Hand-made OS in 30 days!

Hidemi Kawai (translated by bumbread)

Translation note: the translation notes will be marked in blue color and appear with a smaller font size. Some of the notes clarify the explanations, while other provide additional directions, specific to this translation.

- The support site for this book:

  https://book.mynavi.jp/supportsite/detail/4839919844.html

- This book is the electronic version of the book released in 2006.
- The paper version of the book was shipped with a CD-ROM. The electronic version does not include a CD-ROM, but you can download it from the above website. You can also download it from the translation repository for this book, it is located at ./cdrom
- This book contains commentary and explanations related to the CD-ROM, please read it accordingly.

- The information in this book is basically the production stage of the above-mentioned book.
- The specifications, information, software versions, URLs etc. appearing in this document can change with time. The archived versions of the URLs are provided, some information may have been updated and reflected in this document.
- The screen images provided are examples of reproduction in an environment with specific settings. Note that depending on hardware and software, the screen may not look exactly the same.
- This book is for informational purposes only. All operations are at your own risk and discretion.
- Although we have made accurate statements within the book, neither author nor publisher (nor the translator :D) takes the responsibility for any operational results.
- Company names and product names are registered trademarks of their respective owners.

Though the format of this book is simplified compared to the original, most of the content is preserved, some may have adapted for the general english audience. For example japanese versions of windows use the yen sign (¥) as path separators (the JIS X 0201 encoding is mostly like ASCII but instead of backslash they have the yen sign). Minor details like this are only reflected in text and should display normally on any other computer without modification, since the files on the CD-ROM are not UTF-8 encoded.

Preface

"I want to make an OS". Every programmer has had friends that had this dream. Well, "every" is probably an overstatement, but this phrase most likely hits the top 10 programmer's dreams.

You probably think that creating an OS is a tremendous task, but that's just OS insdustry's conspiracy (lol). If one believes the OS are difficult to make, they can put sell it under a high price tag, also they creators of OS get respected, etc. ...but what about the reality? Compared to other programs, the OS are not that hard to make. At the very least the writer of this book thinks so.

Among the readers there may probably people who tried to make an OS once, and gave up because it was too difficult. People like this probably won't find this explanation very be convincing. ...no, no, it didn't fail because it was difficult, it's just that there was no one to explain that it was easy.

It doesn't apply only to creation of OS. Even if you get an explanation from a person who thinks about it as a hard thing, you can't expect a nice easy-to-understand explanation. Even if you make them explain the same thing, they should give difficult and complicated explanation. It naturally becomes hard to understand.

Come on, would you go onto a challenge together with the author? If you ever thought about making an OS once, you'll probably enjoy it.

□□□□□

You probably think "where's the `easy' and `fun' that you're talking about, on this 700 page book?". I understand its weak when it's said, but it simply got longer. On average it's 22 pages per day. See? It's not that long.

Since the text is too light it's possible that you will read it too swiftly. But if you do that it won't stay in your head, so try chewing through the text slowly. Since the code is as important as the explanation, read through the code carefully as well. If you're aware of this, you should understand the text enough.

In this book the OS is written using the C language and an assembly language. But don't worry. You can study these while making an OS. I'm writing carefully, to a degree, where where are people who say things like "I started understanding pointers in C for the first time after reading this book!". So even if you start from this point, you'll still make a nice OS in 30 days. Please enjoy.

Table of Contents

Preface....3
Chapter 0 *(day 0)*....5
1. Introduction....5
2. What is an OS?....7
3. Various ways to make an OS....7
4. Is it ok if I'm ignorant?....8
5. What to do to make an OS?....9
6. Hardships when making an OS....10
7. When reading chapter 1 and beyond (important!)....11
8. The flow of the story....13
Chapter 1 *(day 1)*....14
Chapter 2 *(day 2)*....14
Chapter 3 *(day 3)*....14
Chapter 4 *(day 4)*....14
Chapter 5 *(day 5)*....14
Chapter 6 *(day 6)*....14
Chapter 7 *(day 7)*....14
Chapter 8 *(day 8)*....14
Chapter 9 *(day 9)*....14
Chapter 10 *(day 10)*....14
Chapter 11 *(day 11)*....14
Chapter 12 *(day 12)*....14
Chapter 13 *(day 13)*....14
Chapter 14 *(day 14)*....14
Chapter 15 *(day 15)*....14
Chapter 16 *(day 16)*....14
Chapter 17 *(day 17)*....15
Chapter 18 *(day 18)*....15
Chapter 19 *(day 19)*....15
Chapter 20 *(day 20)*....15
Chapter 21 *(day 21)*....15
Chapter 22 *(day 22)*....15
Chapter 23 *(day 23)*....15
Chapter 24 *(day 24)*....15
Chapter 25 *(day 25)*....15
Chapter 26 *(day 26)*....15
Chapter 27 *(day 27)*....15
Chapter 28 *(day 28)*....15
Chapter 29 *(day 29)*....15
Chapter 30 *(day 30)*....15
Chapter 31 *(day 31)*....15

