How to Create Threads in Linux (With a C Example Program)

by Himanshu Arora on April 6, 2012

In the part I of the Linux Threads series, we discussed various aspects related to threads in Linux.

In this article we will focus on how a thread is created and identified. We will also present a working C program example that will explain how to do basic threaded programming.

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

Thread Identification

Just as a process is identified through a process ID, a thread is identified by a thread ID. But interestingly, the similarity between the two ends here.

- A process ID is unique across the system where as a thread ID is unique only in context of a single process.
- A process ID is an integer value but the thread ID is not necessarily an integer value. It could well be a structure
- A process ID can be printed very easily while a thread ID is not easy to print.

The above points give an idea about the difference between a process ID and thread ID.

Thread ID is represented by the type 'pthread_t'. As we already discussed that in most of the cases this type is a structure, so there has to be a function that can compare two thread IDs.

```
#include <pthread.h>
int pthread_equal(pthread_t tid1, pthread_t tid2);
```

So as you can see that the above function takes two thread IDs and returns nonzero value if both the thread IDs are equal or else it returns zero.

Another case may arise when a thread would want to know its own thread ID. For this case the following function provides the desired service.

```
#include <pthread.h>
pthread_t pthread_self(void);
```

So we see that the function 'pthread_self()' is used by a thread for printing its own thread ID.

Now, one would ask about the case where the above two function would be required. Suppose there is a case where a link list contains data for different threads. Every node in the list contains a thread ID and the corresponding data. Now whenever a thread tries to fetch its data from linked list, it first gets its own ID by calling 'pthread_self()' and then it calls the 'pthread_equal()' on every node to see if the node contains data for it or not.

An example of the generic case discussed above would be the one in which a master thread gets the jobs to be processed and then it pushes them into a link list. Now individual worker threads parse the linked list and extract the job assigned to them.

Thread Creation

Normally when a program starts up and becomes a process, it starts with a default thread. So we can say that every process has at least one thread of control. A process can create extra threads using the following function :

```
#include <pthread.h>
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void *(*start_rtn)(void), void *restrict arg)
```

The above function requires four arguments, lets first discuss a bit on them :

- The first argument is a pthread_t type address. Once the function is called successfully, the variable whose address is passed as first argument will hold the thread ID of the newly created thread.
- The second argument may contain certain attributes which we want the new thread to contain. It could be priority etc.
- The third argument is a function pointer. This is something to keep in mind that each thread starts with a function and that functions address is passed here as the third argument so that the kernel knows which function to start the thread from.
- As the function (whose address is passed in the third argument above) may accept some arguments also so we can pass these arguments in form of a pointer to a void type. Now, why a void type was chosen? This was because if a function accepts more than one argument then this pointer could be a pointer to a structure that may contain these arguments.

A Practical Thread Example

Following is the example code where we tried to use all the three functions discussed above.

```
#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;
}
```

So what this code does is :

- It uses the pthread_create() function to create two threads
- The starting function for both the threads is kept same.
- Inside the function 'doSomeThing()', the thread uses pthread_self() and pthread_equal() functions to identify whether the executing thread is the first one or the second one as created.
- Also, Inside the same function 'doSomeThing()' a for loop is run so as to simulate some time consuming work.

Now, when the above code is run, following was the output :

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

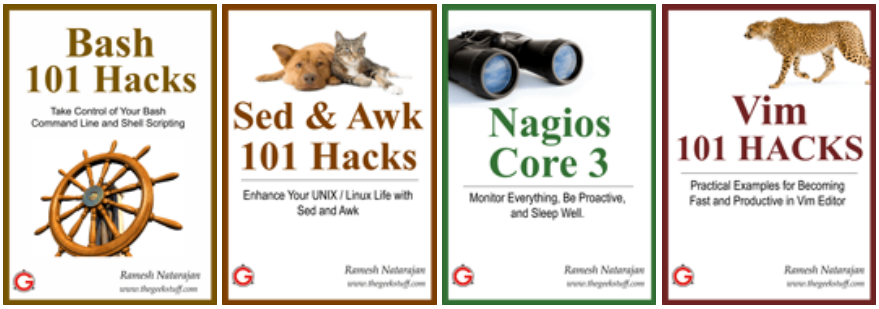
As seen in the output, first thread is created and it starts processing, then the second thread is created and then it starts processing. Well one point to be noted here is that the order of execution of threads is not always fixed. It depends on the OS scheduling algorithm.

Note: The whole explanation in this article is done on Posix threads. As can be comprehended from the type, the pthread_t type stands for POSIX threads. If an application wants to test whether POSIX threads are supported or not, then the application can use the macro _POSIX_THREADS for compile time test. To compile a code containing calls to posix APIs, please use the compile option '-pthread'.

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_create Example, pthread_self

{ 18 comments… add one }

- steve April 6, 2012, 9:46 am

  Hello,

  Great article but missing a few things.

  I noticed you are return NULL from your worker/background thread. Should be pthread_exit(NULL) NULL since you are not returning any arguments. This should clean up a few things before the thread exits.

  Also you have a sleep() function at the end of main. I guess if the thread had a long task that takes more than 5 seconds then main would finish and as the main thread created the worker thread that would get killed as well, which could result it UB and cause you program to crash.

  You should just pthread_join(thread_id) for main to wait for the worker thread to finish.

