The Night Watch

JAMES MICKENS

James Mickens is a researcher in the Distributed Systems group at Microsoft's Redmond lab. His current research focuses on web applications, with an emphasis on the design of JavaScript frameworks that allow developers to diagnose and fix bugs in widely deployed web applications. James also works on fast, scalable storage systems for datacenters. James received his PhD in computer science from the University of Michigan, and a bachelor's degree in computer science from Georgia Tech. mickens@microsoft.com

As a highly trained academic researcher, I spend a lot of time trying to advance the frontiers of human knowledge. However, as someone who was born in the South, I secretly believe that true progress is a fantasy, and that I need to prepare for the end times, and for the chickens coming home to roost, and fast zombies, and slow zombies, and the polite zombies who say "sir" and "ma'am" but then try to eat your brain to acquire your skills. When the revolution comes, I need to be prepared; thus, in the quiet moments, when I'm not producing incredible scientific breakthroughs, I think about what I'll do when the weather forecast inevitably becomes RIVERS OF BLOOD ALL DAY EVERY DAY. The main thing that I ponder is who will be in my gang, because the likelihood of post-apocalyptic survival is directly related to the size and quality of your rag-tag group of associates. There are some obvious people who I'll need to recruit: a locksmith (to open doors); a demolitions expert (for when the locksmith has run out of ideas); and a person who can procure, train, and then throw snakes at my enemies (because, in a world without hope, snake throwing is a reasonable way to resolve disputes). All of these people will play a role in my ultimate success as a dystopian warlord philosopher. However, the most important person in my gang will be a systems programmer. A person who can debug a device driver or a distributed system is a person who can be trusted in a Hobbesian nightmare of breathtaking scope; a systems programmer has seen the terrors of the world and understood the intrinsic horror of existence. The systems programmer has written drivers for buggy devices whose firmware was implemented by a drunken child or a sober goldfish. The systems programmer has traced a network problem across eight machines, three time zones, and a brief diversion into Amish country, where the problem was transmitted in the front left hoof of a mule named Deliverance. The systems programmer has read the kernel source, to better understand the deep ways of the universe, and the systems programmer has seen the comment in the scheduler that says "DOES THIS WORK LOL," and the systems programmer has wept instead of LOLed, and the systems programmer has submitted a kernel patch to restore balance to The Force and fix the priority inversion that was causing MySQL to hang. A systems programmer will know what to do when society breaks down, because the systems programmer already lives in a world without law.

The Night Watch

Listen: I'm not saying that other kinds of computer people are useless. I believe (but cannot prove) that PHP developers have souls. I think it's great that database people keep trying to improve select-from-where, even though the only queries that cannot be expressed using select-from-where are inappropriate limericks from "The Canterbury Tales." In some way that I don't yet understand, I'm glad that theorists are investigating the equivalence between five-dimensional Turing machines and Edward Scissorhands. In most situations, GUI designers should not be forced to fight each other with tridents and nets as I yell "THERE ARE NO MODAL DIALOGS IN SPARTA." I am like the Statue of Liberty: I accept everyone, even the wretched and the huddled and people who enjoy Haskell. But when things get tough, I need mission-critical people; I need a person who can wear night-vision goggles and descend from a helicopter on ropes and do classified things to protect my freedom while country music plays in the background. A systems person can do that. I can realistically give a kernel hacker a nickname like "Diamondback" or "Zeus Hammer." In contrast, no one has ever said, "These semi-transparent icons are really semi-transparent! IS THIS THE WORK OF ZEUS HAMMER?"

I picked that last example at random. You must believe me when I say that I have the utmost respect for HCI people. However, when HCI people debug their code, it's like an art show or a meeting of the United Nations. There are tea breaks and witticisms exchanged in French; wearing a non-functional scarf is optional, but encouraged. When HCI code doesn't work, the problem can be resolved using grand theories that relate form and perception to your deeply personal feelings about ovals. There will be rich debates about the socioeconomic implications of Helvetica Light, and at some point, you will have to decide whether serifs are daring statements of modernity, or tools of hegemonic oppression that implicitly support feudalism and illiteracy. Is pinching-and-dragging less elegant than circling-and-lightly-caressing? These urgent mysteries will not solve themselves. And yet, after a long day of debugging HCI code, there is always hope, and there is no true anger; even if you fear that your drop-down list should be a radio button, the drop-down list will suffice until tomorrow, when the sun will rise, glorious and vibrant, and inspire you to combine scroll bars and left-clicking in poignant ways that you will commemorate in a sonnet when you return from your local farmer's market.

