# Preface

## To Everyone

Welcome to this book! We hope you'll enjoy reading it as much as we enjoyed writing it. The book is called **Operating Systems: Three Easy Pieces**, and the title is obviously an homage to one of the greatest sets of lecture notes ever created, by one Richard Feynman on the topic of Physics [F96]. While this book will undoubtedly fall short of the high standard set by that famous physicist, perhaps it will be good enough for you in your quest to understand what operating systems (and more generally, systems) are all about.

The three easy pieces refer to the three major thematic elements the book is organized around: **virtualization**, **concurrency**, and **persistence**. In discussing these concepts, we'll end up discussing most of the important things an operating system does; hopefully, you'll also have some fun along the way. Learning new things is fun, right? At least, it should be.

Each major concept is divided into a set of chapters, most of which present a particular problem and then show how to solve it. The chapters are short, and try (as best as possible) to reference the source material where the ideas really came from. One of our goals in writing this book is to make the paths of history as clear as possible, as we think that helps a student understand what is, what was, and what will be more clearly. In this case, seeing how the sausage was made is nearly as important as understanding what the sausage is good for[1].

There are a couple devices we use throughout the book which are probably worth introducing here. The first is the **crux** of the problem. Anytime we are trying to solve a problem, we first try to state what the most important issue is; such a **crux of the problem** is explicitly called out in the text, and hopefully solved via the techniques, algorithms, and ideas presented in the rest of the text.

In many places, we'll explain how a system works by showing its behavior over time. These **timelines** are at the essence of understanding; if you know what happens, for example, when a process page faults, you are on your way to truly understanding how virtual memory operates. If you comprehend what takes place when a journaling file system writes a block to disk, you have taken the first steps towards mastery of storage systems.

There are also numerous **asides** and **tips** throughout the text, adding a little color to the mainline presentation. Asides tend to discuss something relevant (but perhaps not essential) to the main text; tips tend to be general lessons that can be

[1] Hint: eating! Or if you're a vegetarian, running away from.

applied to systems you build. An index at the end of the book lists all of these tips and asides (as well as cruces, the odd plural of crux) for your convenience.

We use one of the oldest didactic methods, the **dialogue**, throughout the book, as a way of presenting some of the material in a different light. These are used to introduce the major thematic concepts (in a peachy way, as we will see), as well as to review material every now and then. They are also a chance to write in a more humorous style. Whether you find them useful, or humorous, well, that's another matter entirely.

At the beginning of each major section, we'll first present an **abstraction** that an operating system provides, and then work in subsequent chapters on the mechanisms, policies, and other support needed to provide the abstraction. Abstractions are fundamental to all aspects of Computer Science, so it is perhaps no surprise that they are also essential in operating systems.

Throughout the chapters, we try to use **real code** (not **pseudocode**) where possible, so for virtually all examples, you should be able to type them up yourself and run them. Running real code on real systems is the best way to learn about operating systems, so we encourage you to do so when you can.

In various parts of the text, we have sprinkled in a few **homeworks** to ensure that you are understanding what is going on. Many of these homeworks are little **simulations** of pieces of the operating system; you should download the homeworks, and run them to quiz yourself. The homework simulators have the following feature: by giving them a different random seed, you can generate a virtually infinite set of problems; the simulators can also be told to solve the problems for you. Thus, you can test and re-test yourself until you have achieved a good level of understanding.

The most important addendum to this book is a set of **projects** in which you learn about how real systems work by designing, implementing, and testing your own code. All projects (as well as the code examples, mentioned above) are in the **C programming language** [KR88]; C is a simple and powerful language that underlies most operating systems, and thus worth adding to your tool-chest of languages. Two types of projects are available (see the online appendix for ideas). The first are **systems programming** projects; these projects are great for those who are new to C and UNIX and want to learn how to do low-level C programming. The second type are based on a real operating system kernel developed at MIT called xv6 [CK+08]; these projects are great for students that already have some C and want to get their hands dirty inside the OS. At Wisconsin, we've run the course in three different ways: either all systems programming, all xv6 programming, or a mix of both.

## To Educators

If you are an instructor or professor who wishes to use this book, please feel free to do so. As you may have noticed, they are free and available on-line from the following web page:

`http://www.ostep.org`

You can also purchase a printed copy from `lulu.com`. Look for it on the web page above.

The (current) proper citation for the book is as follows:

**Operating Systems: Three Easy Pieces**
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
Arpaci-Dusseau Books
March, 2015 (Version 0.90)
`http://www.ostep.org`

The course divides fairly well across a 15-week semester, in which you can cover most of the topics within at a reasonable level of depth. Cramming the course into a 10-week quarter probably requires dropping some detail from each of the pieces. There are also a few chapters on virtual machine monitors, which we usually squeeze in sometime during the semester, either right at end of the large section on virtualization, or near the end as an aside.

