# オープンソースの組み込みOSを用いた「オペレーティングシステム」の実習

Exercises Using an Open Source Embedded OS for "Operating Systems" Course

**塩澤秀和**

Hidekazu Shiozawa

玉川大学工学部ソフトウェアサイエンス学科，　194-8610　東京都町田市玉川学園6-1-1

Department of Software Science, College of Engineering, Tamagawa University,

6-1-1 Tamagawagakuen, Machida, Tokyo 194-8610

Abstract

Department of Software Science, Tamagawa University offers the subject "Operating Systems" since 2006. This paper reports the exercises using an open source embedded operating system in this course for the students to understand internal structures and algorithms of operating systems. As a result of evaluating several open source operating systems for the course, the author selected HOS (Hyper Operating System) developed based on the μITRON specifications. It is expected that the programming exercises using a real operating system help students understand and investigate more practically how operating systems work internally.

Keywords: programming exercises, educational operating system, embedded OS, computer science education

## 1. はじめに

ソフトウェアサイエンス学科では，3年次または4年次の科目として「オペレーティングシステム」が開講されている．この科目がテーマとするオペレーティングシステム（OS）は，複雑なコンピュータシステムを動かすためになくてはならないソフトウェアであり,「基本ソフトウェア」とも呼ばれる．代表的なOSとしては，Windows，macOS，Linuxなどがよく知られている．

この科目は, OSの基本的な内部構造やアルゴリズムを学ぶものであり，大学の情報系（ソフトウェア系）学科では一般的な内容である．しかし，OSの主要部分は通常隠蔽されており, その内部で行われている複雑な処理をユーザが意識する必要はない．よって，学生にとっても，それらをイメージして技術的に理解するのは，必ずしも容易なことではない．

そこで, 大学の専門教育では, 講義だけでなく, ソースコードが公開された（オープンソースの）オペレーティングシステムを使って，その内部構造を調べたり，自分で新しい機能を追加したり，一部の処理を改造したりするような実習を取り入れている例が（特に海外に）多い．ソフトウェアシステムの動作を知るには，ソースコードを読むのが最も深く理解できる方法だといえる．

例えば，Linuxはソースコードと内部構造が完

全に公開されているので，プログラミング実習をともなう大学の授業での使用例[1)]が多い．そのLinuxは，書籍で内部構造が詳しく解説されたMINIX[2)]という教育用OSがきっかけとなって開発が始まったものである．

このような背景のもと，2006年度から筆者が担当する本科目でも，本学の学生に適切な難易度を考慮した上で，実際に少しでもOSの内部処理に迫るようなプログラミング実習を行なうことを試みてきた．

本科目では，小規模なオープンソースOSをいくつか検討した結果，μITRON[3)]の仕様に基づいて開発されたHOS（Hyper Operating System）[4,5)]を用いている．このOSは，開発者自身がOSの仕組みを学ぶための習作として開発した非常に小さな組み込みOSであり，C言語で開発されてる．

HOSのソースコードは，実行効率よりも分かりやすさを優先しているようであり，コメントも日本語なので内部処理を理解しやすい．また，HOSは，ARMなどの組み込み用マイコンで動作するだけでなく，タイマー割り込み等のエミュレーションによって，Windowsの中のアプリケーションプログラムとしても動作するので，学生が自分のPCで動作させて実習を進めることができる．

## 2. 本科目の全体構成

表1は，本科目の2017年度のシラバスである．全体的な流れとしては，インタフェース，プロセス管理，同期処理，メモリ管理，ファイルシステムの順で進行する．各回の詳細は，筆者が文献6)のWebサイトで公開している授業資料（パワーポイント資料）を参照されたい．

本科目ではほぼ毎回，授業の最後に実習課題を出している．表の右のHOSの列は，その中でも特にHOSを利用する課題を出した回である．これらの課題は，毎回の授業内容に関連しているが，毎

