# Asynchronicity

Paul R. Potts

05 Oct 2019

Another long week has passed. I wound up staying up very late on Wednesday. Thursday morning I stayed home and worked on soldering together a prototype board that I intend to send to the Thorlabs software team in China. I don't know if I'll ever be able to say whether the long hours were worth it or not. I am blessed in many ways though. This is the kind of work I would be doing even if I wasn't getting paid for it. Well, sort of — if I wasn't getting paid, I'd be scrambling to find some other way to make money, but I'd still be trying to do this kind of thing in my spare time.

I'm pleased with my recent work on this project; over the past few months, I've designed a schematic, turned it into a circuit board layout, ordered boards, tested them, revised them, had our technician Scott do some of the assembly, and done the rest of the assembly myself. I *can* solder the microchips and surface-mount parts, but it is easier for Scott. For one thing, he has years of experience soldering surface-mount parts, and for another, he has a big stereoscopic microscope to look through. Also, I'm pretty sure that even though he's older than I am, his eyes are better than mine, and he doesn't have the little bit of essential tremor that I've got; it really only bothers me when I'm trying to do very, very fine work like holding a soldering-iron tip to a resistor half the size of a grain of rice, while I'm trying to hold the resistor in place with the tip of a very sharp pair of tweezers.

(And, no, before you ask, the tremor doesn't seem to be related to my caffeine intake, at least not in the way one might expect; it actually gets better when I've had my coffee, and is aggravated by stress).

The latest iteration of the prototype board is black, for the Halloween season. I didn't even know you could get printed circuit boards in black, but apparently you can, and it looks pretty cool!

## Saturday

This morning Grace and I got up and out reasonably early, although we both woke up feeling pretty beat-up from a hard week. We had the kids toast a tray of bagels with butter, and then Grace and I made the short drive to Milan for

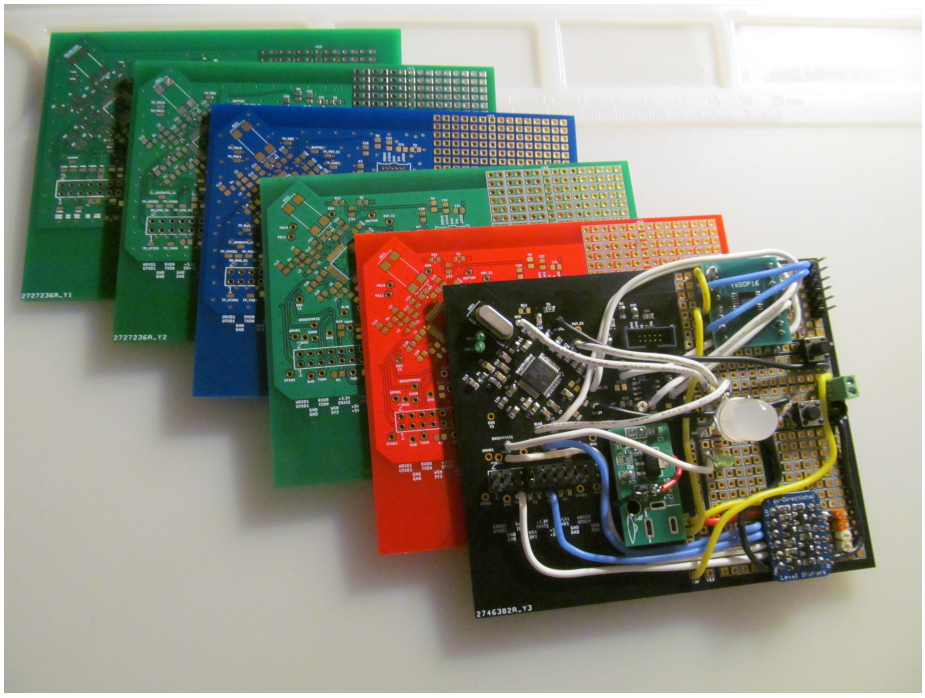

Figure 1: green, blue, red, and black printed circuit boards

coffee and bread at two neighboring local businesses, Milan Coffee Works and Mother Loaf Breads.