Chapter 0 *(day 0)*

1. Introduction

In recent times it became possible to create a unique PC with its own personality by chosing and combining together different parts. Also if you use a proper compiler[1], you can create games and tools by yourself. If you use a website builder, you can create a custom website. And if you read the masterpiece called "How to make a CPU"[2], you can make your own CPU.

By the way isn't there something missing here? Right, it's the OS[3] creation. It doesn't have the atmosphere that any beginner can easily challenge creating it. Even though PC, Games, Web Pages, CPU all seem to be challenging for beginners, it's sad that only OS are left out. Well, if they are, I'll undertake writing about it.

I think it's because of the lack of books aimed at beginners, but for some reason OS are thought of as complex and advanced things. Especially Windows, and even Linux, they are so big they take up multiple CD-ROM's, that if you did make something like that as a hobby alone, it will take a desperately long amount of time to make something like that. Definately, they're so big it won't take lifetime to finish them, even I think so.

But no worries needed! I have the experience of developing a small OS, which doesn't even take up 80KB[4] of space. Even though it's small, it's a properly-featured OS. It's not a console[5]-only OS and doesn't omit multitasking.

---

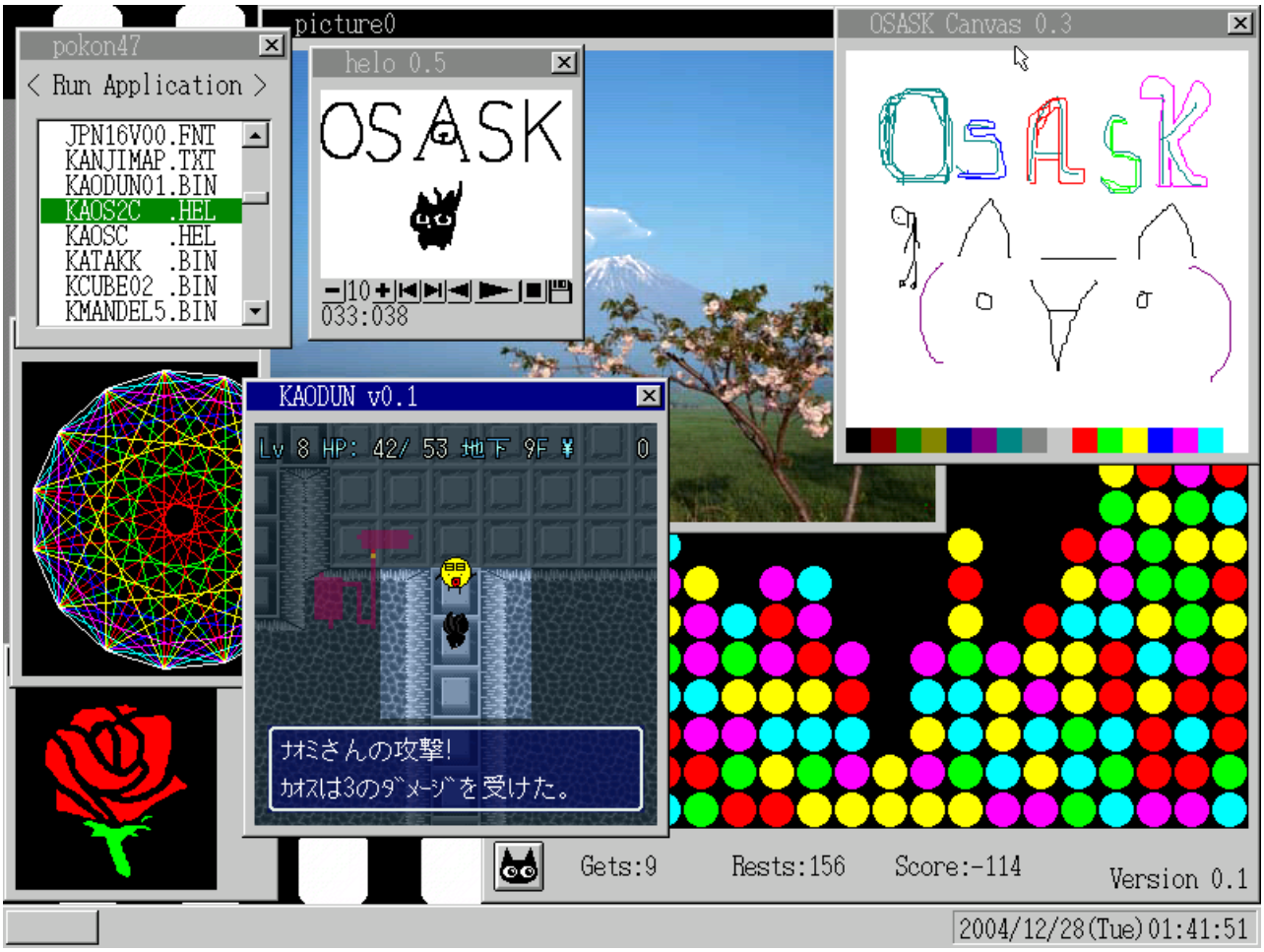
1 Software that generates machine language programs from text files

