2007－OS－105 (5)
2007／4／5

*Tender*におけるOS動作の可視化のための情報収集と表示の方式

木下彰 † 河原太介 †† 田端利宏 †† 谷口秀夫 ††

† 岡山大学工学部
†† 岡山大学大学院自然科学研究科

ソフトウェアの規模は大規模化しており，ソフトウェアの構造やその動作状況を把握することは容易でない．特に，OSのように処理流れの複雑なソフトウェアでは，処理流れを把握するのは困難である．また，不具合の発生時，不具合箇所の特定が難しいという問題がある．これまでに，OS動作の可視化の研究は行われているが，OSの内部処理の動作の流れや，割込や例外の実行を可視化できるものは少ない．

そこで，本論文では，***Tender***オペレーティングシステムにおいて割込を考慮したOS動作の可視化機能を述べる．***Tender***では，OSレベルで可視化機能を実現できるプログラム構造を採用し，割込処理などの処理契機を意識して，OS内部の処理流れを収集できる．これにより，学習を目的としたOS動作の可視化や，プログラム開発支援を目的とした可視化のための情報を収集できる．OSの学習を目的とした可視化の例として，プロセススケジューリングの様子を可視化する表示部の実現方式と可視化例を示し，実現した方式により，OS動作の学習を支援できることを示す．

Mechanism of Information Gathering and Display for Visualization of Operating System Behavior in *Tender*

Akira Kinoshita † Taisuke Kawahara †† Toshihiro Tabata †† Hideo Taniguchi ††

† Faculty of Engineering, Okayama University
†† Graduate School of Natural Science and Technology, Okayama University

It is difficult to grasp the behavior situation of software, because a scale of software grows. In addition, it is hard work to understand the flow of complicated software such as OSs. Although the visualization of OS behavior has been developed, it is difficult to grasp the flow of internal processing of OS including interrupts and exceptions.

In this paper, we describe the mechanism of the visualization of OS behavior including interrupts in ***Tender***. In ***Tender***, the information of the flow of internal processing of OS can be gathered by adopting the program structure, and the mechanism of visualization can be achieved at OS level by considering the processing type of interrupt and so on. Employing this mechanism, the information of the visualization of OS behavior can be used for studying OSs and for supporting program developments. We describe the mechanism of display and the visualization of the process scheduling as an example of OS behavior to study.

1. はじめに

サービスの多様化や要求の高度化により，ソフトウェアの規模が大きくなっており，その構造を把握するのは難しい．また，ソフトウェアは，その動作状況を見ることができないため，様々な問題を抱えている．例えば，動作状況を把握できないため，処理の実際の流れを理解するのが難しく，不具合の発生時，不具合箇所の特定が難しい．特に，オペレーティングシステム(以降，OS)のように処理の流れが複雑なソフトウェアでは，これらは非常に大きな問題である．このような背景から，OSの動作を可視化する研究がある．

多くのOSでは，プロセッサやメモリといった資源の現在の利用状況(以降，状態情報)を可視化できる．しかし，状態情報の把握では，処理の流れを理解することは難しい．処理の流れを理解するには，少なくとも，プログラムの時系列の実行状態(以降，動作情報)を可視化できる必要がある．そこで，不具合箇所の発見，改修を支援するために，特定アドレスでの状態情報を取得する機能の研究[1)2)]がある．この機能では，

特定アドレスが実行・参照・更新されたことは把握できるが，プログラムモジュール間の参照順序関係は把握できない．また，OS の動作の学習を目的に，動作情報を可視化する研究もある[3]．しかし，この研究では，割込処理を考慮したプログラムモジュール間の呼出関係を把握できない．しかし，OS の動作を理解するには，割込処理の動作を理解することは重要である．

そこで，本論文では，***Tender***[4] における OS 動作の学習とプログラム開発支援を主目的としたOS の動作の可視化機構について述べる．具体的には，OS プログラム構造への要求条件を示し，割込を考慮した情報収集機構について述べる．また，収集した情報を基に，OS 動作の学習を目的とした可視化表示部の実現方式について述べる．さらに，プロセススケジューリングと OS 処理の切り替え契機の学習を目的とした可視化の例を示し，提案機構がOS 動作の理解を支援することを示す．

