教育用コンピュータのOSの設計

村田 侑暉[*1]　重村 哲至[*2]

Design of the OS for Educational Computer

Yuki MURATA[*1] and Tetsuji SIGEMURA[*2]

Abstract

In lectures of OS(Operating System) and compiler at college, these are often ending with the theories. But truly in order to understand them, it is preferable to perform practical learning with the system that actually works. So National Institute of Technology, Tokuyama College developed an educational computer of simple hardware configuration, in order to develop educational OS. This computer is called TaC. We show an example of the OS and compiler that implemented on TaC, and students can better understand OS and compiler. In this paper, we report the design of the educational OS that is mounted on TaC. TaC-OS is simple OS, so students can easily understand its mechanism.

Key Words : Operating System, Educational Computer, Educational OS

1　まえがき

高専や大学のコンピュータサイエンス系学科で行われているOSやコンパイラの授業は，理論の学習だけで終わることが多い．真にそれらを理解するには，実際に動作するシステムを教材とした実践的な学習を行うことが望ましいが，既存のシステムは，高機能かつ複雑で教材に適さない．

OSやコンパイラを勉強する学生向けに，教育用コンピュータが開発されている．代表的な教育用OS，教育用コンピュータとして，MINIX[1]とMieruPC[2]が挙げられる．MINIXは，本格的すぎて大規模なため限られた授業時間で取り上げるには向いていない．また，MieruPCは，小規模だが，言語処理系は対象外なためコンパイラの学習には向いていない．

そこで，徳山高専情報電子工学科では，教育用の分かりやすいOSを実装するために，単純なハードウェア構成の教育用パソコンTaC[3]を開発した．TaCは，OSやコンパイラの実装例を学生に示し，学生のOSやコンパイラに対する理解を深めることを目的とする．

本研究では，TaCに搭載する教育用OS(TaC-OS)の設計を行った．TaC-OSは小規模で学生が容易に仕組みを理解できるOSである．

図1　TaC

2　TaC

TaCは，OSやコンパイラの実装例を参照しながら学習を行うための教育用パソコンである．図1にTaCの写真を示す．TaCはキーボード，ディスプレイ，16bitのCPU，56kBの主記憶，μSDカード，デバッグ用コンソールを備えている．TaC用のシステム記述言語C--言語[4]で記述したOSやアプリケーションを実行することができる．また，デバッグ用コンソールを用いてOSの内部までトレース実行できる．

[*1] 情報電子工学専攻

[*2] 情報電子工学科

TaC が一般的な PC に類似していることを学生に実感させるために，学生が TaC 上でプログラムを作成し実行できる環境を目指している.TaC では二次記憶装置として FAT16 でフォーマットされたμSD カードを採用している．起動時に，ROM に配置されている IPL プログラムが,OS をμSD カードから主記憶に読み込み実行する.

3　TaC-OS

C--言語で記述された簡単な OS である．TaC の OS として動作するだけでなく，学生にソースコードを公開しOSの実装例を示すことも目的にしている.そのため,性能よりも分かりやすさを重視して設計した.

3.1　概要

TaC-OS の構造を図 2 に示す．学生が理解しやすいマイクロカーネル方式を採用した．機能ごとに 7 つの要素に分割されており，機能順に学習することができる.

カーネルはプロセスのスケジューリングやディスパッチ機能と，プロセス間通信（IPC）を提供する．プロセスマネージャ(PM)はプロセスの生成と終了を行う．ファイルシステム(FS)はファイルシステムの管理を行い，μSD に対するアクセスを行う．メモリマネージャ(MM)はメモリの管理を行う．初期化プロセス(INIT)はシェルプロセスをユーザプロセスとして起動する．アイドルプロセス(IDLE)は他に実行可能なプロセスがないとき，アイドルループを実行する.