2 『CPU の創りかた』 by Iku Watanabe

3 Operating System. A general term for software like Windows, Linux, MacOS, MS-DOS.

4 Kilobyte. Unit of measuring amount of data that equals 1024 bytes. The capacity of a 3½ floppy disk is 1440 Kb. By the way 1024 kilobytes is 1 MB (Megabyte). One byte is 8 bits, which is amount of information required to store a sequence of exactly 8 zeroes or ones. It may be difficult to tell whether B is a byte or bit, so in this book uppercase B is used for bytes, and lowercase b is used for bits.

5 Structure that lets you operate the computer by inputting text commands. It was a mainstream computer operating method in the old OSes like MSDOS.



The OS I'm developing [ASKOS]

What do you think? If it's about 80KB, don't you think you can create something like this if you try apply some effort? Don't you feel like even a beginner could create something like this? Yes, it can be made in about a month! With that said let's take it easy.

Even if you never thought about creating your own OS, this book still might be interesting to you. When you read a book about building PC, you learn all about which parts PC is made of and how and where the performance is decided. When you read about how to develop games, you start to understand the structure behind what makes the games move. Likewise, when you understand the process of making an OS, you'll start to see the structure of the OS and what makes them work. So if you have interest in these things, please consider reading a bit.

Also I'll cover it in detail later, but this book requires very little knowledge. Any programming language is fine if you have a sense for writing simple programs in it (even if you don't have this experience it might turn out fine), since this book is suitable for beginners. There is a lot of C in this book, and actually if you've been studying C and gave up because it was too hard it's also fine. Of course, the more things you know the easier the reading would be, but I'm explaining things carefully so rest assured.

In this book we'll talk about the IBM PC/AT compatibles (in other words, Windows computers). For other models, such as Macintosh or PC-9821, this book may contain parts that can be used as a reference, but you won't be able to make an OS that will run on these platforms[6]. Sorry. Strictly speaking when talking about AT compatibles, we're speaking only about computers powered by the Intel 386 CPU or later (since we want to make a 32-bit OS). That's all the models that run Windows 95 or newer, but since (even including second-hand market) its harder to find models that don't support that, most likely your PC can be used without problems.

The takeaway is supposed to be that you won't be able to follow through this tutorial if you use Mac OS. This book works with any PC version of windows and linux. If you use linux, note, that you'll need to build the tools from CD-ROM yourself.

You don't have to worry about the size of RAM or hard disk. If the requirements above are met even old and slow models are fine.