**表 1　「オペレーティングシステム」のシラバス（2017 年度）**

| 授業回 | 内容 | 主なキーワード | HOS |
|---|---|---|---|
| 第1回 | オペレーティングシステム概論 | 授業ガイダンス，コンピュータとOSの歴史，OSの種類と役割，OSの基本構造 | ○ |
| 第2回 | OSのインタフェース | ユーザインタフェース（シェル，GUI），プログラミングインタフェース（システムコール，API，サービスコール） | |
| 第3回 | カーネルとデバイスドライバ | カーネル（核）の役割，デバイスドライバ，ブートシーケンス（OS内部の起動手順） | |
| 第4回 | プロセスとスレッドの基礎 | プログラム実行手順，マルチタスク，プロセスの状態遷移，プロセスの親子関係，プロセスのメモリマップ | ○ |
| 第5回 | コンピュータハードウェア | CPUの仕組み（レジスタ，プログラムカウンタ，スタックポインタ），割り込み，プログラムの読み込みと実行 | |
| 第6回 | プロセスのスケジューリング | プロセス情報の管理，コンテクストスイッチ，スケジューリングアルゴリズム（先着順，優先度，EDF，SJF，RR） | ○ |
| 第7回 | 小テストとプログラミング実習 | 小テストと集中実習 | ○ |
| 第8回 | 小テストの解説と中間レポート提出 | | |
| 第9回 | 共有資源と資源管理 | 共有資源（リソース），クリティカルセクション，デッドロック，資源（リソース）管理 | ○ |
| 第10回 | プロセスの同期とプロセス間通信 | 排他制御機能，プロセス同期機能，セマフォ，プロセス間通信，同期型通信と非同期型通信 | ○ |
| 第11回 | メモリ管理 | 再配置可能プログラム，メモリ割り付け方式，メモリの断片化とコンパクション，メモリ保護，メモリ割当てAPI | ○ |
| 第12回 | 仮想記憶 | 仮想記憶，仮想アドレスと実アドレスの変換，ページング，ページ置換アルゴリズム（FIFO，LRU），スラッシング | |
| 第13回 | ファイルシステムとセキュリティ | ファイルとディレクトリ，仮想ファイルシステム，ユーザ管理，認証，ファイルやプロセスの保護，代表的な攻撃 | |
| 第14回 | 小テストとプログラミング実習 | 小テストと集中実習 | ○ |
| 第15回 | 小テストの解説と期末レポート提出 | | |

回の宿題とはせずに，中間と期末の2回のレポートとしてまとめて提出させている．提出回の前の回には，集中的に時間をとって授業中に実習（特に質問対応等）の機会を設けている．

## 3. 本科目の実習課題

以下順に，本科目の2017年度の授業において，実際にHOSを利用して行ったプログラミング実習の課題の概要を報告する．具体的な履修者への説明は文献6)を参照してほしい．

### 3.1 実習環境の構築

第1回から第3回までは，OSの構造などの概要がテーマである．実習では，学習者は各自のPCで実習環境の構築を行う．以下に各課題の内容を示す．

**・HOSのインストール（準備課題）**

第1回の授業では，学習者は実習用のOSであるHOSを各自のPCにインストールする．その際，本科目で用意したHOSの配布ファイルには，課題のプログラムが先の回の分まであらかじめ同梱されている．また，HOSの実行にはVisual Studioが必要であるため，未インストールの者にはこの回でインストールしてもらった．授業初回であるのでPCを持参していなかった者には，第3回までにインストールするように指示して宿題とした．

**・課題3a HOSのブート過程**

第3回の実習課題では，HOSを題材にしてOSのブート（起動）過程を観察する．学生は，HOSに付属しているサンプルプログラムを開いて実行し，デバッガのステップ実行機能によって，1つずつ関数の処理を追いかける．それと同時に実行される関数を順に記録し，日本語のコメントから処理内容を推測する．この課題はあくまでもHOSに慣れるのが目的である．

### 3.2 プロセスに関する実習

第4回から第6回までの授業では，プロセスの基礎とOSが複数のプロセスを見かけ上同時並行的に実行するマルチプロセス（マルチタスク）について学ぶ．その際，CPUの構造等についての知識が必要なので第5回で簡単に扱う．

