システムソフトウェアと
オペレーティング・システム 74−35
マルチメディア通信と分散処理 81−35
（1997. 2. 28）

プロセスのグループ化による スケジューリングとファイルのアクセス制御方式

藤枝隆行　　新城靖　　板野肯三

筑波大学大学院
理工学研究科

筑波大学
電子・情報工学系

筑波大学
電子・情報工学系

UNIXシステムにおいて、スケジューリングは、個々のプロセスを対象として行われ、ファイルのアクセス制御は、ユーザを基準に行われている。本論文では、UNIXシステムに対して、プロセスの集合であるプロセスグループの概念を導入し、これに対応したスケジューラ、およびファイルのアクセス制御方式を提案する。複数のプロセスからなるアプリケーションプログラムを一つのプロセスグループにまとめることで、個々のアプリケーションプログラムが一つのまとまった優先度を持ち、アプリケーションプログラム毎のスケジューリングが可能となる。また、個々のファイルに対して、アプリケーションプログラム毎の動的なファイルアクセス権が設定可能となる。

Scheduling and file access control based on grouping process

Takayuki Fujieda, Yasusi Shinjo and Kozo Itano

Graduate School of
Science and
Engineering
University of Tsukuba

Institute of
Information science
and Electronics
University of Tsukuba

Institute of
Information science
and Electronics
University of Tsukuba

In the UNIX system, the kernel schedules individual processes, and file access control is done on the basis of user. In order to give a facility of grouping to a multi-process application in UNIX, we propose a facility of process group which groups processes into a single virtual process. The virtual process possesses schedulable eligibility, and can behave as an ordinary process in UNIX, and the component processes are scheduled within the virtual process. Another feature of the virtual process is the file access control based on the dynamic protection mechanism. This mechanism provides application specific file protection.

1 はじめに

UNIX システム [1, 2] では、一つのアプリケーションプログラムを複数のプロセスで構成することがしばしば行なわれる。このとき、スケジューリングとファイルアクセス制御に関する次のような問題がある。

1. アプリケーションプログラムに割り当てられるCPU 時間は、それを構成するプロセスの数に従って増える。
2. 特定のファイルへのアクセスを特定のプロセスのみに限定するようなアクセス制御を行なうことができない。
3. 異なるユーザにより実行された、同一種類のプロセスが同一のファイルにアクセスする場合に、アクセス権を同一に扱うことができない。
4. 特定のプロセスに対するファイルのアクセス制御を一時的に変更することや、その対象となるプロセスを動的に変更することはできない。

本論文では、このような問題を解決するために UNIX システム (BSD/OS、FreeBSD) に**プロセスグループ**の概念を導入する。PG とは、複数のプロセスを取りまとめて扱うためのしくみである。

この PG に対応したスケジューリング方式と、ファイルのアクセス制御方式を提案する。これらの方式によって、アプリケーションプログラム毎のスケジューリングや、動的なファイルのアクセス制御が可能となり、UNIX システムにおける上記の問題は解決される。

2 プロセスグループ (PG)

2.1 PG の概念

PG(Process Group) は、複数のプロセスで構成されるアプリケーションプログラムに含まれるプロセスを一つの単位として取り扱うためのしくみである。PG は、システムコールによって必要となった時に作成でき、必要ではなくなった時には、PG の削除を行なうことができる。任意のプロセスを PG のメンバとすることができ、また、PG のメンバであるプロセスを PG から削除することができる。

PG の制御を行なうためのシステムコールを追加した。次に、これらのシステムコールについて述べる。

createpgs() PG の作成を行なうシステムコールである。システムコールが成功した場合には、戻り値として PG 識別子を返す。

deletepgs() 指定された PG の削除を行なうシステムコールである。

bindpgs() 指定されたプロセスを指定された PG に登録するシステムコールである。

unbindpgs() 指定されたプロセスを指定された PG から削除するシステムコールである。

2.2 グループ化プロセス (GVP)

システム内部では、PG の取り扱いを容易にするために**グループ化プロセス** (Grouped Virtual Process,**GVP**) と呼ばれる概念を導入する。

グループ化プロセスと PG は、一対一に対応する。グループ化プロセスは、普通のプロセスと同様に、優先度や、PID を持っている。それに加えて、PG 構造体へのポインタも持っている。しかしながら、普通のプロセスとは異なり、メモリ管理、同期、シグナル、タイマ管理に関する情報は持っていない。