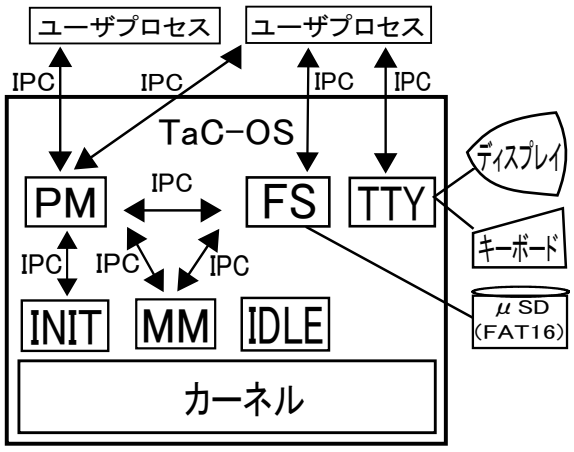

図 2　TaC-OS の構造

3.2　カーネル

カーネルには，必要最小限の OS の機能だけを実装している．カーネルは，スケジューリング，セマフォ，タイマー，IPC を提供する.

3.2.1　プロセス

TaC-OS のカーネルは，複数の並行プロセスを管理する．プロセスコントロールブロック(PCB)とカーネルスタックの関係を図 3 に示す.PCB はプロセスを表現するデータ構造体であり，プロセス毎に用意される．PCB の直後に，プロセスのカーネルスタックが配置される.

(1)　プロセスの状態遷移

プロセスの状態は,実行中または実行可能(P_RUN)，待ち（P_SLEEP），実行終了（P_ZOMBIE）のいずれかをとる．図 4 にプロセスの状態遷移図を示す．実行可能プロセスの 1 つが実行される．待ち状態との遷移はセマフォ操作のみで発生する.

(2)　プロセスコントロールブロック(PCB)

PCB 構造体の宣言を図 5 に示す．sp には，プロセスのカーネルスタックポインタのコピーを保存する．pid は，プロセス番号を示す． stat は，プロセスの現在の状態（P_RUN 等）を示す． nice，enice は，プロセスの優先度を示す.

evtCnt と evtSem は,プロセスのイベント用カウンタとセマフォである．これらは,タイマー等に使用する.

memBase と memLen は，ユーザプロセスのメモリ空間の開始アドレスと長さである

parent は,親プロセスの PCB を指すポインタである．exitStat には,プロセスが終了し P_ZOMBIE に移行する際に終了ステータスが格納される．終了ステータスは,親プロセスの wait システムコールが利用する.

fds はプロセスがオープン中のファイルの一覧である．exit システムコールが使用する.

prev と next は前後の PCB へのポインタである.magic はカーネルスタックのオーバーフロー検知に用いる.

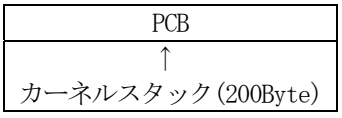

図 3　PCB とカーネルスタック

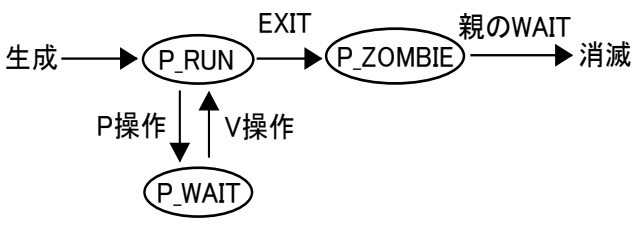

図 4　プロセスの状態遷移

```
struct PCB {      //PCB 構造体
  int sp;         //スタックポインタ
  int pid;        //プロセス番号
  int stat;       //プロセスの状態
  int nice;       //プロセスの本来優先度
  int enice;      //プロセスの実質優先度
  int idx;        //プロセステーブル上のインデクス
  int evtCnt;     //イベント用カウンタ
  int evtSem;     //イベント用セマフォの番号
  char[] memBase; //プロセスのメモリ領域のアドレス
  int memLen;     //プロセスのメモリ領域の長さ
  PCB parent;     //親プロセスへのポインタ
  int exitStat;   //プロセスの終了ステータス
  int[] fds;      //オープン中のファイル一覧
  PCB prev;       //PCB リスト（前へのポインタ）
  PCB next;       //PCB リスト（後ろへのポインタ）
  int magic;      //スタックオーバーフロー検知用
};
```