  Or you can use pthread_exit(NULL) at the end of main. So that the main thread would finish without killing the worker thread. The worker thread would finish without any problem. However, I would recommend pthread_join() so you can exit your program in a controlled way.

  Hope this helps,

  Link
- Himanshu April 7, 2012, 3:34 am

  @steve
  Hi Steve, thanks for you valuable comments. The two points that you brought into notice will be covered in the next part of the ongoing series of Linux threads. I intentionally did not focus on ‘pthread_exit()’ and the ‘sleep()’ function in this part as this part was intended to just give a small overview of how threads come alive. The subsequent parts will peel of the layers and will focus on deeper aspects including ‘pthread_exit()’ and why sleep() should not be used etc..

  Link
- SF November 5, 2012, 4:52 am

  Also for clarity, it should be added to link against library -lpthread for the example given to build correctly.

  Link
- Chokho November 5, 2012, 11:48 am

  Another Example that might help

  ```
  #include
  #include
  #include

  void * func(void * param){
  printf(“Inside Thread function\n”);
  char ** var = (char **)param;
  printf(“The passed argument is \”%s\”\n”, *var);
  int * i;
  i=(int *)malloc(4);
  *i=100;
  ```

```
pthread_exit((void*)i);
}

int main(int argc, char * argv[]){ //Argument to be passed as command line argument
int * retval;
retval = (int *)malloc(4);
pthread_t tid;
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_create(&tid,&attr,func,&argv[1]);
pthread_join(tid, (void **)&retval);
printf("retval is %d\n", *retval);
}
```

Link

- Dominic Pritham July 22, 2013, 6:46 pm

  Hi, you mentioned that thread ID can be a structure. And you also mentioned that thread IDs can be compared. In C, can structures be compared? Isn't it a syntax error if you try to compare two structures?

  Thank you,
  Dominic Pritham

  Link
- tweety October 14, 2013, 11:00 am

  can u tell me when will dosomething function be invocked??

  Link
- saikumar November 23, 2014, 11:05 pm

  hello sir,
  i have a doubt in my program
  HOW TO KILL REMAINING THREADS WHEN WE COME OUT OF ANY SINGLE THREAD ?
  I searched for this but no result! i think u have great knowledge in this area could u help me to solve this

  Link
- Sam November 26, 2014, 5:20 pm

  simple, clear, and helpful. thanks dude.

  Link
- Austin January 2, 2015, 1:00 am

  Hello,
  Great write-up.
  I was a bit confused because I believe:

  void *(*start_rtn)(void) in:

  int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void *(*start_rtn)(void), void *restrict arg)

  may be incorrect because the definition for doSomeThing wouldn't match up.

  void* doSomeThing(void *arg)

  In fact in pthread.h the definition is:

  extern int pthread_create (pthread_t *__restrict __newthread,
  const pthread_attr_t *__restrict __attr,
  void *(*__start_routine) (void *),
  void *__restrict __arg) __THROWNL __nonnull ((1, 3));

  Austin

  Link
- Abdul Ghaffar Malik May 6, 2015, 12:43 pm

When I try to run above code in DEVC++ compiler it produces following errors…
1)- [Linker error] undefined reference to `_imp__pthread_self'
2)- [Linker error] undefined reference to `_imp__pthread_equal'
3)- [Linker error] undefined reference to `_imp__pthread_create'
4)- ld returned 1 exit status

Link

- Anonymous July 11, 2015, 12:07 pm

  I believe pthread isn't implemented on Windows, that's why you are getting linker error with DEVC++.

  Link
- Amit November 23, 2015, 10:07 pm

  Nice article… Thanks for understanding the requirement from the beginners point of view.

  Link
- Stefan Yohansson January 2, 2016, 7:07 pm

  @Abdul Ghaffar Malik, pthread win32 – here

  Link
- Anonymous February 24, 2016, 5:07 am

  /tmp/ccGoGiGt.o: In function `main':
  pthread.c:(.text+0x55): undefined reference to `pthread_create'
  pthread.c:(.text+0x91): undefined reference to `pthread_join'

  Q.what is solution for this error?

  Link
- David March 5, 2017, 9:29 pm

  @Anonymous

  You are not linking phtread library when compiling the program. Do it this way:

  gcc threadprogram.c -o threadprogram -phtread

  Link
- David March 5, 2017, 9:30 pm

  @Anonymous

  You are not linking phtread library when compiling the program. Do it this way:

  gcc threadprogram.c -o threadprogram -pthread

  Link
- David March 5, 2017, 9:31 pm

  @upComment

  You are not linking pthread library when compiling the program. Do it this way:

  gcc threadprogram.c -o threadprogram -pthread

  Hope it helps.

  Link
- sergey April 27, 2017, 7:27 am

  Although you included the pthread header and the compilation phase completed successfully, you still have a problem in the linkage phase.
  You need to link the pthread dynamic library to your executable as it doesn't link automatically.

  using gcc link as follows:

  gcc -o prog …. -lpthread

Link

Leave a Comment

Name

Email

Website

Comment

Submit

Notify me of followup comments via e-mail

Next post: Introduction to Linux IP Routing Fundamentals (Part 1)

Previous post: How to Upgrade Big-IP F5 LTM from 10.1 to 11.1 (or 9.4 to 11.1)

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

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

Vim 101 Hacks eBook

Nagios Core 3 eBook