

# Pthreads, Part 1: Introduction

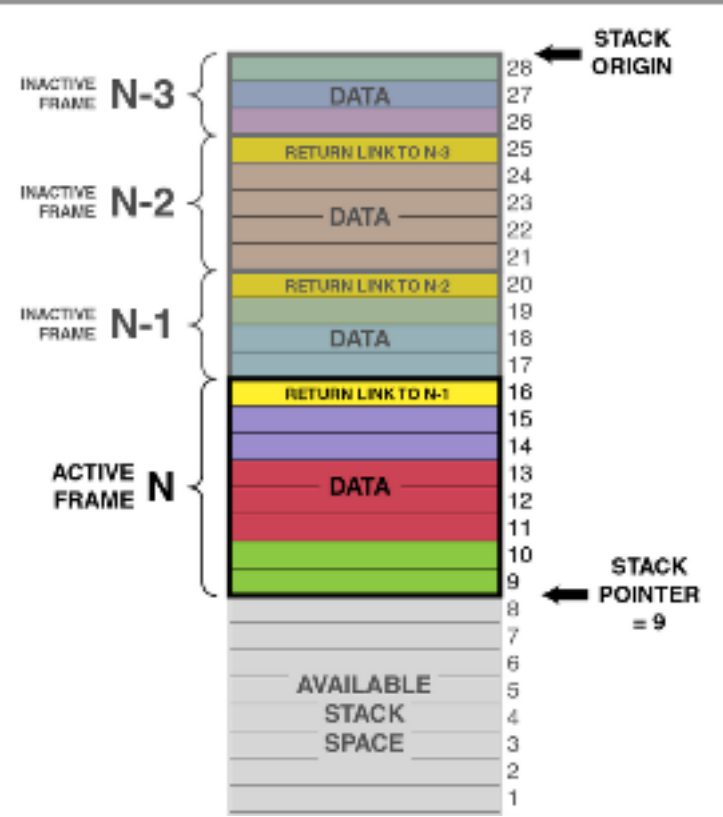
Scott Bigelow edited this page on Mar 10 · 1 revision

## What is a thread?

A thread is short for 'thread-of-execution'. It represents the sequence of instructions that the CPU has (and will) execute. To remember how to return from function calls, and to store the values of automatic variables and parameters a thread uses a stack.

## How does the thread's stack work?

Your main function (and other functions you might call) have automatic variables. We will store them in memory using a stack and keep track of how large the stack is by using a simple pointer (the "stack pointer"). If the thread calls another function, we move our stack pointer down, so that we have more space for parameters and automatic variables. Once it returns from a function, we can move the stack pointer back up to its previous value. We keep a copy of the old stack pointer value - on the stack! This is why returning from a function is very quick - it's easy to 'free' the memory used by automatic variables - we just need to change the stack pointer.



## How many threads can my process have?

You can have more than one thread running inside a process. You get the first thread for free! It runs the code you write inside 'main'. If you need more threads you can call `pthread_create` to create a new thread using the pthread library. You'll need to pass a pointer to a function so that the thread knows where to start.

The threads you create all live inside the same virtual memory because they are part of the same process. Thus they can all see the heap, the global variables and the program code etc. Thus you can have two (or more) CPUs working on your program at the same

Edit

New Page

▼ Pages 51

[Home](#)

[#Example Markdown](#)

[#Informal Glossary](#)

[#Piazza: When And How to Ask For Help](#)

[C Programming, Part 1: Introduction](#)

[C Programming, Part 2: Text Input And Output](#)

[C Programming, Part 3: Common Gotchas](#)

[C Programming, Part 4: Debugging](#)

[Deadlock, Part 1: Resource Allocation Graph](#)

[Deadlock, Part 2: Deadlock Conditions](#)

[File System, Part 1: Introduction](#)

[File System, Part 2: Files are inodes \(everything else is just data...\)](#)


[File System, Part 3: Permissions](#)

[File System, Part 4: Working with directories](#)

[File System, Part 5: Virtual file systems](#)

Show 36 more pages...

Clone this wiki locally

 Clone in Desktop

time and inside the same process. It's up to the operating system to assign the threads to CPUs. If you have more active threads than CPUs then the kernel will assign the thread to a CPU for a short duration (or until it runs out of things to do) and then will automatically switch the CPU to work on another thread. For example, one CPU might be processing the game AI while another thread is computing the graphics output.

# Hello world pthread example

To use pthreads you will need to include `pthread.h` AND you need to compile with `-pthread` (or `-lpthread`) compiler option. This option tells the compiler that your program requires threading support

To create a thread use the function `pthread_create`. This function takes four arguments:

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                  void *(*start_routine) (void *), void *arg);
```

- The first is a pointer to a variable that will hold the id of the newly created thread.
- The second is a pointer to attributes that we can use to tweak and tune some of the advanced features of pthreads.
- The third is a pointer to a function that we want to run
- Fourth is a pointer that will be given to our function

The argument `void *(*start_routine) (void *)` is difficult to read! It means a pointer that takes a `void *` pointer and returns a `void *` pointer. It looks like a function declaration except that the name of the function is wrapped with `(* .... )`

Here's the simplest example:

```
#include <stdio.h>
#include <pthread.h>
// remember to set compilation option -pthread

void *busy(void *ptr) {
    // ptr will point to "Hi"
    puts("Hello World");
    return NULL;
}
int main() {
    pthread_t id;
    pthread_create(&id, NULL, busy, "Hi");
    while (1) {} // Loop forever
}
```

If we want to wait for our thread to finish use `pthread_join`

```
void *result;
pthread_join(id, &result);
```

In the above example, `result` will be `null` because the busy function returned `null`. We need to pass the address-of result because `pthread_join` will be writing into the contents of our pointer.

Legal and Licensing information: Unless otherwise specified, submitted content to the wiki must be original work (including text, java code, and media) and you provide this material under a [Creative Commons License](#). If you are not the copyright holder, please give proper attribution and credit to existing content and ensure that you have license to include the materials.