図 5　PCB

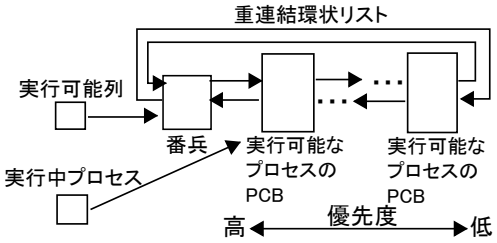

図 6　実行可能列

(3)　カーネルスタック・ユーザスタック

ユーザプロセスにはカーネルスタックとユーザスタックが用意される．ユーザスタックはプロセスのメモリ空間後端に配置され，サイズはプロセス毎に決定できる．カーネルスタックは例外や割込みが発生しプロセスがカーネルモードに移行した時に使用される．

(4)　スケジューリング

実行可能プロセスはスケジューラにより優先度順に図 6 に示す実行可能列に登録される．実行可能列の先頭プロセスは，その場に置かれたまま実行される．プロセスを切り換える時は，ディスパッチャが実行中プロセスのコンテキストをカーネルスタックに退避し，その後，列の先頭プロセスのコンテキストを復元する．

3.2.2　セマフォ

セマフォ操作は，P 操作，V 操作，ディスパッチを発生しない V 操作の 3 つである．図 7 にセマフォ構造体の宣言を，図 8 にセマフォの構造を示す．

```
struct SEM {      //セマフォ構造体
  boolean inUse;  //セマフォが使用中か
  int cnt;        //カウンタ
  PCB queue;      //待ち行列
};
```

図 7　セマフォ

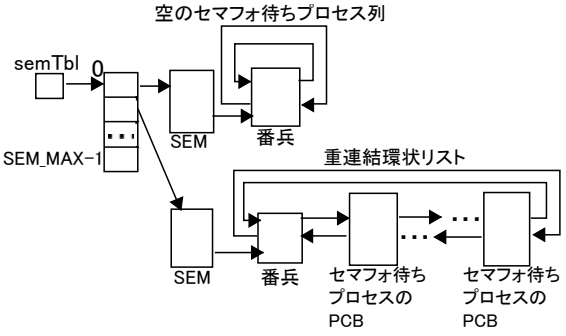

図 8　セマフォの構造

3.2.3　プロセス間通信(IPC)

TaC-OS の IPC は、ランデブ方式である．図 9 にランデブ方式 IPC によるシステムコールの様子を示す．ランデブ方式では，サーバは一度に 1 つのシステムコールしか受け付けないため分かりやすい．

図 10 に示すように，サーバにはそれぞれ 1 つのリンクが割り当てられる．リンクを介してクライアントとサーバ間でメッセージのやり取りを行う．図 11 にリンク構造体の宣言を示す．リンクは 3 つのセマフォを用いて排他・同期制御を簡潔に実現している．

3.3　サーバ

OS の機能の多くをサーバがシステムコールとして提供する．表 1 にシステムコールの一覧を示す．ディスプレイやμSD カードなどのハードウェア資源には，決められたサーバのみがアクセスするので排他処理の多くを省略できる．PM，FS，TTY，MM がサーバである．