We picked up two seasonal sourdough loaves, one spelt with apple chunks and spices, and the other a buckwheat walnut with sassafras and molasses. Both were terrific, although to my taste the buckwheat was the better of the two, and really declicious with butter. I'm a big fan of some of the lesser-used grains, like buckwheat, millet, spelt, Einkorn (an heirloom wheat), and most especially rye, even 100% rye, strong and aromatic and suitable for eating with a shot of icy cold vodka. We also brought home a foccacia made with marinated peppers and onions, and that was terrific too — like a small crispy vegan pizza, combining spicy peppers and sweet caramelized onions.

After we got our bread, we walked through the door to Milan Coffee Works, as they are actually in the same building, and got ourselves a couple of lattes. Grace had chai with oat milk, and I had coffee with oat milk. We didn't stay to drink them there, because we had more errands to run and wanted to get home before the kids could get too crazy, so we took our drinks to go.

Grace wanted to run a quick errand in downtown Milan, so we parked down the street. As I sipped my delicious latte, the caffeine gently massaged my brain, and I started to feel like I was alive again. I noticed that it was a beautiful fall day. This is something to attend to cherish — we might not get that many of them! The week was mostly gray and rainy and the temperature has been bouncing around from the eighties to the forties. I can't figure out what to wear when I leave in the morning. Some nights I go to sleep cold and wake up sweating. Some nights it is just the opposite.

## And Now, In Color!

Sitting in the car, sipping my coffee, my color vision came back. It wasn't really gone — it's not like I was actually seeing in black and white and then started seeing in color, but I was suddenly able to *notice* color again. It *feels* like watching *The Wizard of Oz*. The same thing happened to me back around 1996 when I first went on an antidepressant. As I continued to drink down my latte, a gentle tingling in my scalp told me that I was getting the correct dose of caffeine. (If my heart starts to race unpleasantly, I know I've had a bit too much). When I can manage with coffee, I really prefer to stick to coffee and stay off of prescription SSRIs or related drugs. It is an addictive drug, but the side effects are a lot more pleasant than the side effects from even a mild SSRI like Celexa.

I was on Celexa from last fall through this summer. This helped me to get through some of the worst months of our time with houseguests. This time I wasn't really taking it for depression, *per se*, but anxiety. It helps a lot with that. It takes the edge off my anxiety, but it also takes the edge off my *thinking*. I no longer have my lifelong companion, obsessive-compulsive personality disorder. This also means I'm not as good at my work. And my anxiety is so flattened,

even on a small dose, that I feel indifferent to just about everything; it also tends to flatten out my empathy, and I'm almost indifferent to the kids. It also makes me uninterested in writing, and so on it, I really don't seem like myself, to myself.

So, I wanted to wean myself off Celexa again. In mid-summer I started cutting my pills in half for a couple of months. Then after our houseguests left, I stopped taking them. That didn't go so well. Sometimes the worst side effects come from withdrawing from these drugs. For weeks and weeks I felt like I had a mild flu all the time. I gained ten pounds or so. It's hard to describe the odd sensory effects. Turning my head would make me slightly nauseated.

Those effects have gradually improved, but it might take a long time for them to fade completely. And this is one of the reasons I am hesitant to take these medications, and try to stay off them, unless I'm becoming a danger to myself, or am unable to function in my various capacities.

## Once More Into the Basement

I went down into the basement and did some additional work on my prototype circuit board for work, and then put some time into cleaning up my work area down there, which desperately needed it. When I'm working on a circuit board, bits of insulation and wire fly everywhere. Sometimes I need to clip pins with a wire cutter, and they are made of a hard metal that snaps off and goes flying when cut. So I needed to sweep that all up. I also continued with the project of getting my Mac Pro working well again.

Veronica wanted me to try to restore some old iPad backups so she could find some writing she did a few years ago. That turned into a challenge. There are dozens of backup files and they aren't dated, and we weren't sure which one had the data she was looking for. Restoring the backup worked, after struggling with some error messages that didn't make any sense and, it turns out, didn't describe what was actually wrong. (iTunes kept telling me it couldn't restore the backup because there wasn't enough space available on my computer, but I had over five hundred gigabytes of space available, and it turns out the error message was actually popping up because there wasn't enough space left on the iPad; this error, by the way, is what convinced me a couple of weeks ago that I needed to replace the old system hard drive, which *was* nearly full, with something bigger, and it wasn't even correct. I'm not unhappy to have a bigger drive, but I would have preferred to put off spending the money a little longer).