2. What is an OS?

Honestly, I don't know. "But how can you write a book", you might scold me. Sorry... I looked at various OS and there were totally multifunctional OS as well as those which don't have much features. And when I compared all these different OS, I couldn't find anything in common. In various places different author might insist "This is an OS", and people around start thinking "is that true though?", wondering about what kind of program an OS is. Currently that's what I'm thinking.

On the other hand if I was to define what an OS is at the beginning, I'd create convenient situation for myself there, where I could just create the program that would meet these conditions. Of course that would be an OS. For example like MS-DOS, where on a black screen the white characters appear you execute programs by typing in commands. For the author that is easy.

But if we look from readers' perspective, that'll miss the expectations. Beginners when think about an OS imagine have something like Windows or Linux floating in their mind. So what people expect is windows you can move around, a mouse cursor and a few applications. So to meet these expectations we'll make an OS like this.

3. Various ways to make an OS

There are different ways to make an OS.

I think that the best way to make an OS is to find an existing OS that is the closest to the one you want to make and modify it. This will take the least time to make.

But in this book we won't do that, instead we'll do *everything* from scratch. Because I wanted to introduce you OS development in a general way. If we started by finding a base OS, removing all the features we don't need, adding all the features we want, we wouldn't be able to touch the topic of the OS development. I thought that people would be dissatisfied with this. Since we'll be creating

6 Other models: with the contents of the book there's no way to make an OS for macintosh, nor is there a way to run it. However if you created your OS on PC and ran using an emulator, that might work.

everything from scratch the story will be long, well, try reading it leisurely. The author already wrote it carefully.

As to what programming language we'll use, I think we'll go with C. Ah... I can hear the dissatisfied voices (bitter smile). Like "C is not good", "I want to do it in C++", "Java is good", "I like Delphi", "Visual Basic is good", ... I understand your feelings, but for the ease of explanation please accept the C lagnuage. C doesn't have many functions, that's why it's easy to use, so it was the right language. If we did other language, I'd have to first give the explanation of that language which is daunting.

It's a bit sudden, but here's one tip to teach you how to make an OS from scratch. Don't think about how to suddenly make an OS from scratch. You just have to make something that looks like an OS. Because when you think about making an OS you think that you have to do this, or do that, and your head fills up with these things and it just becomes tedius. What to hide, I myself have been repeating this frustration for multiple years. So the motivation isn't that in this book we're making an OS, but rather a demo that looks like an OS. And strangely enough, when making this demo it becomes not a demo but a full OS.

4. Is it ok if I'm ignorant?

When you try to make an OS there surely will be people around you that sprinkle jargon at you such as what the kernel is, what about the shell, whether the kernel is monolithic or microkernel and so on. It may be beneficial at times, but anyway having that told suddenly is bothersome.

In order to keep people around you quiet you have to study these things and show a certain view, but to be frank such complicated things are unnecessary for beginners. You'll just spend a lot of time studying to then end up despairing over how shallow your ideas are compared to world's operating system or you'll be so overwhelmed that you'll just end up combining their techniques, which is not fun at all.

So let's go without studying. Even if you learn jargon and reasoning, knowing only that is not interesting. It doesn't matter how rudimentary your OS will become, as long as you have fun it's fine. Moreover, it's better to try create a simple OS once, see the where the places where problems arise, then learn what is proposed to overcome these problems. This will let you understand complex theory in depth. As a result you won't have the answer for the naggers. In the meantime I'll just have to ask you to talk about it with someone else, because I want to do it at my pace.

□□□□□

On the other hand not knowing is also a good thing. Because you don't know anything, you can also do very silly things experts would laugh at. It's a good thing that we don't know anything. Sure, most of the time it's just plain stupid, but sometimes I discover something amazing that experts have overlooked (!). Experts have a lot of preconceptions, so there are quite a few things that they can't do even if they haven't tried it, or that they have decided it's no good after a while. Only trying to be ignorant like us can challenge these things. Anyone can become an expert by studying, but once you do its very hard to regain the state of being the beginner. So let's try, at least in the beginning to do as

much as we can in ignorance. And if we hit a wall, we can just learn the things that are necessary at that time.