サーバは、クライアントから IPC を用いてシステムコールを受信し処理結果を返す．システムコールは IPC により順次渡されるのでサーバの処理は単純である．

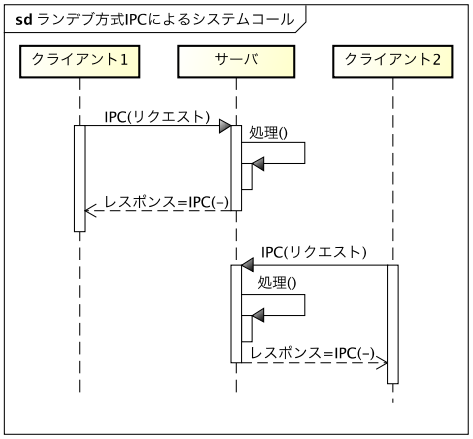

図9　ランデブ方式 IPC によるシステムコール

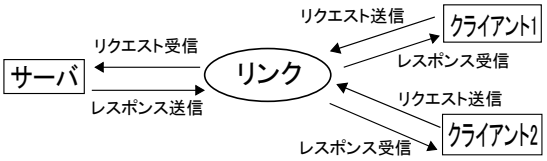

図10　サーバとクライアント間の IPC の様子

```
struct LINK { //リンク構造体
  PCB server; //リンクを所持するサーバ
  PCB client; //リンク使用中のクライアント
  int s1;     //リクエスト待ち用のセマフォ
  int s2;     //クライアント排他用セマフォ
  int s3;     //レスポンス待ち用のセマフォ
  int op;     //メッセージの種類
  int prm1;   //メッセージのパラメータ 1
  int prm2;   //メッセージのパラメータ 2
  int prm3;   //メッセージのパラメータ 3
};
```

図11　リンク構造体の宣言

3.3.1　メモリマネージャ（MM）

可変分割方式のメモリ管理を行う．空き領域はアドレス順のリストにして管理される．リストの各ノードは，空き領域サイズと次の空き領域アドレスを保持する．メモリの割り当てを行う malloc システムコール，メモリの解放を行う free システムコールを提供する．

(1)　malloc システムコール

ファーストフィット方式で要求されたメモリサイズの領域を割り当てる．リストをたぐる処理には時間がかかるが，IPC により要求は順次渡されるので割り込み許可状態で処理を行うことができる．

表1　システムコールの一覧

名前	機能	サーバ
exec	プロセスを生成し実行する	PM
exit	プロセスを終了する	PM
wait	子プロセスの終了を待つ	PM
creat	ファイルを作成する	FS
remove	ファイルを削除する	FS
open	ファイルをオープンする	FS
openDir	ディレクトリをオープンする	FS
close	ファイル･ディレクトリをクローズする	FS
read	ファイルから読み出す	FS
readDir	ディレクトリエントリを読み出す	FS
write	ファイルに書き込む	FS
seek	ファイルの参照位置を変更する	FS
conRead	キーボード入力を受け取る	TTY
conWrite	ディスプレイに出力する	TTY
malloc	メモリを割り当てる （注：カーネルプロセスのみ使用）	MM
free	メモリを解放する （注：カーネルプロセスのみ使用）	MM

(2)　free システムコール

解放領域を空き領域リストにアドレス順で追加する．

3.3.2　ファイルシステム（FS）

ファイルシステムの管理を行い，FAT16 でフォーマットされたμSD カードにアクセスする．ファイル操作に関するシステムコールを提供する．

(1)　creat システムコール

ファイルを作成する．作成するファイルの名前が適切か調べる．次に，ディレクトリの空きエントリを探索し，ファイルのディレクトリエントリを作成する．

(2)　remove システムコール

ファイルを削除する．削除するファイルがオープンされていなければファイルのディレクトリエントリを削除する．次に，ファイルが使用していたクラスタを解放する．

(3)　open システムコール

ファイルをオープンする．引数として与えられたパスを解析しファイルの存在を確認した後，オープン中ファイルの管理データ（図12，FILE 構造体）を生成する．生成したFILE構造体にファイルの情報を格納する．

(4)　openDir システムコール

ディレクトリファイルをオープンする．パスを解析し対象が存在するディレクトリファイルであることを確認する．その後の処理は open システムコールと同様である．

(5)　close システムコール

ファイルをクローズする．クライアントプロセスが

```
struct FILE {    //FILE 構造体
  char[] buffer; //ファイル毎のバッファ
  int bufPtr;    //buffer 上の現在位置
  boolean isDir; //ディレクトリかどうか
  int[] len;     //ファイルの長さ
  char[] name;   //ファイル名
  char[] ext;    //拡張子
  int mode;      //オープンモード
  int pid;       //オープンしたプロセスの PID
};
```