ここでは、グループ化プロセスに割り当てられた PID を、PG 識別子と定義する。PG 識別子は、PG の識別を行なうために使用されている。

2.3 PG 構造体

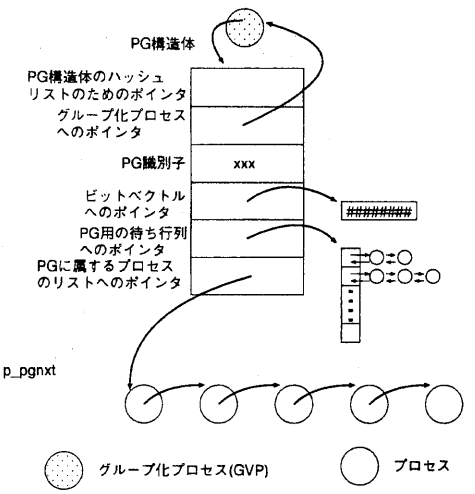


図 1: PG の内部表現

システム内部では PG を図 1に示すような PG の構造体によって表現している。この PG 構造体は、グループ化プロセスへのポインタ、PG に属するプロセスのリストへのポインタ、PG 識別子、PG 構造体のハッシュリストを作るために使われる PG 構造体へのポインタと、PG 用の走行プロセス待ち行列とその状態を表すビットベクトルへのポインタから構成される。

この構造体は、createpgs() システムコールが呼ばれた時に割り当てられる。このとき、GVP、PG 識別子、PG 用の待ち行列と、ビットベクトルも同時に割り当てられる。また PG 構造体は、deletepgs() システムコールが呼ばれた時に解放される。このとき、GVP、PG 識別子、PG 用の待ち行列と、ビットベクトルも同時に解放される。

3 PG のスケジューラ

この章では、PG のスケジューラのモデルとその実現方法について説明する。

3.1 PG のスケジューリングモデル

従来の UNIX システム [1, 2] では、の多段階フィードバック列に基づいたプロセススケジューリングモデルを

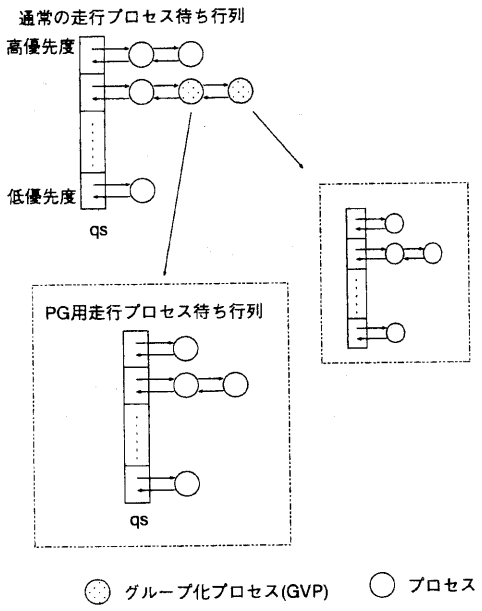


図 2: 走行可能プロセス待ち行列の構造

利用している。本システムでは、PGが取り扱えるように従来の UNIX のモデルを拡張する。

この論文では、混乱を避けるために、PGに属していないプロセスを、通常のプロセスと呼ぶことにする。

図2に示すように、PGに属するプロセスと通常のプロセスでは、異なる行列に存在する。グループ化プロセスと PG の作成の目的の一つが、通常のプロセスは、通常の走行プロセス待ち行列内に存在する。PGに属するプロセスは、そのPG構造体にある走行プロセス待ち行列上に存在する。スケジューラは、はじめ通常の走行プロセス待ち行列から実行可能な通常のプロセス、またはグループ化プロセスを選択する。スケジューラがグループ化プロセスを選択した場合は、このグループ化プロセスに対応する PG 構造体上の走行プロセス待ち行列から実行可能な PG に属するプロセスを選択する。このような方式を導入することによって、PGを単位としたスケジューリングを行なうことができる。

3.2 PG の優先度

図2に示すようにスケジューラは、グループ化プロセスを対象にしてスケジューリングを行っている。すなわちこのグループ化プロセスの優先度が PG 全体の優先度となっている。このような構造にすることによって、複数のプロセスから構成されるアプリケーションプログラムが一つの優先度を持つことができる。