なお，HOSは組み込みOSであり，実行主体である「タスク」は，メモリ領域が割り当てられていないなど，通常のOSのプロセスと全く同じものではないが，本科目の実習においては，実行動作を理解する上では同等とみなして扱っている．

**・課題4a HOSにおけるタスクの実行**

第4回の実習課題は，まずタスクが1つしかないシングルタスクのプログラムを実行し，ITRONにおけるタスクを用いたプログラミングの基本を理解するのが目的である．学習者は，課題のプログラムをコンパイルして実行し，表示結果からタスクの動作について考察する．特に，以降の課題に進む前提として，自タスクを指定時間停止するdly_tsk（delay task）というITRONのAPI（Application Program Interface）について理解しておくことが求められる．

**・課題4b HOSにおけるマルチタスク**

第4回の次の課題の狙いは，マルチタスクの基本を理解することである．一般的なOSでは，実行中のタスクが自ら休止等の処理をすることによって，実行を一時中断して他のタスクに実行を切り替えることができる．課題のプログラム（図1）では，OS起動後に実行されるstart関数の中で，task1の次にtask2が起動されているが，実行結果ではtask1とtask2による表示が混在して現れる．学習者は，dly_tskによる実行権の受け渡しと，task1とtask2の処理のタイミングについて理解することが求められる．

**・課題6b 先着順+優先度スケジューリング**

第6回の授業では，各種のスケジューリングアルゴリズムについて学ぶ．最初の課題は，実際のプログラミングによって，先着順スケジューリングと優先度スケジューリングの仕組みを理解することである．先着順に関しては，start関数の中における起動順で指定することができ，優先度は，

```
/* タスクの情報 */
const T_CTSK ctsk1 = { TA_HLNG | TA_ACT,
  0x1234, task1, 5, 256, NULL };
const T_CTSK ctsk2 = { TA_HLNG | TA_ACT,
  0x1234, task2, 5, 256, NULL };

/* タスクの生成処理  */
void start(VP_INT exinf)
{
  cre_tsk(1, &ctsk1);  /* タスク ID 1 */
  cre_tsk(2, &ctsk2);  /* タスク ID 2 */
}

/* タスク 1 の処理 */
void task1(VP_INT exinf)
{
  SYSTIM st;

  for (;;) {
    get_tim(&st);  /* 経過時間の取得 */
    printf("1: %ld sec¥n", st.ltime/1000);
    dly_tsk(2000);  /* 指定時間停止 */
  }
}

/* タスク 2 の処理 */
void task2(VP_INT exinf)
{
  SYSTIM st;

  for (;;) {
    get_tim(&st);  /* 経過時間の取得 */
    printf("2: %ld sec¥n", st.ltime/1000);
    dly_tsk(3000);  /* 指定時間停止 */
  }
}
```

**図 1　マルチタスクの課題プログラム**
**（main.c から抜粋）**

図1の冒頭にあるタスクの設定情報で指定可能である．学習者は，各タスクの実行順序や優先度を変更していくつかの組み合わせを試し，それらの効果について理解することが求められる．

**・課題6c ノンプリエンプティブなマルチタスク**

ノンプリエンプティブなマルチタスクとは，各プロセスが自主的に実行権を手放すことで，複数プロセスの並列的な実行を実現する方法である．この課題では，その動作とプログラミング作法を理解する．ITRONは優先度ごとにレディキューを持ち，rot_rdq（rotate ready queue）というAPIによって次のタスクに実行を移すことができる．図2は課題のプログラムの一部であり，学習者は(a)の中のrot_rdq関数を有効にし，実行結果を考察する．

```
/* タスク 1 の処理 */
void task1(VP_INT exinf)
{
  int i;
  for (i = 0; i < 20; i++) {
    delay();
    printf("1");
    delay();
    /* 他のタスクに自主的に処理を譲る */
    // rot_rdq(5);  /* 優先度 5 のキュー */
  }
  ext_tsk();
}

/* タスク 2 の処理 */
void task2(VP_INT exinf)
{
  int i;
  for (i = 0; i < 20; i++) {
    delay();
    printf("2");
    delay();
    /* 他のタスクに自主的に処理を譲る */
    // rot_rdq(5);  /* 優先度 5 のキュー */
  }
  ext_tsk();
}
```