I had to restore the iPad to a time when I had set up a "restrictions" passcode to keep the kids from constantly changing settings, about five years ago. This is not the same as the passcode needed to log in; when setting up an iPad for the kids to use, I leave off a passcode. This is a code I set up to lock down every setting I could, so that I could try to prevent the kids from screwing with everything. It didn't work. Back when I was commuting between Saginaw and Ann Arbor, I was trying to use two iPads so that I could say goodnight to my

kids over a video call, and maybe read them a story. But they kept messing with the iPad settings, playing with everything they could. They'd sign out of FaceTime, which for some reason was very tricky to undo. They'd rearrange all the icons and hide everything, so no one could find what they were looking for. They would constantly rename the iPad, which is why the backups are very confusing, because I can't tell which iPad goes with which backup file. None of the available restrictions would actually prevent them from doing these things.

They do, however, prevent me from doing a lot of other things, such as changing an e-mail password. And I have long since forgotten the restrictions passcode that I used for a period of a few months on that iPad before wiping it and starting over. Apple products are pretty secure; there's no good way to crack the passcode. Which is why after wasting far too much time on this project, I finally had to ask Veronica to just copy her old story out of the iPad, by writing it in a notebook.

If she manages to hold on to the notebook, it might be readable in thirty-seven years. I've still got some of *my* old notebooks, with journal writing and story ideas, from when I was sixteen, thirty-seven years ago, and they are still perfectly legible, although the paper has yellowed a bit. And I've got some things I wrote on a word processor on my Commodore 64, when I was sixteen.

The computers I was using are long gone. I don't think I have anything that I made on a computer from the age of 14, because at age 14, I didn't have a printer, except a strange one called a TRS-80 Screen Printer, which used silvery rolls of aluminum electrostatic paper. It shot sparks and created a lot of ozone and blackened aluminum dust and spat out a very gritty, low-contrast, slightly wobbly image of whatever was on the screen, which you could tear off like a cash-register receipt. The paper curled up and did not hold up well over time. In 2019 I doubt you could sell such a thing; it probably would be considered a health hazard. I looked in vain on YouTube for a clip of the thing printing, but couldn't find one. The noise it made when I pressed the "Print" button was impressive — it sounded like someone was trying to drill through an aluminum screen door.

## There's Always More Work

I've continued to make slow progress at work. This week I became bogged down for a couple of days by problems with an Windows application, written in C++, using the Qt framework. I needed to get a pretty simple thing working. The Windows application sends packets of data to the microcontroller over a couple of different possible interfaces. You can plug in a USB cable, and send the data using a library call to the driver for the chip on the other end, which converts from USB on one side to old-fashioned serial on the other. Or if you have an adapter from USB to RS-232 or an older computer with an honest-to-God built-in RS-232 port, you can use that.

And so, it's time to talk about *asynchronicity*.

## Asynchronicity

**Warning: this bit is highly technical, so non-geeks might want to skip ahead past this section.**

The USB connection was working perfectly. I could reliably send firmware updates to the microcontroller that way. But the serial side, which uses a C++ class called, unsurprisingly, **QSerialPort**, was not working reliably at all. So I had some debugging to do.

Sending data with this class is pretty straightforward. You just call one method to send a chunk of data. The call doesn't *block* — that is, the method call doesn't wait until all the data has gone out before returning. Computers run *much* faster than serial ports do, so it doesn't really make sense to stop your whole program, or even one thread of it, while the data is being slowly clocked out onto the wire.

That makes things a bit tricky, though, because I actually want my code to stick with a rigid "call-and-response" plan. I want to send the packet, and wait for the answer before I send the next packet. But the call to receive data *also* does not "block," at least not in the usual sense, and again, you don't usually want it to, since it brings your thread of execution to a halt. Depending on how your program is structured, that might also mean that the graphical user interface would gring to a halt, too, and that's bad.