UNIX では、プロセスのスケジューリングの優先度は、プロセス構造体中の二つの値 *p_estcpu* と *p_nice* によって決定される [1, 2]。通常の *p_nice* の値は、0 であり、負の値がプロセスの優先度を上げて、正の値が優先度を下げる。通常プロセスでは、setpriority() システムコールを用いて *p_nice* の値を設定し、スケジューリング優先度を制御することができる。

PG の優先度を制御するためにこの機構を使用する。PG の作成時において PG の優先度を制御できることが望ましい。従って、PG を作成するシステムコール (createpgs()) の引き数を指定することによって、グループ化プロセスの *p_nice* を設定できるようにした。また、一度設定した p_nice の値を、setpriority() システムコールを用いて制御することも可能である。また、グループ化プロセスの p_estcpu の値を、それに対応する PG に属するプロセスの p_estcpu を加算した値とするようにした。これによって、グループ化プロセスの p_estcpu の値が、その PG に属するプロセス全体の CPU 利用状況を示す。

3.3 PG スケジューリングの効果

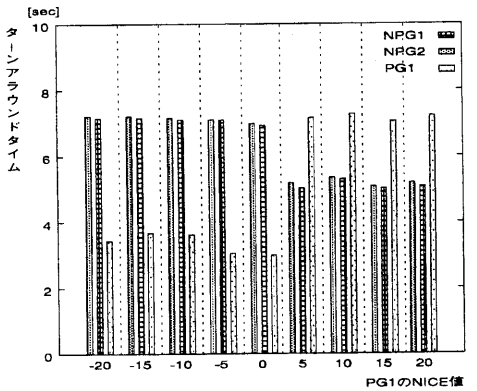


図 3: プログラムのターンアラウンドタイム

ここでは、プログラムに含まれるプロセスを、PG の機能を使ってまとめることを、PG 化と呼ぶことにする。

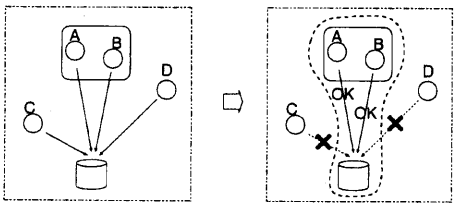
PG 化を行なったプログラムを実際に走行させて、PG スケジューリングの効果を調べる実験を行なった。実験で用いたプログラムは、2つのプロセス間で、ソケットを用いて通信を行うものである。実験に使用したのは、AT Compatible マシン (Pentium 100MHz, 24MB, 1.6GB) である。このマシン上で、このプログラムを 3 つ同時に走行させた。その中の一つのプログラムを PG 化し、その nice 値を変化させ、それぞれのプログラムのターンアラウンド時間を測定した。

図3は、この結果をグラフに表したものである。PG1 は、PG 化を行ったプログラムである。NPG1 と、NPG2 は、PG 化を行なっていないプログラムである。図3のグラフにおいて、PG1 の nice 値が、-20 から-5 の時は、PG1 の優先度が、NPG1、NPG2 のプログラムを構成するプロセスよりも高い。これにより PG1 が優先的に実行され、PG1 のターンアラウンド時間が、NPG1、NPG2 のターンアラウンド時間より短くなる。逆に、PG1 の nice 値が、5 から 20 の時は、PG1 の優先度が、NPG1、NPG2 のプログラムを構成するプロセスよりも低い。これにより、NPG1、NPG2 の方が優先的に実行され、NPG1、NPG2 のターンアラウンド時間が短くなる。PG1 の nice 値が 0 のときは優先度に差がないが、

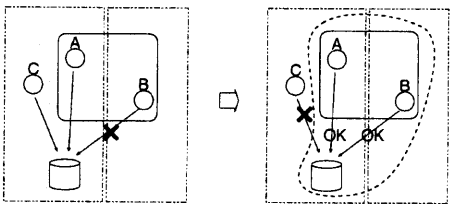
PG1のターンアラウンド時間が、NPG1、NPG2のターンアラウンド時間より短い。これは、PGを利用したことによってPG1を構成するプロセスが実行可能でありかつ、量子時間(time quantum)が残っている間は、PG1を構成するプロセスが実行されるので、PG1が有利に実行されるためである。