2. 可視化の基本方式

2.1 可視化の目的

OS 処理の可視化の主な目的としては，以下の2 つがある．

(1) OS 動作の学習支援
(2) OS プログラム開発支援

OS 動作の学習支援を主目的とした場合,OS に関する細かい知識を持たない者が対象者であると考えられる．この場合，処理の流れやシステム状態の情報をより分かりやすく提供することが求められる．一方，OS プログラム開発支援を主目的とした場合，ある程度 OS に対する知識のある者が対象者であると考えられる．また，可視化の利用形態は，動作試験や不具合発生時の動作確認などである．

2.2 求められる要件

OS 動作の学習では，状態情報の変化だけでなく，プログラムの実行に応じて，OS 内部の処理がどのように呼ばれているのかを示すことが有効である．また，OS 特有のプログラムの実行契機として，割込や例外がある．さらに，プロセスの切り替え要因には，様々なものがある．このため，OS の仕組みについて学習するには，これらの仕組みを収集し，わかりやすく表示できる必要がある．

また，開発支援では，不具合発生箇所や処理遅延箇所を特定し，その部分の動作を確認できるようにすることが有効である．このためには，処理の呼出関係を把握することが必要である．さらに，各処理に要する時間を求めることで不具合発生箇所の特定が容易になる．このため，プログラム開発を支援するためには，処理の呼出関係とともに，各処理に要する時間を求めて表示する必要がある．

2.3 基本構造

可視化機能としての処理は，以下の4 つの部分から構成される．

(1) 可視化情報の収集 (収集部)
(2) 可視化情報の受け渡し (転送部)
(3) 可視化情報を基にしたOS 動作の解析 (解析部)
(4) 解析データの可視化表示 (表示部)

収集部は，情報収集の対象となる処理を常に監視している必要があるため，OS 内部に実現する．解析部と表示部は，データ処理や画面表示の処理を行う．これらの処理部は，OS 内部に実現すると，OS 内部の負荷が増加する要因となるため，OS 外部に実現する．

以上のことから，OS 内部の収集部とOS 外部の解析部の間で可視化情報を受け渡すため，転送部を設ける．

2.4 OS プログラム構造への要求

OS 動作を理解するためには，OS の機能ごとにプログラムが分割され，かつ分割されたプログラムの呼出関係を把握できることが求められる．また，時系列に合わせて処理の呼出関係を得る必要がある．そこで，***Tender*** では，これらの要求を次の形で満足させた．

2.4.1 資源の分離

Tender では，OS が操作する対象を資源として分離し，独立化している．資源には，資源識別子と資源名を付与し，資源操作のインタフェースを統一している．各資源を操作するプログラム部品を資源ごとに分離し，共有プログラムを排除している．また，各資源の管理情報を資源ごとに分離している．従って，プログラム部品に対する呼出の情報を記録できれば，OS の機能とプログラム部品の関連を把握できる．

2.4.2 プログラムモジュール間の呼出把握の支援

Tender のプログラムは，基盤部・表プログラム構造部・入出力制御部・拡張部の4 つの部分で構成され，資源を管理するプログラムは，表プログラム構造部で管理される．

表プログラム構造部では，プログラムをプログラム部品と呼ぶ単位に分割し，独立化する．これらプログラム部品の呼出は，必ず資源インタフェース制御(RIC: Resource Interface Controller) と呼ぶモジュールを経由して行う．RIC は “操作対象の資源の種類” と “操作の種類” から構成されるプログラムポインタ表によってプログラム部品を管理している．このプログラムポインタ表を図1 に示す．“操作対象の資源の種類’ はプロセスや実メモリといった資源の種類であり，“

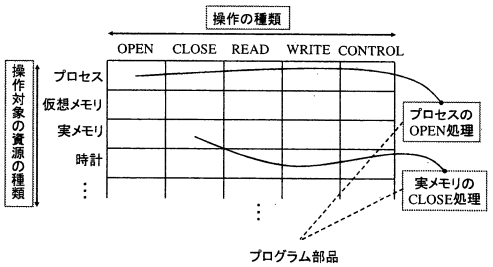

図 1　プログラムポインタ表

操作の種類" は OPEN, CLOSE, READ, WRITE, CONTROL のいずれかである．ここでは，このプログラム部品により行われる処理を資源処理と呼び，情報収集の対象とする．全ての資源処理の呼出は，RIC を経由して行われる．従って，RIC において可視化情報の収集が可能である．以降では，可視化の収集部と表示部の設計と実現方式について述べる．

