システムソフトウェアと
オペレーティング・システム 92－6
（２００３．２．２７）

動的なコンフィグレーションを可能とした オペレーティングシステムの構成手法

†笠井秀一　††毛利公一　†並木美太郎

†東京農工大学大学院工学研究科
††立命館大学理工学部情報学科

本論文では，動的な変更を可能とするオペレーティングシステムの構成手法について述べる．本システムでは，メモリ管理，スケジューラ，ネットワーク処理などの，オペレーティングシステムの主要となる機能をモジュール化し，モジュールの追加，削除を可能にするシステムをユーザに提供する．特にOSの主要機能の一部であるスケジューラを目標とし，スケジューラの動的な変更を可能とするインタフェースの実装を行い，ユーザが本システムが提供するフレームワークに沿ってモジュールを構成することで，任意のスケジューラを構築することができるシステムを開発した．本論文では，それらの実装と性能評価について述べ，その有効性について議論する．

A Design of Dynamic Configurable Scheduler for an Operating System

†Syuichi Kasai　†Mitarou Namiki　††Kouich Mouri

†Graduate School of Engineering, Tokyo University of Agriculture and Technology
††Department of Computer Sicience, Faculty of Science and Engineering, Ritsumeikan University

This paper presents a design of an operating system, which enables dynamic changes of its main functions, such as network system, memory management and process scheduler.These functions are implemented as modules in the system, thus can be loaded and unloaded when it is needed.The design makes it prossible to implement most appropriate algorithms for specific applications and improve its performance significantly without reconstructing operating system.In this paper, an implementation of a process scheduler using the design is shown.We also conducted a performance evaluation on that and show the effectiveness of the design.

1.緒言

ネットワークインフラ，そしてアプリケーションが整備されていく中，それらの間に介在し，相互の同期などの処理を行うオペレーティングシステム(以下OS)にも様々なサービスや，アプリケーションの要求を処理できる機能が必要となっている．

このように，現在のコンピュータを取り巻く環境は，ネットワークを通した大容量通信，そして端末の小型化，それに伴うアプリケーションの増加によって，コンピュータに対する要求というものはますます複雑で正確なものとなっている．つまり，そのコンピュータ

の中心的役割である OS に, 大きな負担がかかっていることになる.

ユーザやアプリケーション開発者にとって, 機能に特化した OS をサービスに合わせて使用することは, 以下の問題が存在する.

(1) OS の資源管理モデルが必ずしも実装に一致しないため, カスタマイズ時の拡張部分の変更が困難である
(2) 単一のスケジューラでは, スケジューラの組替え時にプロセスすべてを移動させなければならず, また複数のスケジューリングアルゴリズムを使用するためには, それらを管理する機能が必要である
(3) すべての要求される機能をカーネルに組み込んでしまうと, PDA などのメモリ量が少ないハードウェアでは, メモリ使用量がシステムを圧迫してしまう.

上記の問題を解決するため, 本論文では Dynamic Kernel Configuration System (以下 DKCS) を提案する. 本論文では OS の主要機能の中で, スケジューラを目標として, スケジューラモジュールを構築するための OS アーキテクチャを定義した. 以下に DKCS の特徴を示す.

(1) スケジューラをモジュール化するためのインタフェース構築
開発者がモジュールとして作成したスケジューラやシステムコールをカーネルに登録するインタフェースを提供することで, スケジューラをカスタマイズすることができる
(2) Meta スケジューラによる複数スケジューラの共存
複数のスケジューラモジュールを組み込むことにより, 複数のスケジューラを共存させ, それらを Meta スケジューラによって管理することでプロセス毎に違ったスケジューリングアルゴリズムを適用することができる
(3) 計算機資源の節約
オンデマンドで必要なモジュールを組み込むため, 不必要な機能をカーネルに組み込むことが無く, リソースの削減につなげることができる

本論文では, 2 章で関連研究について述べ, 3 章で Dynamick Kernel Cnfiguration System の全体構成と特徴を述べる. 4 章で設計について述べ, 5 章で実装について述べる. 6 章でシステムの評価を行い, 7 章で締める.