**(a) main.c から一部抜粋**

```
/* タイマ割り込みハンドラ */
void OsTimer_Handler(VP_INT exinf)
{
  ID id;

  isig_tim();
  if (iget_tid(&id) != E_OK) return;

  /* レディキューの回転（強制的なタスク切替え） */
  // irot_rdq(3);  /* 優先度 3 */
  // irot_rdq(4);  /* 優先度 4 */
  // irot_rdq(5);  /* 優先度 5 */
}
```

**(b)　ostimer.h から一部抜粋**

**図 2　スケジューリング処理の課題プログラム**

なお，delay関数は別の箇所で定義されており，計算によって時間を消費するものである．

**・課題6d ラウンドロビンスケジューリング**

第6回の最後の課題では，ラウンドロビンスケジューリングによるプリエンプティブなマルチタスクを実現する．ラウンドロビンスケジューリングでは，OSが一定周期で実行プロセスを順番に（かつ強制的に）切り替えていく．そのためにOSはタイマー割り込みでスケジューラーを駆動し，

プロセスの切り替えを行う．ITRONでは，これはタイマー割り込み処理の中で，irot_rdq（rot_rdqの割り込み対応版）を使ってレディキューを回転させることで実現できる．図2(b)は，HOSをWindowsで動かす場合のタイマー割り込み処理のエミュレーション部分であり，学習者はこの中のirot_rdqを有効にして，実行結果を考察する．

### 3.3 同期処理に関する実習

第9回と第10回のでは，共有資源に関する排他制御とプロセスの同期処理について学ぶが，プログラミング実習があるのは，第10回だけである．

**・課題10a セマフォを使った排他制御**

セマフォとは，「残り数」を表す一種のカウンタであり，共有資源の排他制御に用いられる．この課題では，排他制御の必要性とそのためのセマフォの使い方を，実際のプログラミングを通して学ぶ．ITRONでは，セマフォのP操作（獲得操作）はwai_sem（wait semaphore），V操作（解放操作）はsig_sem（signal semaphore）というAPIを用いる．図3は課題プログラムの一部であり，共有資源にアクセスしている領域（クリティカルセクション）をwai_semとsig_semではさむことで，適切な排他制御が実現できる．学習者は，セマフォの仕組みと役割について理解することが求められる．

**・課題10b デッドロックと排他制御**

デッドロックとは，複数のプロセスが複数の資源の獲得をしようとするときに，お互いがお互いを待つ循環待ちの関係になり，システムが停止してしまう状態である．この課題の狙いは，これを実際のプログラムで経験した上で，それが起きないようにする対策を考え，検証することである．課題のプログラムは，wai_semとsig_semを用いた排他制御の手順が適切でないために，デッドロックが発生するようになっている．学習者は，第9回の授業で学ぶデッドロックの防止策をプログラムの形で表現し，実行結果を示して検証することが求められる．

```
/* セマフォで守るべき共有資源 */
int shared = 0;

/* 起動ルーチン */
void start(VP_INT exinf)
{
  int i;

  /* セマフォの生成 */
  cre_sem(SEMID, &csem);

  /* 同じタスクのコピー10 個の生成 */
  for (i = 1; i <= 10; i++) {
    cre_tsk(i, &ctsk);
  }
}

/* タスク関数 */
void task(VP_INT exinf)
{
  int a;

  /* ここから shared のクリティカルセクション */

  a = shared;
  a += 10;
  printf("read : %d¥n", a);

  work_something();  /* 時間稼ぎの処理 */

  shared = a;
  printf("write: %d¥n", shared);

  /* ここまで shared のクリティカルセクション */

  ext_tsk();
}
```