So there are two approaches described in the available sample code, synchronous (blocking) and asynchronous (non-blocking). "Asynchronous" means "not synchronized" with your code; it means that the work happens behind the scenes, and it will be done at some point, and so you have to be ready to handle some kind of notification that tells you when it is done.

Qt is a very old framework, and it came into existence before modern versions of C++ existed, and when computers were much slower, and it wasn't so commonplace for applications with graphical user interfaces to have multiple threads of execution. So the QSerialPort class's way of supporting asynchronous sending and receiving is kind of weird, and not very much like the way most modern libraries do it.

A more modern library would, typically, let me call a function to send data, with parameters that included a pointer to a buffer of data to send, a count of the number of bytes in the buffer, and a *timeout* value, typically in milliseconds. When you make a call like this, you're telling the operating system or library or whatever "hey, here's some data; send it out the serial port. Wait for up to 25 milliseconds. If it takes longer than that, stop and let me know." Then that call would return some sort of error code letting you know whether is succeeded or not. This call would "block," but you wouldn't typically care, because your application would be broken into threads, so that if one thread blocked, the others would keep running, and parts like the graphical user interface would not freeze up. My application is already broken into threads; this code is running in

a thread to do serial communication by either RS-232 or USB, and it is separate from the thread that runs the graphical user interface.

Anyway, that's one way to send data. It's synchronous, but it doesn't really matter, because only one thread waits for the data to go out. Receiving can be done in a similar way: provide the number of bytes you expect to get, a pointer to a buffer big enough to hold those bytes, and a timeout. One thread waits for data to come in, but the others keep going.

The relevant part of Qt's synchronous serial receive sample code looks like this:

```
if (serial.waitForReadyRead(currentWaitTimeout)) {
    QByteArray responseData = serial.readAll();
    while (serial.waitForReadyRead(10))
        responseData += serial.readAll();

    const QString response = QString::fromUtf8(responseData);
    emit this->response(response);
} else {
    emit timeout(tr("Wait read response timeout %1")
                 .arg(QTime::currentTime().toString()));
}
```

If you squint, you can kind of see how that looks like the process I described, except that you can't specify how many bytes you want. In fact this example doesn't guarantee you'll get any particular number of characters. That's fine if you have an ongoing stream of characters that might trickle in at any time, and you just want to log them, or something like that, and then stop if you haven't gotten any for a while. But that's not at all how my application works. I need to get my packet with its fixed number of bytes, because the contents of the packet will tell my code what to do next.

Typically libraries also provide a way to read and write data asynchronously. This is typically done using some kind of *callback* — the library will "call back" to a function in your code, and that function needs a way of signaling to another part of your code that the data was sent, or received, or that it didn't work.

In that case, you might want the code in your class object to wait on a *semaphore*; that's the classic "computer science-y" thing to do. Your code waits for a "resource" to be acquired. A semaphore is a flexible mechanism that can be used to manage any kind of resource. In this case of reading data, the "resource" is the packet of data we are waiting for. The called-back function "gives" the semaphore, and my main thread of execution tries to "take" the semaphore. There's a timeout. This is a classic consumer/producer problem. Qt provides a class called **QSemaphore** that seems like it ought to be perfect for this. So I implemented some code kind of like the example below. (This is not the real code; it's a simplified excerpt to illustrate the concept).

First, we need a method that gets called when bytes are ready to read. This

method then receives all the bytes that are currently available, and appends them to another byte array holding all the bytes I've accumulated so far. When it has enough bytes, it "releases" the semaphore, with a parameter of one, which in our case indicates that one packet of reply data is now available. The "ball" is now in the consumer's court.

```
void SendFileWorker::handleReadyRead()
{
    QByteArray bytes = m_serial_port_p->readAll();

    m_rx_bytes_a.append( bytes );
    if ( m_rx_bytes_a.count() == REPLY_PACKET_BUFFER_SIZE )
    {
        m_reply_sem.release( 1 );
    }
    m_rx_byte_a_mutex.unlock();
}
```

Next, we have the method that waits to "take" the semaphore. When we do this, conceptually this "consumer" code now has taken one "resource," one packet of reply data, and the "producer" code no longer has a resource, until another packet comes in:

```
bool SendFileWorker::receive_rs232_reply( unsigned char * data_p )
{
    bool ret_val = m_reply_sem.tryAcquire( 1, 250 );
    if ( true == ret_val )
    {
        ( void )std::memcpy( data_p, m_rx_bytes_a.constData(),
            REPLY_PACKET_BUFFER_SIZE );
        m_rx_bytes_a.truncate( 0 );
    }

    return ret_val;
}
```

That seems so simple that it couldn't *not* work, right? That's what I thought! But, in fact, it didn't work at all. The code always waited for the semaphore until it timed out. The data was coming into the serial port, but my code was never receiving it.