3. 収集部

3.1 収集する情報

処理の呼出関係を把握するには，各処理の要求元と要求先を明確にする必要がある．このとき，時系列に沿った呼出関係を把握するには，各処理の行われた時刻を記録することが有効であり，各処理の内容と結果の情報を知る必要がある．また，割込などのOS 処理を認識するために，どのようなOS 処理を契機として処理が行われているのかを判断する必要がある．

以上のことから，OS 動作の情報を可視化するために，次の情報を資源処理の呼出ごとに記録する．

(1)　処理の要求元を示す情報
(2)　処理の要求先を示す情報
(3)　処理内容
(4)　処理結果
(5)　処理開始契機を示す情報
(6)　処理の行われた時刻

本機構では，表 1 に示す項目を記録することにより，上記の情報を収集する．dest_rid は，処理の要求先を示す情報であり，処理の対象となった資源の資源識別子を記録する．operate は，処理内容に関する情報であり，OPEN, CLOSE, READ, WRITE, CONTROL のいずれかを意味する情報を記録する．ret_val は，処理結果に関する情報であり，処理の戻り値を32 ビット符号無し整数として記録する．flowid は，処理開始契機を示す情報であり，処理に関連付けられた流れ識別子を記録する．流れ識別子の詳細については3 節で述べる．call_k_time は処理開始時の時刻であり，

表 1　記録する項目

項目名	内容
dest_rid	操作対象の資源識別子
operate	操作の種類
ret_val	戻り値
flowid	流れ識別子
call_k_time	呼出時の時刻
ret_k_time	復帰時の時刻

ret_t_time は処理終了時の時刻である．

これらの情報を記録するために，動作記録表と呼ぶデータ構造を用意する．動作記録表は，表1 の項目を 1 エントリとする配列形式のデータ構造である．動作記録表の 1 エントリの記録は，一回の資源処理の情報に相当する．また，以後動作記録表の各エントリ番号を，そのエントリに対応する資源処理の処理番号と呼ぶ．各資源処理は処理番号によって一意に識別できる．

ここで，表 1 には (1) に相当する項目はない．各処理の要求元を示す情報は，要求元の処理番号であり，この情報は解析部において記録済みのエントリを検索することで得られるため，項目として記録しない[5]．

処理の行われた時刻は，IA32 の RDTSC を用いて，CPU 起動時からの動作クロック数を記録する．

3.2 情報収集機構の構造

Tender では，資源処理の要求は，全て RIC を介して行う．その際，RIC は "操作対象の資源識別子または資源名" と "操作の種類" をパラメータとして受け取る．資源識別子と資源名は 1 対 1 に対応しており，これらはいずれも資源の種類を示す情報を含んでいる．資源処理呼出モジュールでは，RIC に渡されたパラメータを用いて要求された "資源の種類" と "操作の種類" の組み合わせに対応するプログラム部品を呼び出し，制御を移す．

そこで，この資源処理呼出モジュールが各プログラム部品の呼出前と呼出後に，必要な情報を記録するように資源処理呼出モジュールの機能を拡張することにより，OS 動作の可視化に向けた情報収集機構を実現する．このようにすることで，全ての資源処理に対して情報の収集を行うことができる．この方式による情報収集機構の概要を図2 に示す．ここで，プログラムポインタ表を参照し，要求されたプログラム部品を呼び出す処理を本処理，本処理の前後に行う記録処理をそれぞれ前処理，後処理と呼ぶ．前処理時に記録される項目は表 1 の dest_rid, operate, flowid, call_k_time である．これらは，動作記録表の先頭のエントリから順に書き込まれる．後処理時に記録される項目は，ret_val, ret_k_time である．これらは，当該処理の前処理時に記録が行われたエントリに書き込まれる．