図 3　排他制御の課題プログラム（抜粋）

**・課題10c HOSにおけるタスク間通信**

本科目では，プロセス間通信に多くの時間は割けないが，この課題では基本的なプロセス間通信（タスク間通信）について，プログラミングを体験する．ITRONにはいくつかのタスク間通信機能があるが，課題のプログラムでは文字列（のかたまり）を送受信するしくみであるメッセージバッファを使用する．学習者は，snd_mbf（send message buffer）を利用する送信タスクと，rcv_mbf（receive message buffer）を利用する受信タスクの処理を理解し，送信側と受信側の対応を複数対複数にするなどして，特に非同期的なプロセス間通信について理解することが求められる．

### 3.4 メモリ管理に関する実習

第11回ではメモリ管理，第12回では仮想記憶，第13回ではファイルシステムについて学ぶが，どれも小規模な組み込みOSでは重視されない機能なので，HOSのようなOSでは対応が難しい．そのためこの部分は実習が不十分とならざるを得ず，今後の課題である．

**・課題11b HOSにおけるメモリ割り当て**

この課題では，基本的なメモリ割り当ての手法とメモリの断片化という現象を実際のプログラムで学ぶ．HOSのような組み込みOSでは，メモリプールと呼ばれる固定領域を予め確保しておき，そこからメモリを割り当てることが多いが，基本的な考え方は，動的な割り当て処理と共通である．課題のプログラムでは、get_mpl（get memory pool）とrel_mpl（release memory pool）というAPIを繰り返すうちに空き領域の総バイト数は足りているはずなのに，断片化によって新たにメモリを確保できない現象を発生させる．学習者はその原因について考察することが求められる．

### 3.5 発展課題

最後に，学習者がOSの内部処理についてより深く学んでいくきっかけとするために，発展課題としてHOSのソースコードを解析する課題を出している．しかしながら，実際のところはこれらの課題を提出する者はほとんどいない状況である．

**・課題14a rot_rdq の内部処理を見てみよう**

HOSのソースコードから，第6回の実習課題で用いたrot_rdqの処理の主要な部分を抜粋したものが印刷・配布されるので，連結リストによるレディーキューの管理方法について理解する．

**・課題14b get_mpl の内部処理を見てみよう**

HOSのソースコードから同じく，第11回の実習課題で用いたget_mplの処理の主要な部分を抜粋したものが印刷・配布されるので，連結リストによる可変長メモリブロックのメモリ領域の管理方法について理解する．

## 4. おわりに

本稿では，ソフトウェアサイエンス学科3年次または4年次の科目として開講されている「オペレーティングシステム」で実践しているオープンソースの組み込みOSを用いた実習について報告した．小規模なオープンソースOSをいくつか検討した結果，μITRONの仕様に基づいて開発されたHOSを用いることで，本学科の学生に適切な難易度とした上で，多少なりともOSの内部処理に迫るようなプログラミング実習が実施できたと考えている．今後は，ファイルシステムや仮想記憶など，現在HOSでは実現できていない実習の実現を目指していきたい．

## 参考文献


1) ゲーリー・ナット: 実習Linuxカーネル―理論と実習 カーネルを効率的に理解するための実習書, 浜田(訳), ピアソンエデュケーション, (2001).

2) A. S. タネンバウム 他: オペレーティングシステム（第2版）―設計と理論およびMINIXによる実装, 今泉(監), ピアソンエデュケーション, (1998).

3) (社)トロン協会 ITRON仕様検討グループ: μITRON4.0仕様 Ver. 4.02.00, (1999). http://www.ertl.jp/ITRON/SPEC/mitron4-j.html

4) Project HOS (渕上竜司 他): Hyper Operating System (ITRON仕様OS), (1998より公開). https://sourceforge.jp/projects/hos

5) 濱原和明: ITRONプログラミング入門―H8マイコンとHOSで始める組み込み開発, オーム社, (2005).

6) 塩澤秀和: オペレーティングシステム授業資料, (2017). http://vilab.org/lecture/?OS



2018年3月1日原稿受付，2018年3月14日採録決定

Received, March 1, 2018; accepted, March 14, 2018