That's exactly what I did, and that's what I'm doing now. I never went to school for programming, and I started to build an OS without learning much of hard theory. But thanks to that I've been praised by many experts, and I've even been given the opportunity to write a book about the OS development for beginners. And ever since starting, I've enjoyed every single day developing it.

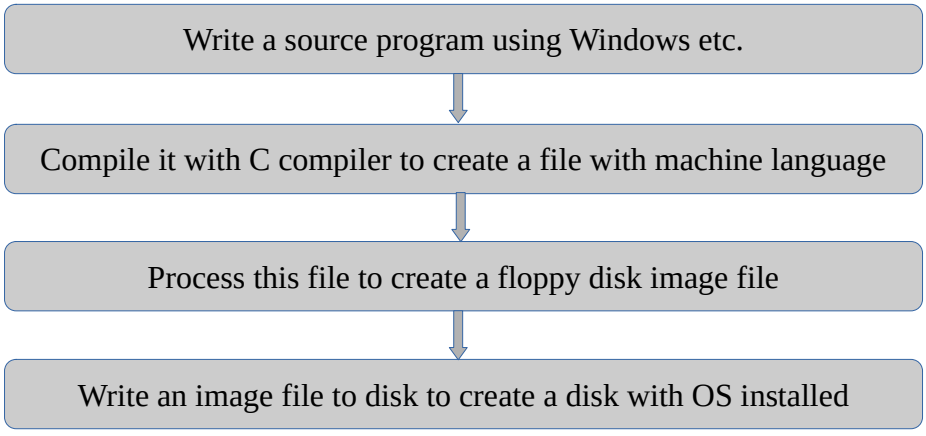
Since we'll be working by exploration, so the story will be easy to understand. However we'll be making mistakes and reworkings, so for those who understand what's going on that may be frustrating. I'm sorry, but you'll have to put up with that.

It may sound like we shouldn't study, but that's not true. If you have to make a program for your job or you have to do this much in a year, then you don't really have the time to go the roundabout way, so it's much better to study before you start making unnecessary mistakes. But this time, I'm making it as a hobby. I want to do it at my own pace, so this is fine.

5. What to do to make an OS?

Speaking of the OS, it starts automatically, when the power is turned on. How do we make such a thing? An executable file (~.exe) created on Windows doesn't start unless you double-click on it with your mouse. This isn't what we want to make, what we'll do is put the CD-ROM or floppy-disk with our OS into the PC, or put the OS onto the hard disk and turn on the power. After we turn on the power the OS should start automatically.

In order to do this we'll perform the following steps:



In other words basically creating an OS means creating a disk that contains the OS and somehow starts automatically.

The disk image file, simply speaking is the backup data for the disk. We want to make a disc with certain content, but we can't directly apply magnet to it and arrange the desired data well. Therefore by first making up the backup data and writing that data to the disk, we can make the desired disk.

Since the total capacity of the floppy disks is 1440 KB, the image file that is backup data is also 1440 KB. If you're free to create any backup image, it's possible to create discs with any content.

I would like you to pay attention here. We're using another OS (Windows) to create our OS. The reason for that is that to use text editor or a C compiler you need an OS running. But what about the world first OS? Of course there's no OS to create the world's first OS. That's why these programmers looked at the CPUs instruction code tables, lined up zeroes and ones, and wrote them to disk (probably not a disk, but another storage device at the time). This is a very difficult task. So maybe the first OS is one with a lot less functionality, and once that OS is created, they used it to create a slightly better OS, and then make a practical OS with that OS... I think they did something like that.

I think that most of the beginners would be Windows users, so I decided to use Windows. Any version of Windows starting from Windows 95 is fine. I'm sure there are also people who are using Linux, so I'll summarise how to do it in Linux on the support page[7]. Please take a look if you need it.

Regarding the C compiler and image file creation tool, small differences may exist and they will be difficult to explain, so I will put them all in the appendix to CD-ROM. Most of the tools are free software[8] released by me when I was working on OSASK. These programs are open source. In addition to these tools I will use some free software, but when the time comes I will explain how to use them in detail.

6. Hardships when making an OS

The C compilers on the market are designed on the premise of creating Windows and Linux applications, and it is hardly considered to create other things such as original OS. The C compiler provided by me is a slight modification of the C compiler called gcc. There may be a C compiler that supports OS development, but I'm not sure. Even if there were, only OS development companies would buy it, so it's probably very expensive. We won't use that kind of thing this time.