This is because **QSerialPort** doesn't *really* support asynchronous sending and receiving using callback functions the way they are usually implemented. Qt uses a somewhat antiquated mechanism called "signals and slots." Signals and slots are *very* useful ways to hook up all kinds of messaging between different pieces of code, and in most cases sending an object a signal is *pretty much* equivalent to calling a method of the object; slots are in fact just methods.

But not *just* like calling a method of the object.

The signals that **QSerialPort** provides to indicate that bytes are ready to receive are really sent when an *event loop* detects a condition and triggers the call to the "slot" method. And that only happens if the event loop runs. And the event loop runs *synchronously*. Doing anything that blocks the thread your Qt object is running in, such as waiting to take a semaphore, will bring the event loop mechanism to a halt, and so that code that is waiting to give the semaphore when it is called will *never be executed*.

It might seem like the logical thing to do is to put the **QSerialPort** object on its own thread, but for various reasons this doesn't really solve the problem. Strangely, **QSemaphore** objects aren't really made to send messages **between** Qt threads. The fundamental method that Qt provides for this is… signals and slots. And your thread won't get those signals if it is is sleeping on a **QSemaphore**, or some kind of queue, or any of the "classic" concurrency programming tools that people who have studied programming formally would expect to use for this purpose.

The sample code available to describe how to use **QSerialPort** does a really, really poor job of explaining this. The asynchronous examples are very contrived and simplified and don't show how to do something that ought to be quite simple: sending a packet of serial data, and waiting on a reply. A quick search of the message boards will reveal that a lot of people have trouble using **QSerialPort** to receive data the way they want it to, and that there is a distinct lack of clear explanations of how to do it.

I got it working, finally, by using something called a local event loop, and it works great. I hook up my signals and slots like so:

```
connect( m_serial_port_p,
         SIGNAL( errorOccurred( QSerialPort::SerialPortError ) ),
         this,
         SLOT( handleError( QSerialPort::SerialPortError )   )  );
connect( &m_rs232_rx_timer,
         SIGNAL( timeout()                                   ),
         &m_rs232_rx_event_loop,
         SLOT( quit()                                        )  );
connect( m_serial_port_p,
         SIGNAL( readyRead()                                 ),
         &m_rs232_rx_event_loop,
         SLOT( quit()                                        )  );
```

My **QSerialPort** object's **errorOccurred** signal, which includes an error parameter, is now hooked to my object's handleError slot, my **QTimer** object **timeout** signal is hooked up to my local event loop's **quit** slot, and my QSerialPort object's **readyRead** signal is *also* hooked up to my local event loop's **quit** slot. I have to use a timer, because that **readyRead** signal is vague; it just means "at least one byte is available for reading." And I have to keep track of my own timeout condition. I also have to handle retries until I either have

the number of bytes I expect, an overflow condition, or I've run out of retries. This is quite a bit uglier and more complicated than just making a single call to wait for a certain number of bytes or a timeout, but it works with perfect reliability now. My receive function now looks something like this:

```
bool SendFileWorker::receive_rs232_reply( unsigned char * data_p )
{
    bool done = false;
    bool overrun = false;
    int tries = 0;
    int max_tries = 5;
    QByteArray packet;

    m_rs232_rx_timer.start( 50 );

    do
    {
        m_rs232_rx_event_loop.exec();

        if ( m_rs232_rx_timer.isActive() )
        {
            /*
                If the event loop exited and the timer is still
                running, we must have gotten the readyRead() signal.
            */
            packet += m_serial_port_p->readAll();

            int len = packet.length();
            if ( len > REPLY_PACKET_BUFFER_SIZE )
            {
                overrun = true;
            }
            else if ( len == REPLY_PACKET_BUFFER_SIZE )
            {
                ( void )std::memcpy( data_p, packet.constData(),
                    REPLY_PACKET_BUFFER_SIZE );
                done = true;
            }
        }
        else
        {
            tries += 1;
        }

    } while ( ( false == overrun ) &&
              ( false == done ) &&
```

```
            ( tries <= max_tries ) );

    return done;

}
```