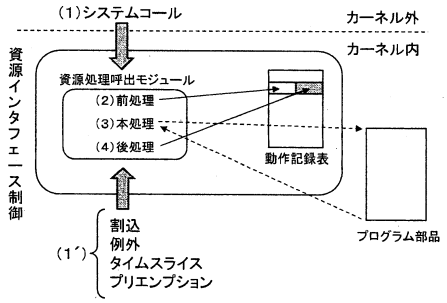

図 2 情報収集機構の概要

dest_rid, operate, flowid は，処理の要求元から前処理へパラメータとして渡される．ret_val は，本処理の戻り値が後処理へパラメータとして渡される．call_k_time, ret_k_time は，それぞれ前処理と後処理において，CPU 起動時からの動作クロック数をハードウェアクロックカウンタによって取得する．

3.3 割込を考慮した処理の流れの把握法

割込の発生により，OS はその時点で実行していた処理を中断し，割込処理を開始する．そして，割込処理の終了後，割込により中断された処理を再開する．割込による処理の流れを把握するためには，割込処理と割り込まれた処理をそれぞれ異なる契機によって開始された処理の流れとして区別し，それぞれの流れの中での呼出関係を把握することが求められる．そこで，割込による処理の流れを明確化するために，処理の開始契機を OS の機能に基づいて分類し，各処理と処理開始契機に関する情報を関連付けて記録する．

3.3.1 処理開始契機の分類

資源処理の開始契機の種類は，その要因の違いにより次のように分類できる．

(契機 1) ユーザプロセスによる資源処理要求
(以降，システムコール発行)
(契機 2) 割込
(契機 3) 例外
(契機 4) タイムスライス
(契機 5) プリエンプション

処理を開始契機によって区別し，同一の開始契機を持つ処理間での呼出関係を把握することで，割込を考慮した処理の流れを明確化できる．また，処理の開始契機を分類することで，処理の流れと OS の機能との関連を容易に把握できる．

3.3.2 流れ識別子を用いた処理と開始契機の関連付け

処理を開始契機と関連付けて記録するために，処理の開始契機を示す“流れ識別子(flowid)” を定義し，各処理に関連付けて記録する．この構成を図 3 に示す．

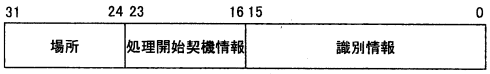

図 3 流れ識別子の構成

flowid は，分散環境における場所を示す場所情報のほかに，処理開始契機情報と識別情報を持つ．これらを表 2 に示す．処理開始契機情報は，(契機 1) から (契機 5) のいずれかを示す定数値である．また，識別情報は，同一種類の処理開始契機を識別する情報を持つ．これにより，例えば，割込処理中に発生した別の割込による処理の流れを把握するために，割り込まれた処理の流れと割り込んだ処理の流れにそれぞれ異なる flowid を与えて区別する．

3.3.3 流れ識別子の記録法

資源処理要求を行う際に RIC に対して flowid を渡すことにより，flowid の記録を行う．ここで，RIC に対して資源処理要求を行う処理を以下の2つに分ける．

(1) 資源処理開始モジュール
(2) 資源処理

処理開始契機と資源処理開始モジュールの対応を表 2 に示す．システムコール発行，タイムスライス，プリエンプションを契機とする処理は，それぞれシステムコールハンドラ，タイムスライス処理，プリエンプション処理によって開始される．割込を契機とする処理は，割込ベクタ番号に対応する割込ハンドラによって開始される．例外を契機とする処理は，例外ベクタ番号に対応する例外ハンドラによって開始される．

flowid の受け渡しの様子を図4 に示す．資源処理開始モジュールは，資源処理要求の際,flowid を生成し，RIC へ渡す．各資源処理が必要とする資源処理要求の際には，自らへ渡された flowid を RIC へ渡す．RIC による資源処理呼出の際には，自らへ渡されたflowid を資源処理へ渡す．