So I have no choice but to make an OS somehow using a C compiler for application development. In other words, do the unreasonable. Since it is unreasonable there are various invonveniences that arise.

For example take the function `printf("hello\n");` a function that appears at the beginning of every C textbook. We can't use it. Because `printf` is a function that has to be provided by the OS. But we haven't created the OS yet. So if you happen to call this function the CPU will throw a general protection exception. `printf` is not the only function that cannot be used. Most C functions can't be used.

> If you let me excuse myself, I chose C language this time because there are not so many functions in C language that are premised on the OS support. Basically forgive me for deciding solely on the number of functions. For C++, the basic and important operators such as new/delete can't be used and there are various ways the classes are created, so the good parts of C++ cannot be utilized. Of course, you can build your OS in the way where you can use these you can overcome this. Though it is frustrating to

7 http://hrb.osask.jp/. This is a japanese website so I will think about how to support Linux users later.
8 These programs have been licensed as either GPL or LGPL. The details are in Chapter 31.

> think about whether you're making an OS in C++ or an OS for C++. In other languages the situation might have been even more difficult. So please forgive me.

By the way assembler is probably the only language that has no restrictions when creating an OS. Assembler the Greatest (lol). If I was to write a book like "Let's make an OS with only assembler", very few people would actually read it, so only a reckless writer would do something like that.

> Also when creating an OS it is necessary to play with various registers. Since C compilers are made for application development there are not a single way to operate these registers directly. Also C compilers do some nice optimizations, but it can be a nuisance.

So, in order to overcome these problems the parts that can't be written in C have to be written in assembler. At the time you have to be aware of how C compiler translates your program into machine language. If you don't, you won't be exchange information with parts made in C language. Normally you can't taste that kind of thing by programming applications in C! But the annoyance is stronger than superiority complex (bitter smile).

> Similarly, if you plan to build an OS in C++ you have to know how that language is translated into machine language. Naturally C++ is more complex than C, so the translation rules are more complicated and cumbersome, that's one more reason I decided to go with C. At the end, if you don't know how the language is translated into machine language you can't build an OS in that language.

The books on C and C++, and of course Java and Delphi, and other languages all rarely write about "how is the language translated to machine code". Even though the languages give the instructions to the CPU to execute the program, the books don't explain the basic mechanisms of the CPU. As OSdever[9] I'm lonely. I can't help it so I'll start with that in this book (since it's the basic requirement for OS creation).

This kind of experience may change how you think about programming. Even though until now I was thinking about making a cool and beautiful source program, I noticed that how it's translated into machine language it is also important! No matter how cool your source code is, if it doesn't come out with the right machine language it won't work and it won't make sense. On the other hand if the source code is dirty and can be compiled with a specific C compiler, it's okay if it becomes the target machine language. Once you get the machine language you want you can just throw away the source code! I wouldn't say a thing like that, but having that feeling is nice.

For OSdevers source programs are a "means" of obtaining an OS, not a "purpose". If you're too focused on the means and it becomes a hassle you will get overwhelmed.

Oh by the way, the fact that we're making an OS in C and assembly doesn't mean that C++ applications won't work on it. The language we use to create the OS has nothing to do with the language in which the OS was created. So don't worry about that.

---

9 OS developer.

7. When reading chapter 1 and beyond (important!)

Starting from chapter 1 the days of development are written. The book is divided into days, but I divided it as I was writing, with respect to my current abilities and the lengths of explanations. Whether people find explanations difficult depends on a person so it's quite possible that a single day may take you a week or, on the countrary 3 days may take you a single day.

Of course there are some things you don't understand when you read. In such case it may be a good idea to read for a day or two. Then you may suddenly figure it out. But sometimes, I think, there are cases where you might not understand even after that. In such cases verify how well you understood and it's fine if you return back. Don't rush because the it may turn out to be sloppy.

Sometimes it's clear what you know and what you don't know, and no matter how many times you read back what you don't understand, you can't start understanding it. In such case please check the support page. There may be a commentary on the Q&A page.

