SCCP 2014年度後期 s1200196 弥栄　俊介

Minix3によるOSの内部構造の学習

Operating Systems DESIGN AND IMPLEMENTATIONを読み進めていくことによって、Minix3のOS内部構造、一般的なものを学習していく。

なぜこの本を読み進めていくことに決めたのか

大學に入ってからCSを学び初めて最も興味を持ったものがOperating Systemであり、２年次のOS論やOSの自作を通して少しずつ学んできた。そこで今回SCCPでは教育用OSとして有名なMinixの構造と実装を解説しているこの本を選び読んで行くことにした。

内容に入る前に

本を読み進めていくにあたって、ソースコードと説明を行き来することが多く、本の末ページに乗っているが、参照しにくいのでGithubでminix3のコードが管理されていたので、基本的にgithubに公開されているのを参照した。
ここで最新のバージョンだと、変更部分が多くあまり参考にならないが、v3.0などのコードが本の説明と合致するのでそちらを使用した。

この本のChapter. 1とChapter. 2の前半の内容が、前年度のOS論の講義で学んだ基礎と重複するために、他の一歩進んだ内容の節に時間をかけて読み進めた。

Chapter. 1はOSのIntroductionで、内容はOSとは何なのか、歴史、システムコールについて、OSの構造、この本のアウトラインになっている。システムコールとOSの構造については、この後の章に詳しくあるので簡単な説明にとどまっている。

Chapter. 2はProcessesというタイトルで、前半の部分2.1〜2.4の内容は、プロセスとは何なのか、プロセス間通信、スケジューリングについての説明。現代のOSではアプリケーションが同時に複数動いてることが当たり前であり、そのために実装されるのがプロセスである。

今回のSCCPで詳しく読んだのはChapter 2の続きの

- 2.5 Overview of Processes in Minix3
- 2.6 Implementation of Processes in minix3

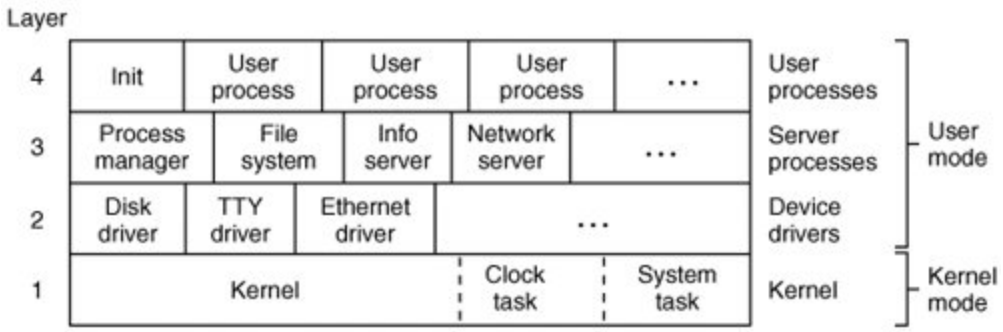
の２章である。

OSDI 2.5章、 2.6章

2.5 Overview of processes in Minix3
Minix3におけるプロセスの扱い方や、システムとしてのプロセス、Layerの分け方など概念を中心にこの章では扱っている、実際の実装については2.6章で扱う。

2.5.1 The Internal Structure of Minix3

minixでは、プロセスのモードをkernelとuserの２つに分けている。
そしてuser modeでは更に、User processes, Server processes, Device driversの３つの階層に分かれている。



(*1より)

大まかな分類の代表的なプロセスをまとめたものが上の図である。

- Kernel modeとは

Kernel modeとはx86のリングプロテクションの特権モードとされるring0のことであり、上の図のLayer 1,2,3,4はring0, 1, 2, 3として階層化され実装されている。

kernel の機能は
- process schedule
- transition process state(ready, running, blocked)
- all message handle (between process)
- message handling requires check
- locate send and receive buffer (in physical memory)
- etc...

などがある。

kernelにはtaskとして一つの機能をもたせ分離したものがclock taskとsystem taskの２つある。
clock taskとはハードウェアで作られるタイマーのシグナルを受け取り管理する。clock taskは直接ユーザーからはアクセスできない。
system taskはシステムコールの呼び出しの実装。

- User modeとは

User mode は前述のとおり３つのレイヤに分かれていて、Device driver, Server processes, User processesがあり、はじめの２つのDeivce driverとServer processはOSの機能の一部であり、ユーザーが普段プロセスを起動したり,shellを起動したり目に触れるものはUser processesにある。

- Device driverについて
  ここでは名前の通りに、デバイスドライバがプロセスとして動いている

- Server processes
  OSとしての機能を提供するProcessがこのレイヤーで動いていていくつか代表てきなものを上げると
  - PM(Process manager)
  - FS(File system)
  - NS(network server)

