Determine the fully qualified hostname of the local machine in C

Content

1 Objective
2 Scenario
3 Method
  3.1 Overview
  3.2 Get the hostname reported by the operating system
  3.3 Canonicalise the hostname using getaddrinfo
4 Alternatives
  4.1 Using uname to get the hostname reported by the operating system
  4.2 Using gethostbyname to canonicalise the hostname
5 Furthe reading

Tested on

Debian (Lenny, Squeeze)
Ubuntu (Lucid, Precise, Trusty)

Objective

To determine the fully qualified hostname of the local machine in C

Scenario

Suppose you wish to print the hostname of the local machine to stdout. This should be the fully qualified hostname if that is obtainable.

Method

Overview

POSIX defines two functions for obtaining the hostname of the local machine: `uname` and `gethostname`. Unfortunately it does not specify whether they should return the unqualified or fully qualified hostname, and there is no consensus as to which of these alternatives is preferable. For this reason, to be sure of obtaining the fully qualified hostname, a two step process is necessary:

1. Get the hostname reported by the operating system
2. Canonicalise the hostname using `getaddrinfo`

The following header files will be used:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/utsname.h>
```

```
#include <netdb.h>
```

Get the hostname reported by the operating system

Neither of the functions provided by POSIX have ideal behaviour, but of the two, `gethostname` is preferable because its drawbacks are more manageable. It must be supplied with a buffer to hold the hostname, and if this is too small then the result is truncated. The difficulty is deciding how large the buffer should be.

It might be possible to obtain an upper limit for the size by calling `sysconf` with an argument of `_SC_HOST_NAME_MAX`, but this facility is not as widely implemented as `gethostname`, and where it does exist it may report that the size is indeterminate. It would be reasonable to use the constant `NI_MAXHOST` where it exists, but the safest method is to use `gethostname` itself to establish the limit dynamically:

1. Guess the required length and allocate a buffer of that size.
2. Attempt to fetch the hostname.
3. Detect whether the hostname was truncated.
4. If it was then increase the size of the buffer and try again.

If the buffer is too small then `gethostname` should truncate the result, and should report whether or not it completed successfully. There are three complications:

- If the result is truncated then it may or may not be terminated.
- Some implementations interpret a truncated result as a success, others as a failure.
- Some implementations write nothing to the buffer if it is too small (which is arguably a form of truncation, but probably not what was intended).

Despite this variation, there is a reasonably safe method to detect truncation:

1. Create a buffer that is one byte longer than will be declared to `gethostname`.
2. Set the last byte of the buffer to zero, ensuring that it is and will remain null-terminated.
3. Call `getnameinfo`.
4. Measure the length of the string in the buffer.

If an error is reported, or if the length is one byte less than the buffer size, then the hostname has definitely been truncated. If the length is three or more bytes less than the buffer size then it has definitely not been truncated. If it is two bytes less then it has not necessarily been truncated, but you should assume that it has been.

For efficency it is best to start with a buffer that is large enough for most hostnames, then increase its size geometrically as required:

```
size_t hostname_len=128;
char* hostname=0;
while (1) {
    // (Re)allocate buffer of length hostname_len.
    char* realloc_hostname=realloc(hostname,hostname_len);
    if (realloc_hostname==0) {
        free(hostname);
        die("failed to get hostname (out of memory)");
    }
    hostname=realloc_hostname;

    // Terminate the buffer.
```

```
    hostname[hostname_len-1]=0;

    // Offer all but the last byte of the buffer to gethostname.
    if (gethostname(hostname,hostname_len-1)==0) {
        size_t count=strlen(hostname);
        if (count<hostname_len-2) {
            // Break from loop if hostname definitely not truncated
            break;
        }
    }

    // Double size of buffer and try again.
    hostname_len*=2;
}
```

The variable `hostname` should now point to a heap-resident buffer containing the hostname as reported by the operating system. This should be deallocated using `free` when it is no longer needed.

Canonicalise the hostname using getaddrinfo

The hostname obtained above may not be the fully qualified hostname, however it can reasonably be expected to resolve, and to be an alias for the fully qualified hostname (if it is not already fully qualified). Assuming that to be the case, the fully qualified hostname can be obtained by canonicalising the hostname returned by the operating system. This can be done using `getaddrinfo` with the `AI_CANONNAME` flag set:

```
struct addrinfo hints={0};
hints.ai_family=AF_UNSPEC;
hints.ai_flags=AI_CANONNAME;

struct addrinfo* res=0;
if (getaddrinfo(hostname,0,&hints,&res)==0) {
    // The hostname was successfully resolved.
    printf("%s\n",res->ai_canonname);
    freeaddrinfo(res);
} else {
    // Not resolved, so fall back to hostname returned by OS.
    printf("%s\n",hostname);
}
```

The value `AF_UNSPEC` for the address family indicates that any type of network address is acceptable. This is necessary because it cannot be safely assumed that all hostnames resolve to an IPv4 address (or to any other single address family).

Failure to resolve the hostname is not a good state of affairs, but nor is it an error as such. Most programs should therefore fall back to the uncanonicalised hostname if they are unable to canonicalise it.

Alternatives

Using uname to get the hostname reported by the operating system

The `uname` function returns the hostname in a fixed-length buffer:

```
struct utsname name;
if (uname(&name)==-1) {
    die("%s",strerror(errno));
}
const char* hostname=name.nodename;
```

POSIX warns that the fixed-length buffer may not be large enough to accommodate the actual hostname. For this reason it is better to use `gethostname` if it is available. That should be the case for systems that conform to POSIX:2001 or later, but historically `gethostname` was a feature of BSD that did not exist in UNIX System V. Programs that need to be maximally portable should have the ability to use either.

As an alternative to using `uname` directly, Gnulib (the GNU portability library) includes a fallback implementation of `gethostname` which obtains the information using `uname`.

Using gethostbyname to canonicalise the hostname

Prior to the introduction of `getaddrinfo`, the function used to resolve hostnames was `gethostbyname`. The main reason for the change to `getaddrinfo` was to provide better support for IPv6, (and in particular, for mixed use of IPv4 and IPv6). Given a choice it is better to use `getaddrinfo` in preference to `gethostbyname`, but on older systems that may not be an option.

```
struct hostent* ent=gethostbyname(hostname);
if (ent) {
    printf("%s",ent->h_name);
} else {
    printf("%s",hostname);
}
```

The result may be overwritten by subsequent calls to `gethostbyname` or `gethostbyaddr`, so if you want to keep any of the information returned then you need to make a copy of it.

Gnulib includes a fallback implementation of `getaddrname` which obtains the information using `gethostbyname`.

Furthe reading

- gethostname, Base Specifications Issue 7, The Open Group, 2008
- uname, Base Specifications Issue 7, The Open Group, 2008
- Host Identification, The GNU C Library Reference Manual, The GNU Project
- Portable gethostname()

Tags: c