This is not the world of the systems hacker. When you debug a distributed system or an OS kernel, you do it Texas-style. You gather some mean, stoic people, people who have seen things die, and you get some primitive tools, like a compass and a rucksack and a stick that's pointed on one end, and you walk into the wilderness and you look for trouble, possibly while using chewing tobacco. As a systems hacker, you must be prepared to do savage things, unspeakable things, to kill runaway threads with your bare hands, to write directly to network ports using telnet and an old copy of an RFC that you found in the Vatican. When you debug systems code, there are no high-level debates about font choices and the best kind of turquoise, because this is the Old Testament, an angry and monochromatic world, and it doesn't matter whether your Arial is Bold or Condensed when people are covered in boils and pestilence and Egyptian pharaoh oppression. HCI people discover bugs by receiving a concerned email from their therapist. Systems people discover bugs by waking up and discovering that their first-born children are missing and "ETIMEDOUT " has been written in blood on the wall.

What is despair? I have known it—hear my song. Despair is when you're debugging a kernel driver and you look at a memory dump and you see that a pointer has a value of 7. THERE IS NO HARDWARE ARCHITECTURE THAT IS ALIGNED ON 7. Furthermore, 7 IS TOO SMALL AND ONLY EVIL CODE WOULD TRY TO ACCESS SMALL NUMBER MEMORY. Misaligned, small-number memory accesses have stolen decades from my life. The only things worse than misaligned, small-number memory accesses are accesses with aligned buffer pointers, but impossibly large buffer lengths. Nothing ruins a Friday at 5 P.M. faster than taking one last pass through the log file and discovering a word-aligned buffer address, but a buffer length of NUMBER OF ELECTRONS IN THE UNIVERSE. This is a sorrow that lingers, because a 2^{893} byte read is the only thing that both Republicans and Democrats agree is wrong. It's like, maybe Medicare is a good idea, maybe not, but there's no way to justify reading everything that ever existed a jillion times into a mega-jillion sized array. This constant war on happiness is what non-systems people do not understand about the systems world. I mean, when a machine learning algorithm mistakenly identifies a cat as an elephant, *this is actually hilarious*. You can print a picture of a cat wearing an elephant costume and add an ironic caption that will entertain people who have middling intellects, and you can hand out copies of the photo at work and rejoice in the fact that everything is still fundamentally okay. There is nothing funny to print when you have a misaligned memory access, because your machine is dead and there are no printers in the spirit world. An impossibly large buffer error is even worse, because these errors often linger in the background, quietly overwriting your state with evil; if a misaligned memory access is like a criminal burning down your house in a fail-stop manner, an impossibly large buffer error is like a criminal who breaks into your house, sprinkles sand atop random bedsheets and toothbrushes, and then waits for you to slowly discover that your world has been tainted by madness. Indeed, the common discovery mode for an impossibly large buffer error is that your program seems to

be working fine, and then it tries to display a string that should say "Hello world," but instead it prints "#a[5]:3!" or another syntactically correct Perl script, and you're like WHAT THE HOW THE, and then you realize that your prodigal memory accesses have been stomping around the heap like the Incredible Hulk when asked to write an essay entitled "Smashing Considered Harmful."

You might ask, "Why would someone write code in a grotesque language that exposes raw memory addresses? Why not use a modern language with garbage collection and functional programming and free massages after lunch?" Here's the answer: Pointers are real. They're what the hardware understands. Somebody has to deal with them. You can't just place a LISP book on top of an x86 chip and hope that the hardware learns about lambda calculus by osmosis. Denying the existence of pointers is like living in ancient Greece and denying the existence of Krackens and then being confused about why none of your ships ever make it to Morocco, or Ur-Morocco, or whatever Morocco was called back then. Pointers are like Krackens—real, living things that must be dealt with so that polite society can exist. Make no mistake, I don't *want* to write systems software in a language like C++. Similar to the Necronomicon, a C++ source code file is a wicked, obscure document that's filled with cryptic incantations and forbidden knowledge. When it's 3 A.M., and you've been debugging for 12 hours, and you encounter a virtual static friend protected volatile templated function pointer, you want to go into hibernation and awake as a werewolf and then find the people who wrote the C++ standard and bring ruin to the things that they love. The C++ STL, with its dyslexia-inducing syntax blizzard of colons and angle brackets, guarantees that if you try to declare any reasonable data structure, your first seven attempts will result in compiler errors of Wagnerian fierceness:

```
Syntax error: unmatched thing in thing from std::nonstd::__
map<_Cyrillic, _$$$dollars>const basic_string< epic_
mystery,mongoose_traits &lt; char>, __default_alloc_<casual_
Fridays = maybe>>
```