2.既存のシステムの問題点

現状の LinuxOS では, カーネルの機能を動的に拡張する機構として Loadable Kernel Module[1](以下 LKM)がある. LKM は, Linux に実装されている, 動的にカーネルの機能を拡張する機構である. 機能を拡張できるものとして, 現在ではデバイスドライバ, ファイルシステム, ネットワークプロトコルなどがモジュールとして動的にカーネルにロードすることができる.

しかし, LKM で登録できるモジュールというのは現段階では制限があり, 一般的にはデバイスドライバモジュールを組み込むことに使用されているだけである. 何種類もあるデバイスドライバを全てカーネルに組み込まずに, オンデマンドでドライバを組み込むというそのシステムは有用性が高いが, それに限定されてしまっているのが現状である. OS の主要な機能を拡張するための機構として, Mach[2]の外部ページャ[3][4]があげられる. マイクロカーネルは, OS の構成技法のひとつで, Linux などのモノリシックカーネルと対極をなし, OS の API やスケジューリングなどのサービスの機能をサブシステムとしてカーネルの外に出す
ことで, それらの変更や複数の実装を可能としているシステムである.

しかし, あくまでページャに特化しているため, その他の機能を外部に追い出すといったことは行われていない.

Universal Scheduling System [5](以下 USS)はスケジューラに特化して, OS のアーキテクチャを変更するものである. ポリシとメカニズムの分離の概念により, スケジューリングに必要となる汎用的・共通的機能をメカニズムとしてシステムが提供し, アプリケーション固有の処理をポリシとして簡単に実現するためのインタフェースを

提供する手法である.

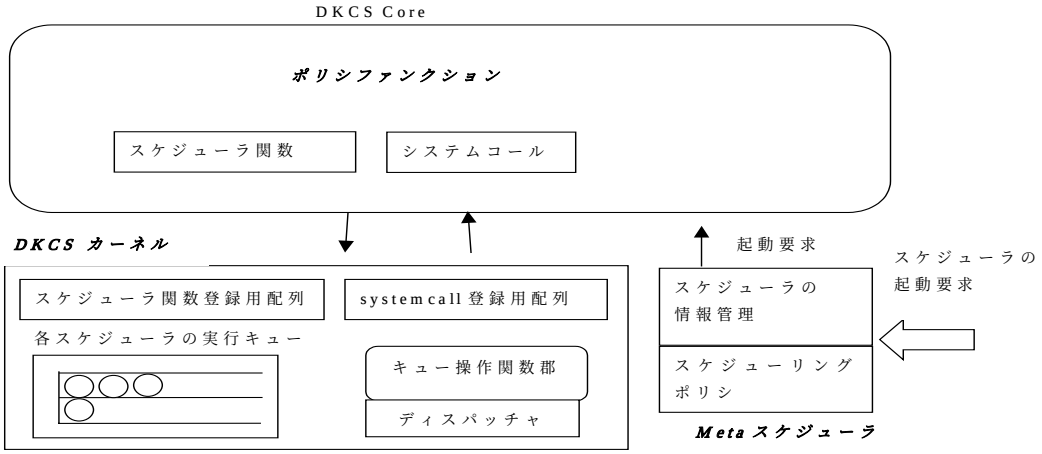

図1 DKCSの全体構成

しかし，これは複数のスケジューリングアルゴリズムを共存させることは考慮されていない．また，概念的なモデルでの提案なので，実装に移したときに，管理モデルと実装のギャップが生じ，カスタマイズを困難としている.

3. DKCSの全体構成と特徴

本論文では，上記の問題点から以下のことを目標として，Dynamic Kernel Configuration System(以下DKCS)を開発した.

- USS のスケジューラをターゲットとしてOSを変更できるシステムをベースに，スケジューリングアルゴリズムを任意に作成し，組み込めるシステムを開発する
- 複数のスケジューラを共存できるシステムを開発する
- モジュールを用い，オンデマンドで組み込めるシステムを開発する

ここでDKCSの特徴となる処理として，以下の二つが挙げられる.

