n First thread processing done\n");
        ret1  = 100;
        pthread_exit(&ret1);
    }
    else
    {
        printf("\n Second thread processing done\n");
        ret2  = 200;
        pthread_exit(&ret2);
    }

    return NULL;
}

int main(void)
{
    int i = 0;
    int err;
    int *ptr[2];

    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
        if (err != 0)
            printf("\ncan't create thread :[%s]", strerror(err));
        else
            printf("\n Thread created successfully\n");

        i++;
    }

    pthread_join(tid[0], (void**)&(ptr[0]));
    pthread_join(tid[1], (void**)&(ptr[1]));

    printf("\n return value from first thread is [%d]\n", *ptr[0]);
    printf("\n return value from second thread is [%d]\n", *ptr[1]);

    return 0;
}
```

In the code above :

- We created two threads using pthread_create()
- The start function for both the threads is same ie doSomeThing()
- The threads exit from the start function using the pthread_exit() function with a return value.

- In the main function after the threads are created, the pthread_join() functions are called to wait for the two threads to complete.
- Once both the threads are complete, their return value is accessed by the second argument in the pthread_join() call.

The output of the above code comes out as :

```
$ ./threads
Thread created successfully
Thread created successfully
First thread processing done
Second thread processing done
return value from first thread is [100]
return value from second thread is [200]
```

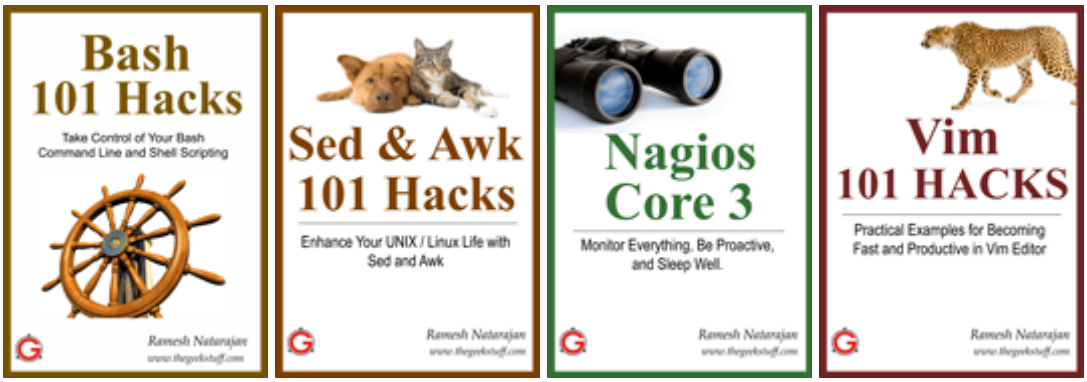
So we see that both the threads execute completely and their return value is accessed in the main function.

 > Add your comment

If you enjoyed this article, you might also like..

1. 50 Linux Sysadmin Tutorials
2. 50 Most Frequently Used Linux Commands (With Examples)
3. Top 25 Best Linux Performance Monitoring and Debugging Tools
4. Mommy, I found it! – 15 Practical Linux Find Command Examples
5. Linux 101 Hacks 2nd Edition eBook Free

- Awk Introduction – 7 Awk Print Examples
- Advanced Sed Substitution Examples
- 8 Essential Vim Editor Navigation Fundamentals
- 25 Most Frequently Used Linux IPTables Rules Examples
- Turbocharge PuTTY with 12 Powerful Add-Ons



Tagged as: pthread_cancel Example, pthread_create Example, pthread_exit Example, pthread_join Example

{ 18 comments… add one }

- Tom July 6, 2012, 7:48 am

  great article series, really helped me understand threading in a concise way.

  Link
- joy July 30, 2012, 12:43 am

really a good work .It clarified some of my doubts on threads.thanks..

Link

- shilpa October 16, 2012, 5:13 am

  Very good article.

  Link
- vidya.v. October 22, 2012, 12:08 am

  Very nice article for beginners.Thanx.

  Link
- prashant October 26, 2012, 12:10 am

  Nice article . .. Thanx

  Link
- Boshra February 11, 2013, 5:36 pm

  Useful. thanks a lot!

  Link
- Kala February 23, 2013, 1:34 am

  Good article. Clearly explains concept .thanks

  Link
- Bijuli March 6, 2013, 4:21 am

  Very good article.Well understanding.

  Link
- shakti singh March 21, 2013, 10:20 am

  after copying above programme in gcc compiler it shows
  undefined reference to ‘pthread_create’
  undefined reference to ‘pthread_join’

  Link
- Vivek March 22, 2013, 12:36 pm

  In your example on Thread Termination, you have the following lines of code in main()

  pthread_join(tid[0], (void**)&(ptr[0])); // line 1
  pthread_join(tid[1], (void**)&(ptr[1])); // line 2

  So does the main thread start waiting as soon as it executes line 1?
  If that is the case, what would happen if:

  1. Main thread executes line 1 and starts waiting for the first thread to finish.
  2. In the meantime, second thread finishes (before the first thread) and returns some value.
  3. Main thread has not yet executed line 2 (which means main thread has not received the return value from second thread)
  4. First thread finishes and main thread is then able to execute line 2

In that case will we lose (drop) the return value of the 2nd thread which has already exited before main called pthread_join on it?

– vivek

Link

- Amir August 8, 2013, 12:39 pm

  Well written and well explained. (needs some typo correction though)
  it has nice examples too

  Link