このように flowid の受け渡しを行うことにより，同じ処理の流れに属する資源処理は同一の flowid と関連付けて記録される．

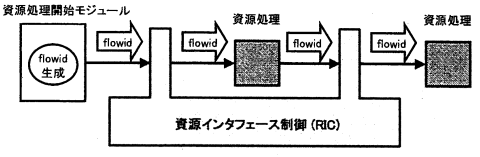

図 4 流れ識別子の受け渡しの様子

表 2　処理開始契機と識別情報および資源処理開始モジュール

処理開始契機情報	処理開始契機情報の値	識別情報	資源処理開始モジュール
システムコール発行	0x0a	処理を要求したプロセスのプロセス識別子の通番	システムコールハンドラ
割込	0xef	割込ベクタ番号	各割込ハンドラ
例外	0xee	例外ベクタ番号	各例外ハンドラ
タイムスライス	0xed	発生時のプロセスのプロセス識別子の通番	タイムスライス処理
プリエンプション	0xec	発生時のプロセスのプロセス識別子の通番	プリエンプション処理

4. 表示部

4.1 基本構造

2.1 節で述べたように，可視化は OS 動作の学習支援を目的の 1 つとしている．そこで，表示部では，割込などの処理契機を考慮した動作の流れを理解しやすい形で提供することを目標とする．割込の発生やプロセススケジューリングによるプロセス切り替えなどの様子を可視化することで，割込を考慮したプロセスの動きを学習できる．この可視化による学習は，OS に関する基礎的な知識を持つ者，例えば，大学で情報系の勉強をしている学部学生程度のレベルを対象とする．

収集部において収集した情報は，資源呼出を基本単位とした詳細な処理の情報である．この情報を表示した例を表 3 に示す．表 3 には表 1 に示す項目以外に，id，src_id，rsc_kind を示している．id は処理識別子で，処理が行われた順に0 から値をインクリメントする．src_id は呼出元の処理識別子を示し，rsc_kind は操作対象の資源の種類を示す．これらの情報はOS 内部処理を理解するためには有効である．しかし，可視化する内容は複雑になり，表示する情報量が多くなってしまうため，処理全体の流れは把握しにくい．そこで，割込を考慮した動作の可視化を行うにあたり，flowid の情報を基に各処理契機と流れ，処理時間の表示方法について述べる．これにより，各プロセスの処理が実行される契機とその長さ，およびその順序が視覚的に把握できる．また，割込や例外の処理が開始された時刻やその処理量も把握することができ，OS の動きを学習できる．

流れ識別子を基にした解析

flowid は処理開始契機情報を持つため，flowid の値から OS 処理の種類を判断できる．また，flowid の持つ情報から，flowid が同じ値である場合は同一処理であり，flowid が変化するごとに違う処理に処理が移ったと判断できる．

これにより，flowid が同じ値である間の資源呼出処理を 1 つの処理として，処理の流れを示す．このとき，処理開始時の時刻と flowid 変化時の時刻を取ることで各処理の処理時間を算出し，時系列に合わせて処理の流れを示す．つまり，時系列に合わせたflowid の移り変わりを処理の流れとして表示する．

4.2 実現方式

ここでは，可視化表示部を実現するためSVG(Scalable Vector Graphics)[6] を用いた．SVG は，XML 形式の文書 (以降，XML 文書) で記述されるベクタグラフィック言語である．SVG を用いる利点として，以下のものがあげられる．

(1) XML 文書で画像を表現できるため，エディタとブラウザがあれば開発可能
(2) ベクタ画像であるため，拡大，縮小が容易
(3) XML 形式なので，SVG 画像を文書として生成するプログラムの記述が容易

なお，ベクタ画像とは，画像を点の座標とそれを結ぶ線や面の方程式のパラメータ，および，塗りつぶしや特殊効果などの描画情報の集合として表現する方式である．また，アニメーションを用いたり，他の文書へのリンクを設定することもできるため，今後の可視化機能の拡張が容易になると期待できる．

4.3 表示方法

提案した表示を行うにあたり，収集した情報を解析して表示を行うアプリケーション(以降，可視化 AP)を用いる．可視化を行うにあたり，まず，***Tender*** 上で動作の記録を行い，動作記録した情報をバイナリファイルに書き出す．そして，可視化 AP 実行時に，動作記録したファイルを読み込んで解析を行い，SVG 画像を XML 文書として出力する．以降，可視化の表示項目，および表示方法について説明する．