などがある。

ユーザーがOSの機能を使おうとした際に、System callを通じてアクセスする。
System callはServer やDevice Driverのプロセスにアクセスして、更にそれぞれのサーバーはkernel callとしてカーネルにアクセスすることで機能が呼び出される。
system call はkernel callのサブセットである。

2.5.2 Process Management in Minix3

Minixにおいて、すべてのプロセスはboot時に起動するinitプロセスのサブプロセスである。
このinit processの起動の流れは、minixのOS自体の起動は一般的なOSと同じ通り、bootstrapをRAM上にロードして、そのbootstrapがOS本体をロードされ、OSが起動する。OS本体としてロードされるboot imageにはkernel (clock task, system taskを含む), process manager,file system が最低でも含まれていなければならない。
boot imageではハードウェアの初期化などが行われたとinit processが起動する。
init processは必要なサーバーを起動して、PID０となる。

2.5.3 Interprocess Communication in Minix3
プロセス間の通信にはいくつかのCの関数が用意されている。
最もシンプルなのが

- send(dest, &message);

dest processにmessageを送信する。

- receive(source, &message)

source processからメッセージをmessageに挿入する。;

- sendrec(src_dest, &message);

返信の為の関数

他には通知のみをする

- notify(dest);

などがある。

2.5.4 Process Scheduling int minix3

Minx3ではプロセススケジューリングにマルチレベルキューを使用してる。
マルチレベルキューは、重要度によって別れ、それぞれ実行される。
現段階の自作OSでのスケジューリングでは、シングルキューによって管理され、優先度も存在しない単純な作りになっているので、MinixやLinuxを参考に作りこみたいと考えている。

2.6 Implementation of Processes in Minix3
2.5章ではMinix3におけるプロセスの実装の概要についての説明があったが、この章では実際のコードを用いて、構造体、マクロ、関数の説明される。

以下の項目はタイトルの通り、Source Codeの作りなどの話なので今回は説明を省略する。
2.6.1 Organization of Minxi 3 Source Code
header fileのパス、ライブラリ、ツールなどのディレクトリ構成
2.6.2 Compiling and Running in Minix3
コンパイルの方法、メモリに展開された際のmemory map
2.6.3 The Common Header Files
include ディレクトリにある、ヘッダーファイルのマクロの説明
2.6.4 The Minix3 Header Files
include/minix, include/ibmにある、ヘッダーファイル
2.6.5 Process Data Structures and Header File.
プロセスを管理する構造体の説明

2.6.6 Bootstrapping Minix3
2.6.7 System Initalization
2.6.8 Interrupt handlining minix3
ハードウェア割り込みの為の処理を割り込みテーブルにそれぞれ登録し、初期化をした後、割り込みがあると割り込みマネージャが割り込み先を確認して、割り込みテーブルの適当な処理を呼び、実際の処理が行われる。
intel のx86での実装なので、基本的にはやらなければならないことは自作OSで行っているものと、ほとんど同じであった。

2.6.9 Interprocesses Communication in Minix3

Overviewで説明したプロセス間通信のsend, receive, notifyはそれぞれ、引数の値が違反していないかを確認したあと、実際に動作が実装してあるmini_send, mini_rec, mini_notifyが呼び出される。
自作OSにはプロセス間通信の実装がまだなく、今回読んだことを活かして作ろうと思ってる。

まとめ

今回はOSDIの2.5章と2.6章を読み進めていった。内容の中心はMinixにおけるプロセスについてで、特有のプロセスの構造であったり、プロセス間通信の実装方法など、マルチタスクの要の内容であった。

洋書の読んでいく速度が遅いのでもっと数を読まなければいけないと実感した。Minixのマイクロカーネルな簡単な仕組みを理解することができたが、その実際の実装まで詳細まで踏み込無ことができなかったことが残念だった。今回のSCCPとしてはここまでしか進まなかったが、興味が大きいのプロセスに引き続きtaskやディスクも読んでいきたいと考えている。
現在自作しているOSのプロセスは最小限でかつ、スケジューラーもまともなものでは無いので今回読んで理解したことなどを使って、よりよいものに作り変えて動かしてみようと考えている。

参考文献

[1]. Andrew A. Tanenbaum and Albert S. Woodhull, Operating System Design and Implementation Third edition, Pearson Education International, 2009
[2]. 浦地輝尚, 初めて読む486, アスキー出版, 2006
[3] Minix source code (http://github.com/minix3/)

引用

*1 : [1]. page 113.
Figure 2-29, [Minix 3 is structured in four layers Only processes in the bottom layer may user privileged(kenel mode) instructions]より