- Younis August 31, 2013, 3:41 am

  This explanation was something I needed to do my assignment.
  I got here and it was the starting point…
  Thank you very much.

  Link
- Manish April 16, 2015, 1:36 pm

  WE need to create the threads as JOINABLE, for pthread_join to work.

  Link
- Prabhu August 27, 2015, 1:11 am

  I am not verified. But Good Explanation….Thanks dude..

  Link
- chitra G M April 6, 2016, 5:29 pm

  Beautiful explanation.Thanks a lot.

  Link
- chitra G M April 6, 2016, 5:41 pm

  very nice explanation.

  Link
- Lamma February 11, 2017, 8:53 pm

  Very great article thank you so much

  Link
- aserrao April 18, 2017, 12:58 pm

  Very nice articles! Thanks

  Link

Leave a Comment

Name

Email

Website

Comment

Submit

Notify me of followup comments via e-mail

Next post: 20 Awesome Google Chrome Browser Tips and Tricks

Previous post: EMC Navisphere – How to Create Raid Group and LUN on CLARiiON

RSS | Email | Twitter | Facebook | Google+

Custom Search Search

EBOOKS

- Free Linux 101 Hacks 2nd Edition eBook - Practical Examples to Build a Strong Foundation in Linux
- Bash 101 Hacks eBook - Take Control of Your Bash Command Line and Shell Scripting
- Sed and Awk 101 Hacks eBook - Enhance Your UNIX / Linux Life with Sed and Awk
- Vim 101 Hacks eBook - Practical Examples for Becoming Fast and Productive in Vim Editor
- Nagios Core 3 eBook - Monitor Everything, Be Proactive, and Sleep Well

The Geek Stuff
17,391 likes

Like Page Share

Be the first of your friends to like this

POPULAR POSTS

- 15 Essential Accessories for Your Nikon or Canon DSLR Camera
- 12 Amazing and Essential Linux Books To Enrich Your Brain and Library
- 50 UNIX / Linux Sysadmin Tutorials
- 50 Most Frequently Used UNIX / Linux Commands (With Examples)
- How To Be Productive and Get Things Done Using GTD
- 30 Things To Do When you are Bored and have a Computer
- Linux Directory Structure (File System Structure) Explained with Examples
- Linux Crontab: 15 Awesome Cron Job Examples
- Get a Grip on the Grep! – 15 Practical Grep Command Examples
- Unix LS Command: 15 Practical Examples
- 15 Examples To Master Linux Command Line History
- Top 10 Open Source Bug Tracking System
- Vi and Vim Macro Tutorial: How To Record and Play
- Mommy, I found it! -- 15 Practical Linux Find Command Examples
- 15 Awesome Gmail Tips and Tricks
- 15 Awesome Google Search Tips and Tricks
- RAID 0, RAID 1, RAID 5, RAID 10 Explained with Diagrams
- Can You Top This? 15 Practical Linux Top Command Examples
- Top 5 Best System Monitoring Tools
- Top 5 Best Linux OS Distributions
- How To Monitor Remote Linux Host using Nagios 3.0
- Awk Introduction Tutorial – 7 Awk Print Examples
- How to Backup Linux? 15 rsync Command Examples
- The Ultimate Wget Download Guide With 15 Awesome Examples
- Top 5 Best Linux Text Editors
- Packet Analyzer: 15 TCPDUMP Command Examples
- The Ultimate Bash Array Tutorial with 15 Examples
- 3 Steps to Perform SSH Login Without Password Using ssh-keygen & ssh-copy-id
- Unix Sed Tutorial: Advanced Sed Substitution Examples
- UNIX / Linux: 10 Netstat Command Examples
- The Ultimate Guide for Creating Strong Passwords
- 6 Steps to Secure Your Home Wireless Network
- Turbocharge PuTTY with 12 Powerful Add-Ons

CATEGORIES

- Linux Tutorials

- Vim Editor
- Sed Scripting
- Awk Scripting
- Bash Shell Scripting
- Nagios Monitoring
- OpenSSH
- IPTables Firewall
- Apache Web Server
- MySQL Database
- Perl Programming
- Google Tutorials
- Ubuntu Tutorials
- PostgreSQL DB
- Hello World Examples
- C Programming
- C++ Programming
- DELL Server Tutorials
- Oracle Database
- VMware Tutorials

About The Geek Stuff

 My name is **Ramesh Natarajan**. I will be posting instruction guides, how-to, troubleshooting tips and tricks on Linux, database, hardware, security and web. My focus is to write articles that will either teach you or help you resolve a problem. Read more about Ramesh Natarajan and the blog.

Contact Us

Email Me : Use this Contact Form to get in touch me with your comments, questions or suggestions about this site. You can also simply drop me a line to say hello!.

Follow us on Google+

Follow us on Twitter

Become a fan on Facebook

Support Us

Support this blog by purchasing one of my ebooks.

Bash 101 Hacks eBook

Sed and Awk 101 Hacks eBook