One slightly unusual aspect of the book is that concurrency, a topic at the front of many OS books, is pushed off herein until the student has built an understanding of virtualization of the CPU and of memory. In our experience in teaching this course for nearly 15 years, students have a hard time understanding how the concurrency problem arises, or why they are trying to solve it, if they don't yet understand what an address space is, what a process is, or why context switches can occur at arbitrary points in time. Once they do understand these concepts, however, introducing the notion of threads and the problems that arise due to them becomes rather easy, or at least, easier.

As much as is possible, we use a chalkboard (or whiteboard) to deliver a lecture. On these more conceptual days, we come to class with a few major ideas and examples in mind and use the board to present them. Handouts are useful to give the students concrete problems to solve based on the material. On more practical days, we simply plug a laptop into the projector and show real code; this style works particularly well for concurrency lectures as well as for any discussion sections where you show students code that is relevant for their projects. We don't generally use slides to present material, but have now made a set available for those who prefer that style of presentation.

If you'd like a copy of any of these materials, please drop us an email. We have already shared them with many others around the world.

One last request: if you use the free online chapters, please just **link** to them, instead of making a local copy. This helps us track usage (over 1 million chapters downloaded in the past few years!) and also ensures students get the latest and greatest version.

## To Students

If you are a student reading this book, thank you! It is an honor for us to provide some material to help you in your pursuit of knowledge about operating systems. We both think back fondly towards some textbooks of our undergraduate days (e.g., Hennessy and Patterson [HP90], the classic book on computer architecture) and hope this book will become one of those positive memories for you.

You may have noticed this book is free and available online[2]. There is one major reason for this: textbooks are generally too expensive. This book, we hope, is the first of a new wave of free materials to help those in pursuit of their education, regardless of which part of the world they come from or how much they are willing to spend for a book. Failing that, it is one free book, which is better than none.

We also hope, where possible, to point you to the original sources of much of the material in the book: the great papers and persons who have shaped the field of operating systems over the years. Ideas are not pulled out of the air; they come from smart and hard-working people (including numerous Turing-award winners[3]), and thus we should strive to celebrate those ideas and people where possible. In doing so, we hopefully can better understand the revolutions that have taken place, instead of writing texts as if those thoughts have always been present [K62]. Further, perhaps such references will encourage you to dig deeper on your own; reading the famous papers of our field is certainly one of the best ways to learn.

[2] A digression here: "free" in the way we use it here does not mean open source, and it does not mean the book is not copyrighted with the usual protections – it is! What it means is that you can download the chapters and use them to learn about operating systems. Why not an open-source book, just like Linux is an open-source kernel? Well, we believe it is important for a book to have a single voice throughout, and have worked hard to provide such a voice. When you're reading it, the book should kind of feel like a dialogue with the person explaining something to you. Hence, our approach.

[3] The Turing Award is the highest award in Computer Science; it is like the Nobel Prize, except that you have never heard of it.

## Acknowledgments

This section will contain thanks to those who helped us put the book together. The important thing for now: **your name could go here!** But, you have to help. So send us some feedback and help debug this book. And you could be famous! Or, at least, have your name in some book.