表示項目

可視化表示において表示する項目を以下に示す．

(1) 処理の流れ
(2) flowid
(3) 処理時間

処理の流れは，flowid の変化に合わせた処理契機の移り変わりを示す．flowid は，各処理の flowid の値と，そのときの処理契機を表示する．処理時間は，処理開始時から終了までの時間軸を表示し，時間軸に合わせて処理の流れを示す．各項目の表示方法については，以下で詳しく述べる．

表 3　資源「仮想空間」の生成処理に対応する動作記録表の例

id	src_id	dest_rid	rsc_kind	operate	ret_val	flowid	call_k_time	ret_k_time
4686	-1	d0140	VMEM	OPEN	140	a0008	df894b764	df899c735
4687	4686	b068b	VREG	OPEN	68b	a0008	df894d4cf	df89593cb
4688	4687	2284e	PMEM	OPEN	284e	a0008	df8958de0	df8958f89
4689	4686	30137	VKM	OPEN	137	a0008	df895a2d2	df8971a2c
4690	4689	b068b	VREG	CONTROL	1000	a0008	df895ac1a	df895ada9
4691	4689	d0001	VMEM	CONTROL	0	a0008	df895af19	df89718e9
4692	4691	b068b	VREG	CONTROL	1000	a0008	df895b1fa	df895b38e
4693	-1	80000	EXEC	CONTROL	80009	ef0000	df8962618	df8968b2d
4694	4691	b068b	VREG	CONTROL	284e000	a0008	df897137f	df897151b

表示方法

まず，各処理の flowid の値，およびそのときの処理契機は図の左部，時間軸は図の下部に表示する．

次に，処理の流れの表示方法について述べる．割込処理を含めた各処理の種類，発生箇所はそれぞれflowidの値，値の変化箇所で判断できる．flowid が変化するごとに処理が別の処理に移ったと判断できるため，1つの flowid ごとに処理を表記する段を設け，処理ごとに分けて表示を行う．このとき，同flowid の処理は同一処理であるため，同じ段に表示する．また，各処理は長方形で表し，処理契機ごとに色を変化させる．

時間変化に合わせた動作の可視化を行うにあたり，以下の項目を考慮して処理の流れを表示する．

(1)　時間軸の表示法
　表示を行うにあたり，処理全体を見るためには時間軸1目盛りあたりの時間を大きく取り，詳細を見るためには時間を小さく取る必要がある．このことから，時間軸1目盛りあたりの時間を指定し，指定した時間で時間軸を表示する．

(2)　時間を考慮した処理の表示法
　処理に要する時間は処理により異なる．そこで，時間軸として指定した1目盛りあたりの時間を用いて各処理を示す長方形の横幅を算出し，時間軸に合わせて各処理を表示する．

(3)　処理数の多い動作の表示法
　動作記録時間が長い動作を可視化する場合や，時間軸1目盛りあたりの時間を小さく取る場合，可視化図は膨大になる．そこで，表示する範囲を指定し，その範囲のみを図として表示する．

5. OS 動作の可視化例

ここでは，実際のOS動作の例としてプロセススケジューリングを取り上げ，プロセス切り替え処理の流れと契機を可視化する例を示す．具体的には，実行速度[7]の異なる3つのプロセスを生成して並行に走行させたときの動作の記録を取り，可視化する．

この例で走行させた3つのプロセスのプロセス識別子 (pid) は，それぞれ 0xa0019，0xa001a，0xa001b である．また，3つのプロセスが走行する以前に走行していたプロセスの pid は，0xa0018 である．pid の持つ情報の内，0xa は資源「プロセス」であることを示し，下位 16 ビットの値は通番を示す．以降，各プロセスを単にプロセス a0018，プロセス a0019，プロセス a001a，プロセス a001b と呼ぶ．

ここで，***Tender*** におけるプロセスの実行速度調整機能について説明する．***Tender*** では，プロセッサの割り当て単位として，資源「演算」を導入している[3]．そして，資源「演算」を持つプロセスだけがスケジューリングの対象となる．スケジューリング方式としては，プログラムの実行速度を調整する方式と優先度による方式の2つの方式を共存させている．この例では，プロセス a0019，プロセスa001a，プロセスa001b に割り当てるプロセッサ処理時間の割合は，それぞれ全体の 10%，30%，50% とした．