- カーネルの主要機能のモジュール化
- モジュール化するためのインタフェースの定義

DKCSは，スケジューラ，メモリ管理，ネットワークといった OS の主要機能をモジュール化するためのインタフェースを提供することを目標としている．インタフェースはモジュールの種類ごとに定義され，ユーザはそれらインタフェースにしたがってモジュールを構築することで，任意にカーネルの主要機能をモジュールとして構築することができる.

本論文では，スケジューラにターゲットを絞ったDKCSインタフェースを構築し，スケジューリングアルゴリズムを任意にモジュールとして作成し，カーネル内に登録することが可能なインタフェースの構築を行ったことを述べる.

3.1 DKCSスケジューラ

DKCSスケジューラは大きく分けて以下3つに構成を分類することができる（図1参照).

- Metaスケジューラ
- DKCS Core
- DKCSカーネル

Metaスケジューラは複数スケジューラを共存させることを前提としている DKCS において，その各スケジューラの管理を担っている．スケジューラをスケジューリングするためのスケジューリングポリシ，また各スケジューラのプライオリティなどの情報を管理する．Metaスケジューラにより，新たに登録したスケジューラの管理，そしてスケジューラのスケジューリングを統一して扱えるようになる.

DKCS Core では，モジュール作成ユーザが任意に作成したスケジューラ関数をモジュール構築する．スケジューラを構築し，DKCSか

ら提供されるカーネルへのレジストリのための関数を使用し，カーネル内へとそれら関数を組み込むことができる．

スケジューラの DKCS Core は，DKCS カーネルで提供されるインタフェースに沿って以下のことを定義しておく．

- スケジューリングアルゴリズムの定義
- ユーザに提供するシステムコールインタフェースの定義
- スケジューリングアルゴリズムで使用される処理の定義

上記のフレームワークにしたがって，ユーザは任意のスケジューリングアルゴリズムを作成することが可能となる．

DKCS カーネルでは，モジュールで定義された関数をカーネルの登録するための関数の提供や複数スケジューラが混在するために必要なシステムなど，DKCS スケジューラインタフェースを提供する．以下にそのインタフェースの特徴を挙げる．

- スケジューラ関数，システムコール関数をレジストするための関数の提供
- スケジューラ関数を構築するためのカーネルパラメータの提供

上記の提供される関数，パラメータを用いてモジュール開発者は任意にスケジューラモジュールを構築することができる．提供されるカーネルパラメータは後述する．

4.DKCS の設計

4.1 Meta スケジューラの設計

Meta スケジューリングでは，各スケジューラの情報をチェックし，どのスケジューラを次に起動するかをスケジューリングする．スケジューリングする対象となるものは，スケジューラ関数とその優先度のセット（以下スケジューラ構造体とよぶ）をスケジューリングする．スケジューラ構造体の優先度を比較し，もっとも優先度の高いスケジューラを最優先で実行する(図 2 参照)．Meta スケジューラでは以下の特徴を持つ．

- 各スケジューラの優先度に基づいたバックグランドスケジューリングによるスケジューラのスケジューリング
- デバイスなどからスケジューラが呼び出されるタイミングで Meta スケジューラを起動させる
- リアルタイムスケジューリングの実行キューにプロセスがある場合，最優先で実行される

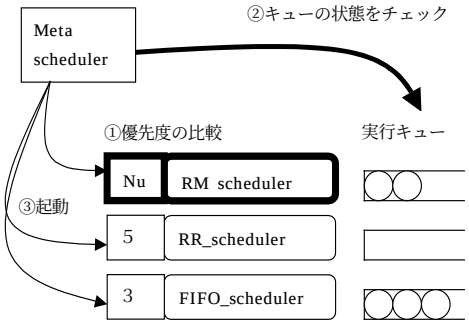

図 2　Meta スケジューリングの流れ

リアルタイムスケジューリングアルゴリズムは，一般に他のリアルタイムスケジューラとの混在は難しい．そこで，DKCS スケジューラではシステムにはリアルタイムスケジューラは一つとし，後のスケジューラは非リアルタイムスケジューラとする．