I can do something similar using the synchronous methods, using **WaitForReadyRead()** instead of using a slot in my own code, but it isn't much simpler. **WaitForReadyRead()** must be allowing my main thread's event loop to run. I tried to take a look at the implementation, to see if it gave me any insight, but it calls another method, and then another, and then some sort of implementor class, using the "pointer to implementor" idiom, also known as "pImpl," and then that implementor calls another method which was defined as a macro, which is a pretty ugly and primitive thing to do in a framework written in C++. Searching for *that* implementation crashed my text editor's "find in files" function as it tried to search through almost 250,000 files of Qt source code.

There's a big difference between "theoretically, you can learn a lot from reading the source code!" to actually being able to read and understand the source code of a project like this, which contains hundreds, if not thousands, of developer-years of work, and at this point, probably a non-zero number of developer-careers, too.

Sometimes you **can** learn a great deal from reading a framework's source code. The PowerPlant framework was written by one very smart guy and it was incredibly readable. I really miss using a framework that was so simple and clear. But Qt is not nearly as easy to understand. And waht I am trying to do isn't really very complex or unusual, so the code to do it shouldn't need to be over-complicated.

Anyway.

I will probably write an blog post about this, because when I figure out things that ought to be well-documented, but isn't, I like to write blog posts about them. I have benefited enormously from other people's blogs explaing things that *they* have figured out, so it only seems right to give back. And in fact I still occasionally get "thank you" comments on blog posts I wrote years ago.

### It's Time to Stroke My Long White Beard and Complain about Kids Today

I first did event-driven programming around 1985, when I was learning how to program the original Macintosh. Event-driven programming has a lot of advantages on small and slow systems. But even on small microprocessors, like the SAM4S2 chip I'm programming, which has only 64K of RAM and a clock speed that is only a bit faster than the original Macintosh, I am accustomed to using tiny operating systems like FreeRTOS that allow me to run multiple tasks and communicate between them very easily, with seamphores and queues that pretty much just work, with no surprises. The old event queue designs were a

hack that allowed very slow chips to behave *almost* like they were running fully multi-threaded operating systems and applications; they aren't *really* all that useful when the chips are fast and it is easy to do things in more "computer science-y" ways.

It's actually really, really easy to write event-driven code using primitive multi-threading constructs like semaphores and queues; FreeRTOS does a brilliant job in showing just how this sort of thing can be done in a portable but very efficient way.

I've used a lot of different C++ frameworks over the years: TurboVision, THINK Class Library, the Microsoft Foundation Class Library, PowerPlant, the Object Windows Library, and others I have no doubt forgotten; I've also used a number of frameworks written for languages *other* than C++, like Dylan, NewtonScript, and Java.

Qt has outlasted pretty much all of the frameworks I mentioned. One reason is that it solves a lot of hard code portability problems. But this "clash of civilizations" — of different programming paradigms, really — at the heart of Qt is really making me wonder if there isn't a better framework out there.

Could I find a framework for Windows programming that is more modern, more consistent, and simpler, and not just bigger and more "modern?" (It would be nice if it was cross-platform; that used to be one of Qt's big selling points, but it has become less important to me now that pretty much everyone we are targeting with our products has access to a PC running Windows, and there isn't a good business case to be made for writing a Macintosh version).

I don't think such a thing exists, unfortunately.

**Non-geeks, it's safe to come back out now!**

## What I'm Reading This Week

I recently picked up a couple of notable books, both recently released in trade paperback editions, and both biographies, more or less.

### ***The Good Neighbor*** **by Maxwell King**

I've been reading this one when I'm on the treadmill. It's fascinating to read how Rogers was the over-protected child of a very wealthy family, heirs to various industries in Pennsylvania. Young Rogers suffered from asthma, likely due to the terrible air pollution in the region, pollution generated largely by the businesses owned and operated by his family. The irony of the over-protected son of incredibly privileged parents stuck inside all summer in front of early, costly air-conditioning units, suffering because of the very same environmental degradation which gave him that privilege, is not lost on the author.