これにより、アプリケーション毎に一つの優先度を与えることができたことがわかる。

4 PGを用いたファイルアクセス制御



(a) ファイルアクセスの区別化



(b) ファイルアクセスの同一化

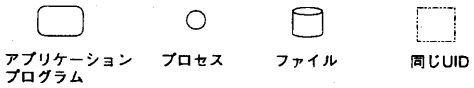


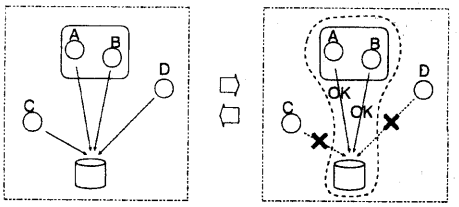
図 4: ファイルアクセスの区別化と同一化

UNIXには、1章で述べたようなファイルアクセス制御の問題があった。この問題を解決する一つの方法として、PGを用いたファイルアクセス制御を提案する。

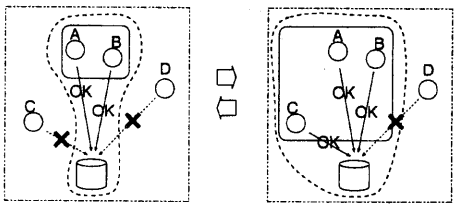
このPGを用いたファイルアクセス制御によって、次のようなことが実現することを目標にする。

ファイルアクセスの区別化 同一のUIDを持つプロセスの中でプロセスAとBを一つのアプリケーションプログラムとして構成して、これらのプロセスのみファイルへのアクセスを許可し、その他のプロセスからのアクセスを拒否する。(図4(a))

ファイルアクセスの同一化 ファイルのUIDと同一のUIDを持つプロセスAと、そのファイルのUIDと異なるUIDを持つプロセスBを一つのアプリケーションプログラムとして構成して、これらのプロセスのみファイルへのアクセスを許可する。(図4(b))



(a) ファイルアクセスの区別化、同一化の一時的設定



(b) ファイルアクセス可能なプロセスの動的変更

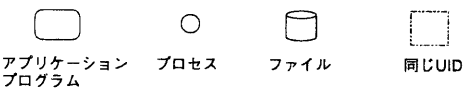


図 5: 動的なアクセス権の設定

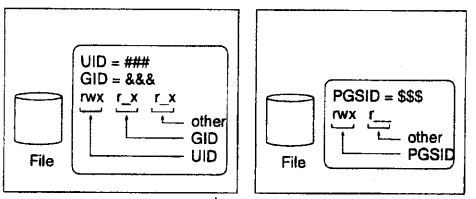


図 6: PGファイルアクセス制御方式におけるファイル所有と属性

ファイルアクセス制御の一時的設定 ファイルアクセスの区別化や同一化の設定を、一時的に行なう。(図5(a))

ファイルアクセス可能なプロセスの動的な変更 ファイルアクセスの区別化や同一化において、アクセスを許すプロセスを後から変更できる。(図5(b))

4.1 PGファイルアクセス権と所有者

ここで提案するファイルアクセス制御は、従来のユーザを基準とするファイルアクセス制御に加えて、PGを基準としたファイルアクセス制御を行えるようにしたものである。図6に示すように、各ファイルに従来のUNIXのファイルの属性に加えて、PGを用いたファイルアクセス制御のための属性を付加する。この属性の付加は動的に行なえる。この属性を**PGファイルアクセス権**と呼ぶことにする。

この属性は、PG識別子と、6bitのモードからなってい

る。この6bitのうち最初の3bitは、PG識別子(PGSID)で指定されるPGに属するプロセスのアクセスに対する読込み、書出し、実行の許可ビットであり、残りの許可ビットは、このPG以外のプロセスに対する読込み、書出し、実行の許可ビットである。

従来のUNIXのファイルアクセス制御では、ファイルの所有者とはファイルの属性UIDに対応したユーザである。このUIDと一致するUIDを持つプロセスは、所有者としてのアクセス権でファイルを操作することができ、かつ、ファイルのモードを変更することもできる。これに対して、PGを用いたファイルアクセス制御では、ファイルの所有者とは、ファイルの属性PGSIDに対応したPGである。このPGSIDと一致するPGSIDをもつプロセスは、所有者としてのアクセス権でファイルを操作することができ、かつ、ファイルのPGアクセス権を変更することもできる。