図 2 から，まず Meta スケジューラは各スケジューラに設定された優先度を比較し，起動対象となるスケジューラを選択する．リアルタイムスケジューラについては最高優先度のため，まず始めに起動対象となる．次に起動対象となったスケジューラの実行キューを参照し，実行可能プロセスがあればそのままスケジューラを起動させる．無い場合には次に高い優先度を持つスケジューラを起動対象とし，同様の処理を行う．

4.2 DKCS カーネルの設計

DKCS Kernel では，DKCS Core とカーネルとを結びつけるインタフェースを，モジュール開発者に提供し，モジュールの作成，登録，削除を行うためのインタフェースを提供する．モジュール開発者に提供するインタフェースとして以下の 3 つがある（表 1,2,3 参照）．モジュール開発者は，モジュール内で記述したスケジューラ関数と，ユーザプログラムに

提供するシステムコールを，上記の関数を用いてカーネルに登録する．これにより，モジュール開発者からカーネル内部の設定を隠蔽し，モジュール追加の処理を単純化することができる．また，モジュール開発者に，スケジューラを構築するために提供されるパラメータは以下のとおりである（表4参照）．
下記のパラメータを使用し，ユーザは任意のスケジューラを作成することが可能となる．

表1 add_schedule

形式	add_schedule
引数	int pri
	int label
	void (*sched)(void)
機能	
スケジューラ関数をカーネルに登録する．その際，Meta スケジューリングで必要となる，スケジューラの優先度を設定することができる．	

表2 delete_schedule

形式	delete_schedule
引数	int label
機能	
スケジューラの削除を行う	

表3 add_syscall

形式	add_syscall
引数	int label
	int (*func)(void)
機能	
システムコールの追加を行う．	

表4 スケジューラ構築のための
パラメータ，関数

カーネルから提供	現在のプロセスへのポインタ
	RUN キューへのポインタ
	ディスパッチ関数
プロセスから提供	周期
	デッドライン
	優先度
	次回起動時間

4.3 システムコールインタフェース

組み込んだモジュールのサービスをアプリケーション開発者が容易に利用するために，統一されたシステムコールをプログラム開発者に提供する．以下にそのシステムコールの処理の流れを図3に示す．

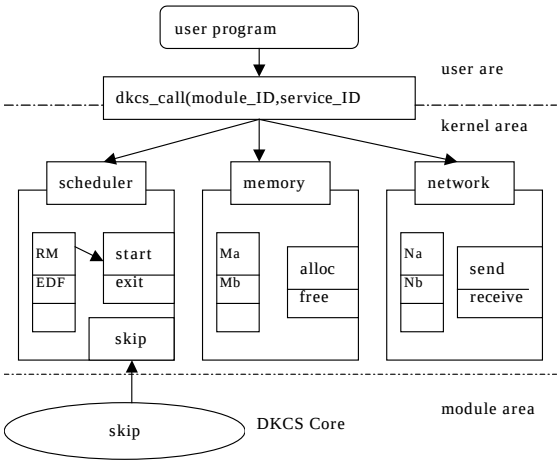

図3 dkcs_call の流れ

dkcs_call によって，モジュールで登録したシステムコールや，各モジュール共通のシステムコールを統一的に呼び出すことが可能となる．以下に仕様を記す．

表5 dkcs_call の仕様

形式	DKCS_call
引数	int module_ID
	int service_ID
	int function
	struct pnp_param *

DKCS では，将来的にスケジューラモジュールだけでなく，ネットワークやメモリ管理モジュールなども扱うことを考えている．それらを実装した場合，サービスを使用する上でシステムコールを統一して扱うことができるようにすることで，引数の識別だけで各モジュールのサービスを使用することができるようになる．

4.4 モジュールの構築例

上記で述べた関数，パラメータを用いてスケジューラモジュールの構築例を示す．スケジューリングアルゴリズムに優先度ベーススケ

ジューリングを適用したスケジューラモジュールの構築例を以下に示す．

```
#define PRI 1
my_schedule()
{
int temp_pri=-999;
struct task_struct *temp,*next,*prev;
 prev = current;
 temp=&(init_task_my_sched->task_next);
for( ;temp!=init_task; temp=temp->task_next)
      if(next->pnp->priority > temp_pri){
         temp_pri = next->pnp->priority;
         next=temp;
         }
}
         dispatch(prev,next,prev);
}
init_module()
{ add_schedule(PRI,my_schedule); }
delete_schedule()
{ delete_schedule(PRI);}
```