□□□□□

The way of explaining pointers in C is different in this book. That's because in this book you will first learn the basic mechanism of the CPU, then do it in assembler and after that study the C language. In other books they don't talk about the basics so when they talk about pointers they suddenly say something about the address of the variable. So if you think you know pointers from other books please read this book as well, you will get it. If you really know what you're doing feel free to skim the explanation.

□□□□□

From now on I will assemble the OS little by little, but as I progress a little the stages in the middle will be summarized. They are summarized on the CD-ROM in the appendix, and if you copy it you can run it immediately. There are caveats in about the program so I would like to write about them.

For example the first program in the appendix is "helloos0", and the program that appears next is "helloos1". If you follow the instruction for the text for "helloos0", it will become "helloos1", wouldn't it? No it wouldn't. That's because sometimes I do some changes behind the scenes. I just didn't explain it because I thought you would understand it without the explanation.

So after all what I want to say is don't just read text, but also look at the programs properly. It's also possible that explanation in text would be vague, but once you look in the code it'll immediately become clear. So the center of this book is not the text, but rather the program in the appendix. The text is just an explanation of how the program in the appendix was created.

Introductions are introductions, not complete explanations. Please be careful not to make a mistake... In that sense rather than "Book and CD-ROM as appendix" it may be "CD-ROM and a book as appendix" (lol).

□□□□□

One more thing about the programs. The copyright of all the programs contained here belongs to the author. However if you read this book and try to make an OS, there may be parts that you want to imitate. Maybe you want to use one confusing part. Or maybe you want to start developing an OS from where the book left off, as if you were making a continuation.

Since this is a teaching material, I thought it would be meaningless if you couldn't use it freely. Therefore you can use it freely. You don't have to file a tax return in advance. It is not necessary to include the author in a copyright notice. You can pretend as if you made it youself. You can sell the OS and get rich. When that happens don't worry I won't tell you to split the share. So without worrying please get rich. (Although if you really want to you can send me the money (lol)).

This is not only a privilege for the person who bought the book, it's OK even if you borrowed it from the library, asked a friend to lend you the book or browsed at a bookstore. Well it would be helpful for the author and the publisher if you bought the book (lol).

As a caveat when diverting in this way, the name of the OS is no longer my work, so please do not make it a misleading name. That's all I want. No matter how similar the internal programs are, it's a different operating system that you released at your own risk. Please give it a worthy name.

The above treatment is only for the programs that appear in this book and the teaching material OS programs in CD-ROM. Reproduction or modification of the text of this document and other tools on the CD-ROM is restricted by copyright law. Please handle within the scope of the law. The licence for the tools in CD-ROM is written in the last chapter.

8. The flow of the story

As you can see from the table of contents there are many items in the table of contents so here is a rough outline. If you want to be "excited and thrilled", wondering what will happen next, while reading this text you can skip this part (lol). As you read this part, if you wonder whether it's actually ok to do that, please thing of this part as the first place to read back.

The translator wanted to be excited and thrilled while translating the book so this part isn't getting translated until the rest of the book is translated. You see, I'm following the tutorial too :D.

Chapter 1 *(day 1)*

Chapter 2 *(day 2)*

Chapter 3 *(day 3)*

Chapter 4 *(day 4)*

Chapter 5 *(day 5)*

Chapter 6 *(day 6)*

Chapter 7 *(day 7)*

Chapter 8 *(day 8)*

Chapter 9 *(day 9)*

Chapter 10 *(day 10)*

Chapter 11 *(day 11)*

Chapter 12 *(day 12)*

Chapter 13 *(day 13)*

Chapter 14 *(day 14)*

Chapter 15 *(day 15)*

Chapter 16 *(day 16)*

Chapter 17 *(day 17)*

Chapter 18 *(day 18)*

Chapter 19 *(day 19)*

Chapter 20 *(day 20)*

Chapter 21 *(day 21)*

Chapter 22 *(day 22)*

Chapter 23 *(day 23)*

Chapter 24 *(day 24)*

Chapter 25 *(day 25)*

Chapter 26 *(day 26)*

Chapter 27 *(day 27)*

Chapter 28 *(day 28)*

Chapter 29 *(day 29)*

Chapter 30 *(day 30)*

Chapter 31 *(day 31)*