図 12　FILE 構造体の宣言

open または openDir システムコールでオープンしたファイルであることを確認し，FILE 構造体を解放する．

(6)　read システムコール

FILE 構造体から，読み込むデータが格納されているセクタアドレスを計算し，データを読み込む．

(7)　readDir システムコール

read システムコールと同様な処理を行いディレクトリから有効なディレクトリエントリを読みだす．

(8)　write システムコール

ファイルにデータを書き込む．FILE 構造体から，データを書き込むセクタアドレスを計算する．アドレスがクラスタ境界ならば，FAT 領域の空きエントリを探索しクラスタチェーンを拡張する．最後にディレクトリエントリのファイルサイズフィールドを更新する．

(9)　seek システムコール

ファイルの読み書き位置を移動する．ファイルの先頭クラスタから指定したバイト位置までクラスタをたぐる．次に，クラスタ内のセクタを特定しセクタの内容をバッファに復元する．

3.3.3　プロセスマネージャ（PM）

プロセスを生成する exec システムコール，プロセスを終了する exit システムコール，子プロセスの終了を待つ wait システムコールを提供する．

(1)　exec システムコール

実行ファイルの名前を引数として呼び出される．まず，open システムコールと read システムコールを利用して実行ファイルのヘッダを読み，プロセス生成に必要なメモリサイズを得る．次に，malloc システムコールを利用しプロセスのメモリ領域を確保し，read システムコールを用いて実行ファイルの内容を読み込む．更に，malloc システムコールを利用し PCB 等の領域を確保する．最後に PCB 等を初期化してプロセステーブルに登録しプロセスをスケジューリングする．

(2)　exit システムコール

終了ステータスを引数として呼び出される．まず，終了プロセスの PCB を確認しクローズし忘れているファイルをクローズする．次に，子プロセスの PCB の parent を INIT に変更する．最後に，メモリ領域を解放し，子プロセスの PCB の exitStat に終了ステータスを格納する．親プロセスが先に wait システムコールを実行しブロックしていれば，V 操作で起床させる．

(3)　wait システムコール

ゾンビ状態の子プロセスがあれば，PCB の exitStat から終了ステータスを取り出し，子プロセスの PCB を解放する．ゾンビ状態の子プロセスがなければ，イベント用セマフォを用いてブロックする．

3.3.4　端末サーバ（TTY）

キーボード入力された文字列をユーザプロセスに渡す conRead システムコールと，ユーザプロセスから受け取った文字列をディスプレイに出力する conWrite システムコールを提供する．

(1)　conRead システムコール

キー入力をバッファリングする．改行キーが入力されたらバッファの内容をユーザプロセスに渡す．

(2)　conWrite システムコール

ユーザプロセスから渡された文字列をディスプレイに出力する．

3.4　システムコールの発行手順

図 13 にシステムコールが実行される様子を示す．各サーバには，システムコールのスタブが定義してある．カーネルプロセスは，スタブを呼び出す（図 13(i)）ことで，IPC を開始する．IPC により要求を受け取ったサーバプロセスは，処理完了後，結果を返す．

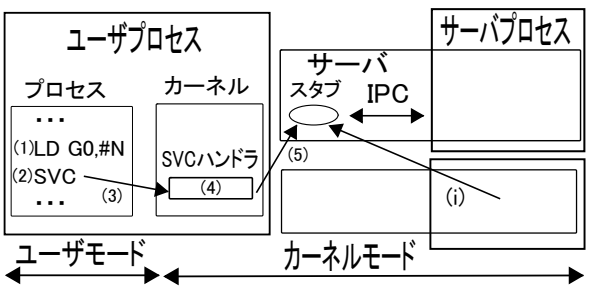

図 13　システムコールが実行される様子

ユーザプロセスは、システムコール番号をレジスタに設定(図13(1))後,SVC命令を実行する(図13(2)). SVC命令を実行すると例外が発生し,SVCハンドラにキャッチされる(図13(3)). ハンドラ内でレジスタ値からシステムコールを特定し(図13(4)), サーバのスタブを呼び出す(図13(5)).