ファイルの所有者であるPGは、そのメンバプロセスが動的に変化し、また、このPG自体も、動的な存在であるという特徴を持っている。

4.2 PGファイルアクセス制御のためのシステムコール

ファイル対してPGファイルアクセス権を設定したり、そのファイルのアクセスモードや所有者を変更するための次のようなシステムコールを設定し、実現した。

setpgmask() ファイルのPGファイルアクセス権の設定を行なうシステムコールである。引数で指定したファイルに対して、ファイルの所有者をこのシステムコールを実行したプロセスの属するPGにする。同時に、ファイルの所有者であるPGのモードと、その他に対するモードを引数で指定されたモードにする。

clearpgmask() PGファイルアクセス権の解除を行なうシステムコールである。

chpgmod() PGファイルアクセス権の設定されているファイルのモードの変更を行なうシステムコールである。

chpgown() PGファイルアクセス権の設定されているファイルの所有者の変更を行なうシステムコールである。

statpgf() PGファイルアクセス権の設定されているファイルの属性情報の取得を行なうシステムコールである。

setpgmask()システムコールは、PGに属するプロセスが実行した場合のみ有効である。statpgf()システムコールは、PGファイルアクセス権の設定されたファイルに対して実行した場合のみ有効である。それ以外のシステムコールは、PGファイルアクセス権の設定されたファイルに対してそのPGSIDと一致するPGSIDを持つPGに属するプロセスが実行した場合のみ有効である。

4.3 PGを用いたファイルアクセス制御の実現

PGを用いたファイルアクセス制御を実現するために、UNIXのファイルシステムにPGファイルアクセス権の制御を行う機能を追加した。

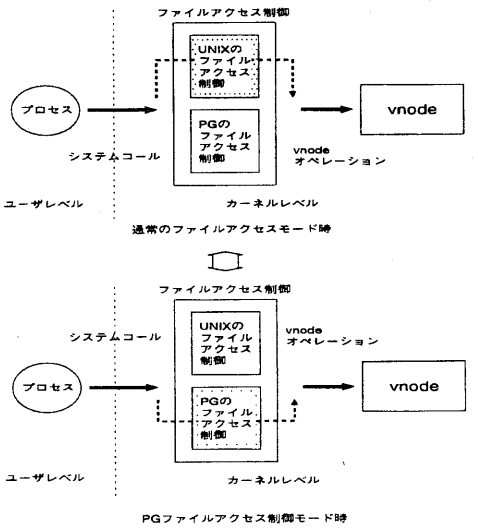


図7: PGファイルのアクセス制御のモデル図

図7のファイルアクセス制御部は、プロセスがファイルのアクセス制御を伴うシステムコールを発した時に、vnodeオペレーションを実行してファイルのアクセス制御を行なう部分である。

システムでは、図7のようにファイルアクセス制御部を二つのモジュールで構成している。一つは、UNIXファイルアクセス制御部であり、もう一つは、PGファイルアクセス制御部である。PGファイルアクセス権の設定されていないファイルに対しては、UNIXファイルアクセス制御部がアクセス権の検査を行う。また、PGファイルアクセス権の設定されていないファイルの所有者の変更もここで管理される。PGファイルアクセス権が設定されているファイルに対しては、PGファイルアクセス制御部がアクセス権の検査を行う。また、PGファイルアクセス権が設定されているファイルの所有者の変更もここで管理される。

PGファイルアクセス制御部では、ファイルに設定されたPGファイルアクセス権の情報を管理している。PGファイルアクセス権は、動的な設定であるので、カーネル内すなわちメモリ上にのみ存在させ、ディスク上にはこの情報を反映させないようにした。このため、PGファイルアクセス制御部を実現するためにvnodeに対する変更を行なう必要はない。すなわち、UNIXにおける既存のファイルシステムをそのまま使用できるという利点がある。

4.4 PGファイルアクセス制御の使用例

ここでは、PGファイルアクセスの使用例について説明する。

4.4.1 PG を用いたファイルアクセス制御の区別化

PGを用いたファイルアクセス制御方式を用いて、ファイルアクセスの区別化(図4(a)参照)を行うことができる。一つのアプリケーションプログラムをプロセスA、Bで構成している。図4(a)の左の状態では、プロセスA～DのUIDと、ファイルのUIDが一致するのでUIDに対するファイルのアクセス権が適用され、プロセスA～Dはどれもファイルに対するアクセスが可能である。以下の操作を行なうことによって、図4(a)の右の状態にすることができる。