If you read this with a generally left-wing perspective, it becomes a great document about how class infiltrates everything. I'm not very far along in it

yet. Young Fred Rogers is a sort of musical prodigy, playing the Steinway grand piano his grandmother bought him when he was only about ten years old. It seems like a tremendously indulgent gift, but apparently that piano inspired him his whole life long. So the question shouldn't be *why* young Rogers deserved a wonderful instrument like that, when he clearly made full use of it. The question should be why all kids don't have access to things that will help inspire and instruct them.

I need to be careful reading this book. Over the decades I've become quite comfortable with my contempt for the wealthy. But Rogers, at least so far, I read as a tremendously sympathetic character. As a child I had a lot of the same painful intraversion and difficulty interacting with my peers. It's hard to stay mad a at him, even though he had a Steinway grand piano and I had a broken-down Yamaha acoustic that my mom picked up for me at a garage sale. It was a bad and cheap guitar, and the guitar repair tech at the local music store couldn't figure out how to straighten the neck to make it easier to play. But if you want to learn something badly enough, you'll rip your fingers to shreds to do it.

Right now Rogers is just heading off to college. I'll have more of a book report when I've read more. I can't say that I really love the writing in this book; it does the job, but so far it feels just a little bit flat. Maybe it will get better in later chapters.

***Astounding: John W. Campbell, Isaac Asimov, Robert A. Heinlein, L. Ron Hubbard, and the Golden Age of Science Fiction* by Alec Nevala-Lee**

This is sort of a biography, but it's not just a biography of one person. It interleaves biographies of several of the most important figures in the rise of pulp science fiction, especially magazines such as *Astounding Science Fiction*. I grew up reading Asimov and Heinlein. I know Campbell as the author of a few influential stories, most notably the novella "Who Goes There?" That story may be most famous these days for the 1992 John Carpenter film adaptation, but it has influenced a lot of other work; for example, the X-Files episode "Ice," which is a very straight-up homage to the story. Campbell is best known as the editor who shaped and influenced the direction of the genre in these early days, known now in retrospect as the genre's "golden age."

The real appeal of this book is the honest and unflinching portrayal of Campbell and the other larger-than-life figures that shaped the genre. Nevala-Lee does not sugar-coat their sexism and misogyny, while at the same time, the author hasn't showed up just to "cancel" these brilliant and flawed people. Campbell was a bad student and a liar. At one point he wrote, in one of his overly-inflated bits of autobiographical writing in *Astounding*, that he was working with automobiles. This left readers imagining that he was probably working on a top-secret project to design an atomic car, or something like that. Nope — he was living in

desperate poverty, and working as a car salesman.

His self-portrait, like Hubbard's, was a big lie that, somehow, in the minds of a generation of readers, became "truthy." He and Hubbard both broke just about every moral code of the time, in their interactions with women, while at the same time somehow imagining themselves to be the ultra-competent, ultra-moral, ultra-masculine heroes of their stories. Nevala-Lee doesn't really accuse his subjects; he lets their life stories convict them, or vindicate them, as the reader sees fit. And he does not hesitate to talk about some of their (fictional) stories, too, and how so many of them were, honestly, just plain bad, along with those few that really were "golden."

## This Week's Thing: Chris Lester and *The Metamor City Podcast*

Back in the early days of podcasting, in the summer of 2006, I was turning William Hope Hodgson's novel *The Boats of the "Glen Carrig"* into an audiobook podcast. I was just barely getting things up and running on a technical level, recording between about 1 and 3 a.m. each night using a battered and patched-together Macintosh PowerBook G4, struggling with bad audio, and listening to other podcasts that were out there. I don't remember exactly when I first heard Christ Lester's *Metamor City Podcast*, but it was probably in 2007. I listened to a few of them, and decided that his urban fantasy setting didn't really fit my tastes, when tended towards the hard science fiction of writers like Greg Egan and vintage "weird tales." And so I forgot about it. Back then I didn't have the space on my beat-up laptop to keep old episodes of all the podcasts I was listening to, I deleted them, and I forgot a about *The Metamor City Podcast*.

