Linux, Pointers and pthreads

Edward Zhang

SOFTENG 370 T1

Hello!

I'm in Part IV, and you probably remember me from SOFTENG 251, SOFTENG 206, and SOFTENG 254

- Ask questions on Piazza instead of emailing me so your classmates can see the answers (also such that Robert can answer questions that I can't, such as specifics regarding what you can and can't do in the assignment)
- If you want to meet, email me first at ezha210@aucklanduni.ac.nz
- These slides will be on Canvas, and any source code demonstrated along with TeX source code for these slides can be found on github.com/encryptededdy

You need a UNIX system

Some ways to get a UNIX system to do this assignment

- Dual Boot Linux
- Run Linux in a Virtual Machine
- Run natively on macOS
  - Probably won’t work for Assignment 2 (no FUSE)
- Run within Windows Subsystem for Linux (WSL)
  - Probably won’t work for Assignment 2 (no FUSE)
- Run within Windows Subsystem for Linux 2 (WSL2)
  - Unreleased, unless you want to run Insider Fast Ring (not recommended)

On Virtual Machines

You can use any distro you want, but you'll probably be able to get more help when googling if you use one of the more popular desktop ones.

- Ubuntu (probably 18.04 LTS)
- Fedora Workstation (my personal preference)
- Debian
- Arch (great wiki, and u use arch btw), Manjaro if you actually want an installer

Hypervisors

Oracle's VirtualBox is the usual free go-to. I personally prefer VMWare Player, feel free to give it a try. Parallels is a good option on macOS, but it's $$$.

Also try Hyper-V on Windows if you have Pro and already have it enabled, as it lets you keep other Windows features on (like Windows Sandbox or Core Isolation). It also supports one-click install of Ubuntu.

Note on Dual Booting

Beware you may be unable to dual-boot on some hardware, such as Surface Devices (drivers are a bit of a pain, especially on the book; check r/surfacelinux for more resources), or the 2019 MacBook Pro (can't even install, T2 chip NVMe storage support broken).

VSCode Remote

You can develop in a Linux environment with a Linux toolchain, while running VSCode from within Windows. This supports WSL. See: https://code.visualstudio.com/docs/remote/wsl

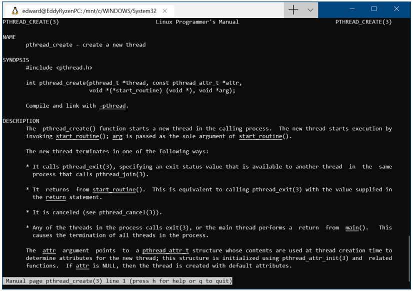
Software to use

- Install gcc (if not part of your distro) using apt/dnf/pacman
- Visual Studio Code is a fine text editor with IntelliSense
- You could also use CLion (JetBrains) if you prefer IntelliJ-like shortcuts and autocomplete, however you will need to create your own CMake file for building. There's no free version, but you can sign up for a JetBrains educational account

Using man to find documentation

Man is a built in documentation tool. In this case, we can check the documentation for pthread_create using...

`$ man pthread_create`



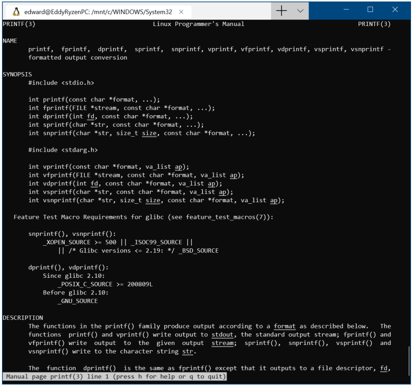
```
PTHREAD_CREATE(3)                  Linux Programmer's Manual                  PTHREAD_CREATE(3)

NAME
       pthread_create - create a new thread

SYNOPSIS
       #include <pthread.h>

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

       Compile and link with -pthread.

DESCRIPTION
       The  pthread_create() function starts a new thread in the calling process.  The new thread starts execution by
       invoking start_routine(); arg is passed as the sole argument of start_routine().

       The new thread terminates in one of the following ways:

       * It calls pthread_exit(3), specifying an exit status value that is available to another thread  in  the  same
         process that calls pthread_join(3).

       * It  returns  from start_routine().  This is equivalent to calling pthread_exit(3) with the value supplied in
         the return statement.

       * It is canceled (see pthread_cancel(3)).

       * Any of the threads in the process calls exit(3), or the main thread performs a  return  from  main().   This
         causes the termination of all threads in the process.

       The  attr  argument  points  to  a pthread_attr_t structure whose contents are used at thread creation time to
       determine attributes for the new thread; this structure is initialized using pthread_attr_init(3) and  related
       functions.  If attr is NULL, then the thread is created with default attributes.

 Manual page pthread_create(3) line 1 (press h for help or q to quit)
```

Finding the correct manpage

What if there are multiple versions of a given function?

```
$ man 3 printf
```

Use 3 to access section 3, which contains the C function version of printf. Without 3 you get the linux command.



```
PRINTF(3)                          Linux Programmer's Manual                         PRINTF(3)

NAME
       printf,  fprintf,  dprintf,  sprintf,  snprintf, vprintf, vfprintf, vdprintf, vsprintf, vsnprintf -
       formatted output conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);

       #include <stdarg.h>

       int vprintf(const char *format, va_list ap);
       int vfprintf(FILE *stream, const char *format, va_list ap);
       int vdprintf(int fd, const char *format, va_list ap);
       int vsprintf(char *str, const char *format, va_list ap);
       int vsnprintf(char *str, size_t size, const char *format, va_list ap);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       snprintf(), vsnprintf():
           _XOPEN_SOURCE >= 500 || _ISOC99_SOURCE ||
               || /* Glibc versions <= 2.19: */ _BSD_SOURCE

       dprintf(), vdprintf():
           Since glibc 2.10:
               _POSIX_C_SOURCE >= 200809L
           Before glibc 2.10:
               _GNU_SOURCE

DESCRIPTION
       The functions in the printf() family produce output according to a format as described below.   The
       functions  printf() and vprintf() write output to stdout, the standard output stream; fprintf() and
       vfprintf() write  output  to  the  given  output  stream;  sprintf(),  snprintf(),  vsprintf()  and
       vsnprintf() write to the character string str.

       The  function  dprintf()  is the same as fprintf() except that it outputs to a file descriptor, fd,
 Manual page printf(3) line 1 (press h for help or q to quit)
```

Defining Pointers

Consider a variable foo. Say we define it as `int foo;`

- `&foo` gives us the address of foo.
- `int *fooPointer` stores a pointer to something of type int. Thus, we could do something like `int *fooPointer = &foo;`

Assignment / Dereferencing

Ok, now we have a pointer to foo that we defined with `int *fooPointer = &foo;`. How can we write to what it's pointing too (foo)?

- You cannot just go `fooPointer = 12`
- We can instead dereference using an asterix and perform a store, such as `*fooPointer = 12`
- We can load the value such as `int bar = *fooPointer;`

Example

```
#include <stdio.h>

int main( int argc, const char* argv[] )
{
    int foo;
    int *fooPointer = &foo;
    *fooPointer = 420;

    printf("%d\n", fooPointer); // Compiler warning
    printf("%d\n", *fooPointer);
    printf("%d\n", foo);

    int bar = *fooPointer;
    bar = 840;

    printf("%d\n", bar);
    printf("%d\n", foo);
}
```