1. createpgs() システムコールを用いてPGを作成する。
2. bindpgs() システムコールを用いて、プロセスA、BをPGに登録する。
3. setpgmask() システムコールを用いてファイルに対してPGファイルアクセス権を設定し、PGをファイルの所有者とする。

この状態で、プロセスA～Dが、再びファイルアクセスを行なうことを考える。ファイルには、PGファイルアクセス権が設定されているので、プロセスA、Bからのファイルアクセスは、PG識別子が一致した場合のアクセス権が使用され成功する。プロセスC、Dからのファイルアクセスは、PG識別子が一致しない場合のアクセス権が使用され失敗する。

4.4.2 PG を用いたファイルアクセス制御の同一化

PGを用いたファイルアクセス制御方式を用いて、ファイルアクセスの同一化(図4(b)参照)を行うことができる。プロセスA、Bは一つのアプリケーションプログラムを構成するものとする。図4(b)の左の状態では、ファイルのUIDとプロセスA、CのUIDが一致するので、ファイルのUIDに対するアクセス権が適用されプロセスAのファイルアクセスは成功する。しかし、ファイルのUIDとプロセスBのUIDが一致しないので、グループまたは、その他のアクセス権が適用され、プロセスBのファイルアクセスは失敗する。以下の操作を行なうことによって、図4(b)の右の状態にすることができる。

1. createpgs() システムコールを用いてPGを作成する。
2. bindpgs() システムコールを用いて、プロセスA、BをPGに登録する。
3. setpgmask() システムコールを用いてファイルに対してPGファイルアクセス権を設定し、ファイルの所有者をPGとする。

この状態で、プロセスA、Bが、再びファイルアクセスを行なうことを考える。ファイルには、PGファイルアクセス権が設定されているので、プロセスA、Bからのファイルアクセスは、PG識別子が一致した場合のアクセス権が使用され成功する。プロセスCからのファイルアクセスは、PG識別子が一致しない場合のアクセス権が使用され失敗する。

4.4.3 PG を用いたファイルアクセス制御の一時的設定

ファイルアクセス制御を一時的に設定(図(5)参照)することができる。プロセスのPG化を行なった後で、ファイルに対してsetpgmask() システムコールを実行することによって、アクセス権を一時的に設定することができる。また、clearpgmask() システムコールを実行することによってファイルアクセス制御を解除することができる。

4.4.4 PG を用いたファイルアクセス可能プロセスの動的変更

ファイルアクセス可能プロセスの動的変更(5(b)参照)を行なうことができる。プロセスCをbindpgs() システムコールを行なってPGに追加することによってファイルにアクセス可能なプロセスを追加することができる。また、unbindpgs() システムコールを行なうことによって、ファイルアクセス可能なプロセスを削除することができる。

5 おわりに

本論文では、既存のUNIXシステムにおけるプロセスのスケジューリングの問題、およびファイルのアクセス制御の問題を解決するために、プロセスグループの概念をシステムに導入した。プロセスグループに対応したスケジューラのモデルを提案し、実際のシステム上において実装を行った。プロセスグループに対応したファイルのアクセス制御方式を提案し、実際のシステム上において実装を行った。カーネルのソースコードに対して、三千行程度のソースコード付加することによってこれらの実現を行なうことができた。今後、プロセスの複数のプロセスグループへの登録や、プロセスをプロセスグループに登録する際の検査機構を実現する。また、プロセスグループを用いたアクセス制御をファイル以外の資源に対しても行なえるようにしたい。

参考文献

[1] Marshall Krik McKusick, Keith Bostic, Michael J. Karels, Jhon S. Quarterman, The Design and Implimentation of the 4.4BSD Operating System Addison-Wesley Publishing Company

[2] Maurice J. Bach, UNIX カーネルの設計, 坂本 文, 多田 好克, 村井 純 訳, 共立出版, 1991.

[3] Uresh Vahalla, UNIX INTERNELS THE NEW FRONTIERS, Prentice Hall, 1996.

[4] Rob Kolstad, Tony Sanders, Jeff Polk, Mike Karels, BSD/OS 2.0 and BSDI Internet Server Release Notes, Berkley Software Design, Inc. , 1995.

[5] 藤枝隆行, プロセスのグループ化によるスケジューリングとファイルのアクセス制御方式, 筑波大学大学院理工学研究科修士論文, 1997.