My Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

README

#C Program Examples#2

These are some C program examples from my course on systems programming (CS2014), the canonical URL for this is here.

Files in this example:

- README.md - this file in markdown format
- README.html - this file, in HTML format (`'make html'` to update that from .md)
- Makefile - to build the example and HTML (there's a clean target too)
- `rndbytes.c` - a couple of wee utility fuctions to get stuff from `/dev/random`
- `rndbytes.h` - header for those functions
- `rbtest.c` - `main()` that calls functions from `rndbytes.h`

After running `'make'` then these files will be produced (if all goes well):

- README.html - the html version of README.md
- rndbytes.o - the rndbytes object file
- rbtest - the rndbytes test program

A More Structured `rndbytes.c` setup

This iteration of the `rndbytes` example demonstrates a bunch of things that we'll talk about in class:

- Making a header file
- Object files and the build
- Documentation (via doxygen, not sure how relevant, but leads to useful thoughts)
- Coding styles such as `Mozilla's`
- Performance (running `time c-prog-2/rbtest 60000` vs. `time c-prog-1/rndbytes 60000`)

Header file

Here's the source:

```
/*!
 *
```

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

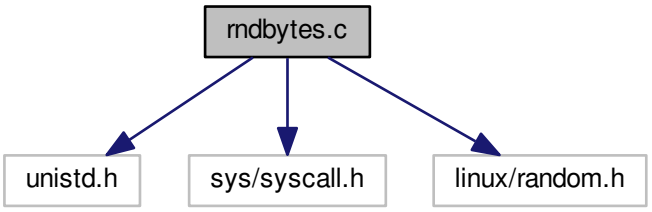
File Documentation

3.1 rndbytes.c File Reference

This is the implementation of the external i/f for the rndbytes example.

```
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/random.h>
```

Include dependency graph for rndbytes.c:



Functions

- unsigned char rndbyte ()

  TODO: check if conditional compile needed as per: https://stackoverflow.com/questions/30800331/getrandom-sy
- int rndbytes (unsigned char *buf, int buflen)

  fill a buffer with random bytes

3.1.1 Detailed Description

This is the implementation of the external i/f for the rndbytes example.

This is part of CS2014 https://down.dsg.cs.tcd.ie/cs2014/examples/c-progs-2/READM↩
E.html

3.1.2 Function Documentation

3.1.2.1 rndbyte()

```
unsigned char rndbyte ( )
```

TODO: check if conditional compile needed as per: https://stackoverflow.com/questions/30800331/getrandom

produce a random byte

3.1.2.2 rndbytes()

```
int rndbytes (
            unsigned char * buf,
            int buflen )
```

fill a buffer with random bytes

Parameters

buf	an allocated buffer of at least the required size
buflen	the number of random bytes to insert into the buffer

Returns

zero for success, nonzero for error

Fill me buffer with randoms.

3.2 rndbytes.h File Reference

This is the external i/f for the rndbytes example.

Functions

- unsigned char rndbyte ()

  produce a random byte
- int rndbytes (unsigned char *buf, int buflen)

  fill a buffer with random bytes

3.2.1 Detailed Description

This is the external i/f for the rndbytes example.

- *
- This is part of CS2014
- https://down.dsg.cs.tcd.ie/cs2014/examples/c-progs-2/README.html */

/*

- Copyright (c) 2017 stephen.farrell@cs.tcd.ie
-
- Permission is hereby granted, free of charge, to any person obtaining a copy
- of this software and associated documentation files (the "Software"), to deal
- in the Software without restriction, including without limitation the rights
- to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
- copies of the Software, and to permit persons to whom the Software is
- furnished to do so, subject to the following conditions:
-
- The above copyright notice and this permission notice shall be included in
- all copies or substantial portions of the Software.
-
- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
- THE SOFTWARE.
- */

/*!

- produce a random byte
  Returns
- the random byte
-
- Get me a random byte from /dev/random
- */ unsigned char rndbyte();

/*!

- fill a buffer with random bytes

Parameters

buf	an allocated buffer of at least the required size

-

Parameters

buflen	the number of random bytes to insert into the buffer

-

  Returns

- zero for success, nonzero for error

-

- Fill me buffer with randoms.

- ∗/ int rndbytes(unsigned char∗ buf,int buflen);

Noteworthy:

- doxygen header @ top and before function prototypes
- function prototypes, one we've seen and one we've not
- memory management of `buf` parameter of `rndbytes`

The Makefile

The significant part of the Makefile is below...

```
all: html rbtest

rbtest: rbtest.c rndbytes.o rndbytes.h

rndbytes.o: rndbytes.c rndbytes.h

doc: rbtest rnd-dox latex/refman.pdf
    doxygen rnd-dox

latex/refman.pdf:
    cd latex;make

rnd-dox:
    doxygen -g rnd-dox

html: README.html

clean:
    @rm -f rbtest rndbytes.o
    @rm -rf latex html

reallyclean: clean
    @rm -f README.html rnd-dox

%.html: %.md
    $(MDCMD) $(MDOPTS) $(@) $(<)
```

Built-in documentation

There are varying opinions as to whether and how to best document your code. Those vary from "don't include any comments" to schemes for generating code from "documentation." [refs needed]

A lot of the "debate" on this topic seems pertty badly justified to me, and more like a whole load of opinion. However, there are some aspects of documentation on which I think a lot of people would agree:

- In most development environments, you will in any case have to follow the local coding style, so you won't get to choose, until you're the one writing the coding style (which takes us beyond this course:-). In other words, usually there's no point in worrying about this as you'll have no choice.
- Adding `Javadoc` style comments to APIs is a fine thing. Those do make it easier to understand an API, and also force you to think a bit more when creating an API, and automatically produced documentation is a fine thing, since it saves you time.
- `usage()` and help options for command line tools are good, as is a man page, if you might want your tool to be adopted by e.g. some Linux distro. If it's just a local tool and not aiming to be part of an open-source distro, then you can probably skip the man page.
- You will inevitably need to leave TODO: and FIXME: breadcrumbs, for yourself or later developers. Those are good things if they help someone to later debug a problem! But it's clearly a bad practice to just leave your code incomplete and think a FIXME: is sufficient.
- Adding comments, but especially keeping comments up to date, takes time, and you probably won't have that much time (or will get bored), so having too many comments does have negative consequences. In the worst case, if code is changed but comments aren't then comments might be misleading.
- There are cases where some fragment of code is just complex or non-obvious and really needs some documentation to explain what's going on. In-line code comments can be a good way to do that, as you will see those when you look at the code, but might not see any other artefact. (Unless comments have been stripped.)
- Making your code as "self-documenting" as you can is good. Choose meaningful names for functions and variables, but it's also ok to just use `foo` or `i`. Do consider what someone reading your code might think, as you write (and re-factor) your code.

The overall goal should be to make your code something that can be understood, fixed or refactored by you or someone else, possibly in many years time.

This is part of CS2014 https://down.dsg.cs.tcd.ie/cs2014/examples/c-progs-2/README.html

3.2.2 Function Documentation

3.2.2.1 rndbyte()

```
unsigned char rndbyte ( )
```

produce a random byte

Returns

the random byte

Get me a random byte from /dev/random

produce a random byte

3.2.2.2 rndbytes()

```
int rndbytes (
            unsigned char * buf,
            int buflen )
```

fill a buffer with random bytes

Parameters

buf	an allocated buffer of at least the required size
buflen	the number of random bytes to insert into the buffer

Returns

zero for success, nonzero for error

Fill me buffer with randoms.

Index