3.5 スタートアップ

カーネルは, ハードウェアやセマフォを初期化した後、静的領域に配置したカーネルプロセスのPCBとカーネルスタックを初期化する. この段階ではmallocシステムコールを使用することができないので, カーネルプロセスのPCB等は静的領域に配置する必要がある. 次にカーネルは, スケジューラを用いて実行可能列にカーネルプロセスを登録しプロセスを開始させる.

サーバは, 待ち受け用のリンクを生成し, 要求待ちになる. 続いて開始されるINITは, execシステムコールを用いてシェルプロセスを実行する.

3.6 実行ファイル

実行ファイルは実行可能なアプリケーションプログラムを格納するファイルである. これはOBJEXEプログラムにより, 図14に示すリロケータブルファイルからシンボルテーブル等を取り除き作成する. ファイル形式が単純なため、PMのプロセス生成処理が簡単になりOSサイズを小さくすることができる.

図15に実行ファイルの構造を示す. テキストセグメントとデータセグメントにはプログラムの機械語と初期化データが格納される. 再配置情報には再配置対象アドレスの一覧がテキスト先頭を基準とした相対アドレスで格納されている. プロセスのメモリ空間にロードされたプログラムとデータは, 実行開始前に全ての再配置対象アドレスの内容にテキストの先頭アドレスを足し込まれる. これにより再配置が完了する.

図16に実行ファイルのヘッダ構成を示す. マジックナンバーは実行ファイルを識別するための固定値である. その他は, 各セグメントの長さを格納している. ヘッダからプロセスに必要なメモリのサイズが分かる.

4 まとめ

学生にソースコードを公開しOSの実装例を示すことを目的に, 教育用パソコンTaC用のOS(TaC-OS)を設計した. TaC-OSは, マイクロカーネル方式を採用し, 性能より分かりやすさを優先して設計されている.

ヘッダ
テキストセグメント
データセグメント
テキスト再配置情報
データ再配置情報
シンボルテーブル
文字列テーブル

図14　リロケータブルファイルの構造

ヘッダ
テキストセグメント
データセグメント
再配置情報

図15　実行ファイルの構造

マジックナンバー(0x0108)
テキストセグメントサイズ（バイト単位）
初期化データセグメントサイズ（バイト単位）
非初期化データセグメントサイズ（バイト単位）
リロケーションサイズ（バイト単位）
ユーザスタックサイズ（バイト単位）

図16　実行ファイルのヘッダ構成

カーネルには, 必要最小限の機能だけを実装し, その他の機能は, サーバがIPCを用いてシステムコールとして提供する. IPCはランデブ方式なため, サーバは1つずつしかシステムコールを受け取らないため分かりやすい.

プロセス生成の処理を簡単にするために単純な形式の実行ファイルを定義し, これをリロケータブルファイルから作成するOBJEXEプログラムを実装した.

今後の課題はTaC-OSの仕組みを学習するための教科書等を整備することである.

文献

1) A.S. タネンバウム他, “オペレーティングシステム：設計と理論およびMINXによる実装”, プレンティスホール出版, 1998.

2) 吉瀬謙二, “シンプルな計算機システムの開発に向けた挑戦”, 情報処理学会論文誌, Vol.54, No.7, pp. 1902-1912.

3) 重村他, “機械語教育用マイコンTeCとシリーズ化した教育用パソコンTaCのアーキテクチャ”, 情報処理学会研究報告書, Vol.2012-CE-117, No.9, pp. 1-7.

4) 重村他, “教育用システム記述言語 C--”, 教育システム情報学会研究報告, Vol.22, No.4, pp. 75-80, 2007.

(2015.9.24 受理)