The people who have helped so far include: Abhirami Senthilkumaran*, Adam Drescher* (WUSTL), Adam Eggum, Aditya Venkataraman, Adriana Iamnitchi and class (USF), Ahmed Fikri*, Ajaykrishna Raghavan, Akiel Khan, Alex Wyler, Ali Razeen (Duke), AmirBehzad Eslami, Anand Mundada, Andrew Valencik (Saint Mary's), Angela Demke Brown (Toronto), B. Brahmananda Reddy (Minnesota), Bala Subrahmanyam Kambala, Benita Bose, Biswajit Mazumder (Clemson), Bobby Jack, Björn Lindberg, Brennan Payne, Brian Gorman, Brian Kroth, Caleb Sumner (Southern Adventist), Cara Lauritzen, Charlotte Kissinger, Chien-Chung Shen (Delaware)*, Christoph Jaeger, Cody Hanson, Dan Soendergaard (U. Aarhus), David Hanle (Grinnell), David Hartman, Deepika Muthukumar, Dheeraj Shetty (North Carolina State), Dorian Arnold (New Mexico), Dustin Metzler, Dustin Passofaro, Eduardo Stelmaszczyk, Emad Sadeghi, Emily Jacobson, Emmett Witchel (Texas), Erik Turk, Ernst Biersack (France), Finn Kuusisto*, Glen Granzow (College of Idaho), Guilherme Baptista, Hamid Reza Ghasemi, Hao Chen, Henry Abbey, Hrishikesh Amur, Huanchen Zhang*, Huseyin Sular, Hugo Diaz, Itai Hass (Toronto), Jake Gillberg, Jakob Olandt, James Perry (U. Michigan-Dearborn)*, Jan Reineke (Universität des Saarlandes), Jay Lim, Jerod Weinman (Grinnell), Jiao Dong (Rutgers), Jingxin Li, Joe Jean (NYU), Joel Kuntz (Saint Mary's), Joel Sommers (Colgate), John Brady (Grinnell), Jonathan Perry (MIT), Jun He, Karl Wallinger, Kartik Singhal, Kaushik Kannan, Kevin Liu*, Lei Tian (U. Nebraska-Lincoln), Leslie Schultz, Liang Yin, Lihao Wang, Martha Ferris, Masashi Kishikawa (Sony), Matt Reichoff, Matty Williams, Meng Huang, Michael Walfish (NYU), Mike Griepentrog, Ming Chen (Stonybrook), Mohammed Alali (Delaware), Murugan Kandaswamy, Natasha Eilbert, Nathan Dipiazza, Nathan Sullivan, Neeraj Badlani (N.C. State), Nelson Gomez, Nghia Huynh (Texas), Nick Weinandt, Patricio Jara, Perry Kivolowitz, Radford Smith, Riccardo Mutschlechner, Ripudaman Singh, Robert Ordòñez and class (Southern Adventist), Rohan Das (Toronto)*, Rohan Pasalkar (Minnesota), Ross Aiken, Ruslan Kiselev, Ryland Herrick, Samer Al-Kiswany, Sandeep Ummadi (Minnesota), Satish Chebrolu (NetApp), Satyanarayana Shanmugam*, Seth Pollen, Sharad Punuganti, Shreevatsa R., Sivaraman Sivaraman*, Srinivasan Thirunarayanan*, Suriyhaprakhas Balaram Sankari, Sy Jin Cheah, Teri Zhao (EMC), Thomas Griebel, Tongxin Zheng, Tony Adkins, Torin Rudeen (Princeton), Tuo Wang, Varun Vats, William Royle (Grinnell), Xiang Peng, Xu Di, Yudong Sun, Yue Zhuo (Texas A&M), Yufui Ren, Zef RosnBrick, Zuyu Zhang. Special thanks to those marked with an asterisk above, who have gone above and beyond in their suggestions for improvement.

In addition, a hearty thanks to Professor Joe Meehean (Lynchburg) for his detailed notes on each chapter, to Professor Jerod Weinman (Grinnell) and his entire class for their incredible booklets, to Professor Chien-Chung Shen (Delaware) for his invaluable and detailed reading and comments, to Adam Drescher (WUSTL) for his careful reading and suggestions, to Glen Granzow (College of Idaho) for his detailed comments and tips, and Michael Walfish (NYU) for his enthusiasm and detailed suggestions for improvement. All have helped these authors immeasur-

ably in the refinement of the materials herein.

Also, many thanks to the hundreds of students who have taken 537 over the years. In particular, the Fall '08 class who encouraged the first written form of these notes (they were sick of not having any kind of textbook to read — pushy students!), and then praised them enough for us to keep going (including one hilarious "ZOMG! You should totally write a textbook!" comment in our course evaluations that year).

A great debt of thanks is also owed to the brave few who took the xv6 project lab course, much of which is now incorporated into the main 537 course. From Spring '09: Justin Cherniak, Patrick Deline, Matt Czech, Tony Gregerson, Michael Griepentrog, Tyler Harter, Ryan Kroiss, Eric Radzikowski, Wesley Reardan, Rajiv Vaidyanathan, and Christopher Waclawik. From Fall '09: Nick Bearson, Aaron Brown, Alex Bird, David Capel, Keith Gould, Tom Grim, Jeffrey Hugo, Brandon Johnson, John Kjell, Boyan Li, James Loethen, Will McCardell, Ryan Szaroletta, Simon Tso, and Ben Yule. From Spring '10: Patrick Blesi, Aidan Dennis-Oehling, Paras Doshi, Jake Friedman, Benjamin Frisch, Evan Hanson, Pikkili Hemanth, Michael Jeung, Alex Langenfeld, Scott Rick, Mike Treffert, Garret Staus, Brennan Wall, Hans Werner, Soo-Young Yang, and Carlos Griffin (almost).