For a long time.

A few months ago it popped into my head again, as these things sometimes do, and I became determined to find that urban fantasy podcast I listened to a bit way back when. But I couldn't remember the name of the show! It's surprisingly hard to search for early podcasts now, because the searches have all been twisted by the gravitational pull of big podcasts that now have media organizations and money behind them. But eventually I found it and was a bit startled to find out that it is still going, sort of.

Back in the day Chris Lester was recording stories that were set in the Metamor City "shared story universe." After warming up with a few stories, he continued with a full novel, called *Making the Cut*. And then did something incredibly impressive, given that podcasts didn't actually make any money and everyone's labor on the project was volunteer work — he adapted the novel into a full-cast radio drama.

*Making the Cut* is a bit overly romantic and soap-opera-ish in places, and it seems at times like the plot is wandering away from the author and going in directions he didn't really plan for very well, and the full cast is of mixed ability.

But despite all that, when it is good, it is *very* good — really moving and impressive.

I should mention that Lester's urban fantasy world really isn't for everyone. It's a world of humans, vampires, elves, demons, werewolves, and other monsters, and there's a lot of magic involved. In *Making the Cut* a lot of the plot centers around polyamory and characters that change gender, and while it generally isn't pornographic *per se*, at least in my judgment, because the story is really about the characters and their relationships, it is quite often erotic, and plays with a lot of sexy-monster tropes. Your judgment may differ; some people are offended by anything that features sexually explicit talk or acts at all. If that describes you, you won't enjoy Chris Lester's work.

So over the last few months I've listened to the early stories, and then the whole full-cast novel, and then *more* stories, and then another novel, called *Things Unseen*, which didn't get the full-cast treatment but which is, in terms of structure and plotting, quite a bit better than *Making the Cut*, and then *even more* stories. And there's another novel in progress now, called *The Lost and the Least*. I haven't started listening to it yet, but I have every reason to believe that Lester's writing has continued to get better.

Over the course of over a decade of podcasts, one can watch (well, actually, listen to) something pretty amazing happening. One can watch Lester go from a skilled amateur to a skilled and *experienced* semi-professional fantasy writer ("semi-professional" only because at least in the episodes I'm listening to, from August of 2017, he's still got a day job).

The original *Metamor City Podcast* petered out after a few years as Lester moved and changed jobs, but he started up a weekly podcast called *The Raven and the Writing Desk* in the same feed, and that one continues to this day. One of the most interesting things about that podcast is that in each episode, after reading a chapter, or part of a chapter, of a work in development, Lester talks about how his writing has been going for the week. He's brutally honest, freely admitting when things have been going badly and he hasn't managed to get enough butt-in-chair time, and also celebrating when he's been able to get a lot of work done.

It's really inspiring. Lester also uses the feed to post interviews with other writers on the craft and business of writing, and is also himself sometimes interviewed. Those interviews have gone from drunken carousing between groups of friends in the early days to to very focused and insightful sessions about what it is really like to try to make a go of it as a writer with a day job. Lester now has a number of books in print, in snazzy editions with professional-looking covers.

If you'd like a taste of his work, which isn't all centered around Metamor City, I would recommend listening to a couple of his short stories and novellas. There's a fascinating novella called *Divide by Zero* which starts in *The Raven and the Writing Desk* episode 81, from December 16th, 2016. You might also like the

story “Last Sunset at the Golden Gate,” episode 42, from March 8th, 2016. (Again, both contain some erotic elements that some people will want to avoid).

There’s a *lot* of content in *The Metamor City Podcast* feed — over 300 episodes, and it’s still very much an active and ongoing project. The *project* he’s been engaged in is truly inspiring, and it makes me think wistfully about roads not taken, but maybe also about side streets that haven’t yet been closed off. I used to write fiction, and some of it was well-regarded, although I haven’t felt the same need to write fiction over the years as I’ve felt to write non-fiction. But I’m not ready to say that I’ll never write fiction again!

As Chris Lester says, “And that’s this week’s story. Keep it on the bright side!”

## About This Newsletter