図 4 スケジューラモジュールの構築例

ここでは，現時点でのカレントタスクのポインタ(current)，各スケジューラ毎の RUN キューへのポインタ(init_task_my_sched)，そしてディスパッチ関数(dispatch)を用いてスケジューラ関数を構築し，登録，削除用関数である add_schedule,delete_schedule によってスケジューラ関数の登録，削除をプログラムしている．

5．実装

本システムは Linux カーネル 2.2.14 をベースに実装を行った．Linux ではデフォルトスケジューラに RR や優先度など，3 つのスケジューリングアルゴリズムを実装している．本スケジューラでは，Meta スケジューラによるデフォルトスケジューラの優先度をどのスケジューラよりも最低とし，他のスケジューラに実行可能プロセスが存在しない場合のみ，デフォルトスケジューラを起動する．

5.1 Meta スケジューラの実装

Meta スケジューラの動作は，バックグランドスケジューリングを基本とした優先度ベースのスケジューリングとした．各スケジューラモジュールのスケジューリングは，スケジューラ構造体(図 5 参照)を対象としてスケジューリングを行う．スケジューラ構造体は，add_schedule 関数内でその領域の確保，初期化を行う．同様に delete_schedule で領域を開放する．

```
struct sched_struct
{
  int pri; //優先度
  int label;//スケジューラ探索用ラベル
  void (*sched)(void); //スケジューリング関数
  struct task_struct *run_queue_head; //runqueue
  struct task_struct *wait_queue_head;
  void (*sched)(void)//スケジューラ関数
  struct sched_struct *prev_sched;//双方向リスト
  struct sched_struct *next_sched;
}
```

図 5 スケジューラ構造体

5.2 DKCS カーネルの実装

DKCS スケジューラでは，タスク構造体に新たな DKCS 用の領域を確保し，その領域に一般スケジューラで扱うパラメータとは別の，DKCS スケジュールで使用するパラメータを設定する（図 6 参照）．

```
struct pnp_param{
  int     policy;   //スケジューリングポリシ
  unsigned long     period; //周期
  unsigned long     priority;//優先度
  unsigned long     next_start_time;次周期
  struct task_struct *skip_queue_next;
  struct task_struct *skip_queue_prev;
  }
```

図 6 スケジューリングパラメータ構造体

タスク構造体では，スケジューリングに必要なパラメータを，ユーザがセットする．周期，プライオリティ，スケジューリングポリシをユーザがセットし，DKCS_call システムコールを通じてカーネルに渡す．

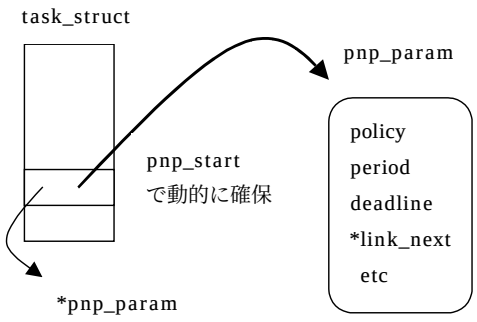

図 7 pnp 領域の確保

タスク領域は，一般のタスクでは不必要であり，タスクすべてに pnp 領域を作成すると，使用されない，無駄な資源を確保してしまうことになるため，オンデマンドで領域を確保し，pnp 領域が必要のないタスクには領域を割り当てない．(図 7 参照)．

5.3 DKCS モジュールの実装

本論文では，モジュール作成の例としてレートモノトニックアルゴリズムを適用したスケジューラをモジュールとして作成した．図４の作成例を元に，周期をスケジューリングパラメータとしてスケジューリングを行う．RM スケジューリングは，周期の短いタスクに対して高優先度を割り与える優先度ベーススケジューリングであり，リアルタイム OS ではよく用いられるスケジューリング方式である．