Although they do not directly help with the book, our graduate students have taught us much of what we know about systems. We talk with them regularly while they are at Wisconsin, but they do all the real work — and by telling us about what they are doing, we learn new things every week. This list includes the following collection of current and former students with whom we have published papers; an asterisk marks those who received a Ph.D. under our guidance: Abhishek Rajimwale, Andrew Krioukov, Ao Ma, Brian Forney, Chris Dragga, Deepak Ramamurthi, Florentina Popovici*, Haryadi S. Gunawi*, James Nugent, John Bent*, Jun He, Lanyue Lu, Lakshmi Bairavasundaram*, Laxman Visampalli, Leo Arulraj, Meenali Rungta, Muthian Sivathanu*, Nathan Burnett*, Nitin Agrawal*, Ram Alagappan, Sriram Subramanian*, Stephen Todd Jones*, Suli Yang, Swaminathan Sundararaman*, Swetha Krishnan, Thanh Do*, Thanumalayan S. Pillai, Timothy Denehy*, Tyler Harter, Venkat Venkataramani, Vijay Chidambaram, Vijayan Prabhakaran*, Yiying Zhang*, Yupu Zhang*, Zev Weiss.

A final debt of gratitude is also owed to Aaron Brown, who first took this course many years ago (Spring '09), then took the xv6 lab course (Fall '09), and finally was a graduate teaching assistant for the course for two years or so (Fall '10 through Spring '12). His tireless work has vastly improved the state of the projects (particularly those in xv6 land) and thus has helped better the learning experience for countless undergraduates and graduates here at Wisconsin. As Aaron would say (in his usual succinct manner): "Thx."

## Final Words

Yeats famously said "Education is not the filling of a pail but the lighting of a fire." He was right but wrong at the same time[4]. You do have to "fill the pail" a bit, and these notes are certainly here to help with that part of your education; after all, when you go to interview at Google, and they ask you a trick question about how to use semaphores, it might be good to actually know what a semaphore is, right?

But Yeats's larger point is obviously on the mark: the real point of education is to get you interested in something, to learn something more about the subject matter on your own and not just what you have to digest to get a good grade in some class. As one of our fathers (Remzi's dad, Vedat Arpaci) used to say, "Learn beyond the classroom".

We created these notes to spark your interest in operating systems, to read more about the topic on your own, to talk to your professor about all the exciting research that is going on in the field, and even to get involved with that research. It is a great field(!), full of exciting and wonderful ideas that have shaped computing history in profound and important ways. And while we understand this fire won't light for all of you, we hope it does for many, or even a few. Because once that fire is lit, well, that is when you truly become capable of doing something great. And thus the real point of the educational process: to go forth, to study many new and fascinating topics, to learn, to mature, and most importantly, to find something that lights a fire for you.

*Andrea and Remzi*
*Married couple*
*Professors of Computer Science at the University of Wisconsin*
*Chief Lighters of Fires, hopefully*[5]

[4] If he actually said this; as with many famous quotes, the history of this gem is murky.

[5] If this sounds like we are admitting some past history as arsonists, you are probably missing the point. Probably. If this sounds cheesy, well, that's because it is, but you'll just have to forgive us for that.

## References

[CK+08] "The xv6 Operating System"
Russ Cox, Frans Kaashoek, Robert Morris, Nickolai Zeldovich
From: http://pdos.csail.mit.edu/6.828/2008/index.html
*xv6 was developed as a port of the original* UNIX *version 6 and represents a beautiful, clean, and simple way to understand a modern operating system.*

[F96] "Six Easy Pieces: Essentials Of Physics Explained By Its Most Brilliant Teacher"
Richard P. Feynman
Basic Books, 1996
*This book reprints the six easiest chapters of Feynman's Lectures on Physics, from 1963. If you like Physics, it is a fantastic read.*

[HP90] "Computer Architecture a Quantitative Approach" (1st ed.)
David A. Patterson and John L. Hennessy
Morgan-Kaufman, 1990
*A book that encouraged each of us at our undergraduate institutions to pursue graduate studies; we later both had the pleasure of working with Patterson, who greatly shaped the foundations of our research careers.*

[KR88] "The C Programming Language"
Brian Kernighan and Dennis Ritchie
Prentice-Hall, April 1988
*The C programming reference that everyone should have, by the people who invented the language.*

[K62] "The Structure of Scientific Revolutions"
Thomas S. Kuhn
University of Chicago Press, 1962
*A great and famous read about the fundamentals of the scientific process. Mop-up work, anomaly, crisis, and revolution. We are mostly destined to do mop-up work, alas.*