One time I tried to create a list<map<int>>, and my syntax errors caused the dead to walk among the living. Such things are clearly unfortunate. Thus, I fully support high-level languages in which pointers are hidden and types are strong and the declaration of data structures does not require you to solve a syntactical puzzle generated by a malevolent extraterrestrial species. That being said, if you find yourself drinking a martini and writing programs in garbage-collected, object-oriented Esperanto, be aware that the only reason that the Esperanto runtime works is because there are systems people who have exchanged any hope of losing their virginity for the exciting opportunity to think about hex numbers and their relationships with the operating system, the hardware, and ancient blood rituals that Bjarne Stroustrup performed at Stonehenge.

Perhaps the worst thing about being a systems person is that other, non-systems people think that they understand the daily tragedies that compose your life. For example, a few weeks ago, I was debugging a new network file system that my research group created. The bug was inside a kernel-mode component, so my machines were crashing in spectacular and vindictive ways. After a few days of manually rebooting servers, I had transformed into a shambling, broken man, kind of like a computer scientist version of Saddam Hussein when he was pulled from his bunker, all scraggly beard and dead eyes and florid, nonsensical ramblings about semi-imagined enemies. As I paced the hallways, muttering Nixonian rants about my code, one of my colleagues from the HCI group asked me what my problem was. I described the bug, which involved concurrent threads and corrupted state and asynchronous message delivery across multiple machines, and my coworker said, "Yeah, that sounds bad. Have you checked the log files for errors?" I said, "Indeed, I would do that if I hadn't broken *every component that a logging system needs to log data*. I have a network file system, and I have broken the network, and I have broken the file system, and my machines crash when I make eye contact with them. I HAVE NO TOOLS BECAUSE I'VE DESTROYED MY TOOLS WITH MY TOOLS. My only logging option is to hire monks to transcribe the subjective experience of watching my machines die as I weep tears of blood." My coworker, in an earnest attempt to sympathize, recounted one of his personal debugging stories, a story that essentially involved an addition operation that had been mistakenly replaced with a multiplication operation. I listened to this story, and I said, "Look, I get it. Multiplication is not addition. This has been known for years. However, multiplication and addition are at least related. Multiplication is like addition, but with more addition. Multiplication is a grown-up pterodactyl, and addition is a baby pterodactyl. Thus, in your debugging story, your code is wayward, but it basically has the right idea. In contrast, there is no family-friendly GRE analogy that relates what my code should do, and what it is actually doing. I had the modest goal of translating a file read into a network operation, and now my machines have tuberculosis and orifice containment issues. Do you see the difference between our lives? When you asked a girl to the prom, you discovered that her father was a cop. When I asked a girl to the prom, I DISCOVERED THAT HER FATHER WAS STALIN."

In conclusion, I'm not saying that everyone should be a systems hacker. GUIs are useful. Spell-checkers are useful. I'm glad that people are working on new kinds of bouncing icons because they believe that humanity has solved cancer and homelessness and now lives in a consequence-free world

of immersive sprites. That's exciting, and I wish that I could join those people in the 27th century. But I live here, and I live now, and in my neighborhood, *people are dying in the streets*. It's like, French is a great idea, but nobody is going to invent French if they're constantly being attacked by bears. Do you see? SYSTEMS HACKERS SOLVE THE BEAR MENACE. Only through the constant vigilance of my people do you get the freedom to think about croissants and subtle puns involving the true father of Louis XIV. So, if you see me wandering the halls, trying to explain synchronization bugs to confused monks, rest assured that every day, in every way, it gets a little better. For you, not me. I'll always be furious at the number 7, but such is the hero's journey.