6．評価

6.1 実験

実験では，動画再生アプリケーションを動作させながら，CPU に負荷を掛ける．そのときの動画再生アプリケーションのフレームレートの低下を，RM スケジューラを用いた場合と，通常スケジューラでプロセスをスケジューリングした場合で測定し，評価を行う．結果を表

- 動画再生プロセスを二つ同時に立ち上げる
- 負荷生成プロセスを 1 個から 10 個まで立ち上げた場合の，MPEG-PLAY のフレームレートを計測する

まずデフォルトスケジューラにおける，フレームレートの測定結果を表 6 に示す．この結果が示す通り，プロセス一つで行った実験と同じようにプロセスが二つになっても，やはり負荷プロセス数が増加するごとにフレームレートの低下が起こった．

一方，RM スケジューラにおいて動画再生プロセスを動作させた場合(表 6 参照)，負荷プロセス数の増加に関わらず，フレームレートは一定の周期を保っており，低下することは無かった．

6.2 考察

この結果により，以下のことがわかる．

- 複数プロセスを RM スケジューラでスケジューリングしても，Meta スケジューリングのバックグランドスケジューリングが正確に動作している
- RM スケジューラで複数プロセスを，周期を保って動作させることができたことで，RM スケジューラとして正確に動作していることを確認できた

図 6 default スケジューラによる

	default scheduler		RM scheduler	
負荷プロセス数	プロセス A	プロセス B	プロセス A	プロセス B
0	24	24.1	33	33
1	22.9	22.6	33	33
2	18.5	18.4	33	33
3	16.2	15.7	33	33
4	13.3	13.4	33	33
5	11.8	11.8	33	33
6	10.3	10.2	33	33
7	9.7	9.8	33	33
8	8.5	8.5	33	33
9	8.2	8.1	33	33
10	7.7	7.8	33	33

以上の結果より，RM スケジューラが正確に動作し，Meta スケジューリングもそのスケジューリングポリシに沿って動作していることがわかった．

7．結言

動的なコンフィグレーションを可能としたオペレーティングシステムの構成手法を提案し，その実装と評価について述べた．
以下に研究を通じて得られた成果について述べる．

(1) スケジューラをモジュール化するためのインタフェースを構築した

OS の core な部分は，システムと密接に関係しているため，カスタマイズするためのインタフェースの定義が難しい．そこで，

スケジューラのスケジューリング関数が一つの関数で設計できることに着目し，スケジューラを目標として，スケジューリングアルゴリズムをモジュール化し，オンデマンドで組み込める Dynamic Kernel Configration System を設計，実装した．

(2) Meta スケジューラを設計，実装することにより，複数スケジューラを登録，そして管理できるようにした

システムに一つのスケジューリングアルゴリズムでは，その機能の変更が容易ではなく，また複数のスケジューリングアルゴリズムを使用するには，それらを管理する機能が必要であった．そこで本論文では，複数のスケジューラを登録，管理する Meta スケジューラを設計，実装した．

今後の課題として，メモリ管理やネットワークを目標とした DKCS インタフェースの構築が挙げられる．本論文で実装したスケジューラモジュールと組み合わせてメモリ管理モジュールやネットワークモジュールを組み込むことができるようになれば，さらにカスタマイズ性に富んだ OS の構築が可能となり，アプリケーションの要求に対しても広範囲で答えることが可能と思われる．

参考文献：

[1] Linux Loadable Kernel Module HOWTO http://www.tldp.org/HOWTO/Module-HOWTO

[2] J.Boykin, D.Kirschen 他著， 岩本信一訳 “Mach オペレーティングシステム”, Dec.1994.

[3]Keith Loepere., “Mach 3 Kernel Principles”, Open Software Foundation and Carnegie Mellon University, Jul.1992.

[4] Acceta, M., Baron, R., Golub, D., Rashid, R., Tevanian, A. and Young, M. “*A New Kernel Foundation for Unix Development,* “ 1986 Summer USENIX Conferrence, pp.93-113 1986.

[5] Ruschitzka, M. and Fabry, R. ”*A Unifying Approach to Scheduling,* “ Comm, ACM, Vol.20, No.7, pp.469-477 1977