この動作の表示例を図5 に示す．この図は，時間軸1目盛りあたりの時間を 10ms として，3つのプロセスが走行している様子を表示した図である．

この例では，まずプロセス a0018 のシステムコール発行処理により3つのプロセス，および演算が生成され，各プロセスに演算が割り当てられる．次に，プリエンプションにより 3 つのプロセスに動作が移る．それから，タイムスライスにより各プロセスが走行する．最後に，プロセスa0018 のシステムコール発行処理により 3 つのプロセス，および演算が削除される．このとき，flowid は各契機の処理発生時のプロセス識別子の情報を含んでいる．例えばflowid が 0xed0019 のタイムスライス処理が発生したときのプロセスは，プロセス a0019 である．すなわち，タイムスライスは，そのときの flowid の持っているプロセス識別子の情報が示すプロセスにおいて，タイムスライスを契機とした処理が行われたことを表す．また，各プロセスは以下に示す順序で走行していることがわかる．

```
a001a->a001b->a001a->a001b->a0019->a001a
    ->a001b->a0018->a001a->a001b->a001a
```

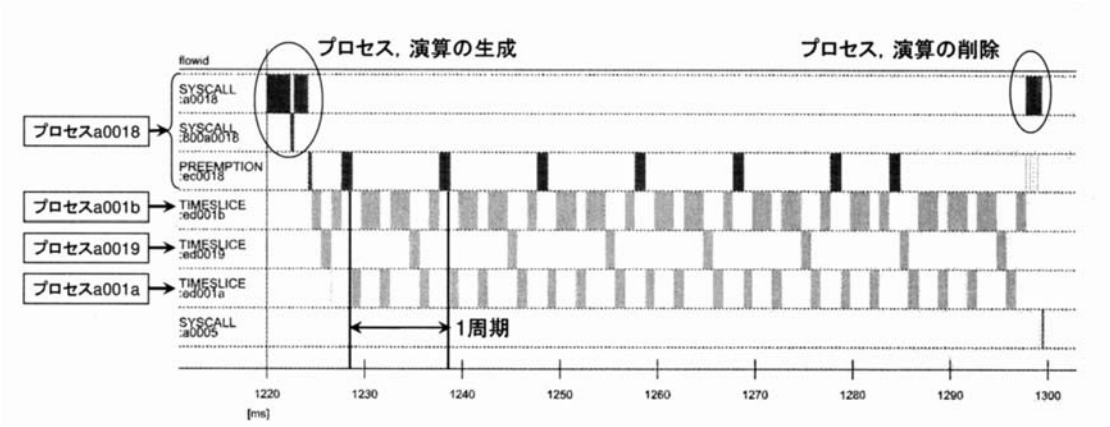

図 5 表示の事例 (プロセススケジューリング)

表 4 各プロセスの走行時間 (1 周期)

pid	a001a	a001b	a001a	a001b	a0019	a001a	a001b	a0018
処理時間 [μsec]	871	2003	1002	2006	1001	1003	1002	1135

```
->a001b->a0019->a001a->a001b->a0018
```

上記の周期性から，

```
a001a->a001b->a001a->a001b->a0019->a001a
    ->a001b->a0018
```

を 1 周期として走行している．このとき，プロセスの 1 周期中の走行時間を表 4 に示す．表 4 より，1 周期中の 3 つのプロセスの走行時間の比は，以下に示す通りである．

$$a0018 : a0019 : a001a : a001b \simeq 1 : 1 : 3 : 5$$

これから，各プロセスに割り当てたプロセッサ処理時間の比と，プロセス走行時間の比は一致することがわかる．また，プロセスa0018 によるプリエンプション処理は，実行速度調整機能で割り当てられない残りの 10% を利用して実行されていることがわかる．

また，プロセスの実行速度調整機能は，タイムブロックとタイムスロットを用いた方式で実現されている．タイムブロックとは，プロセッサの処理時間を一定時間 (1 秒) ごとに区切ったものであり，タイムスロットとは，タイムブロックをある一定の時間(1 ミリ秒) でさらに区切ったものである．この関係を図 6 に示す．実行速度の調整は，タイムブロック中のタイムスロットを指定された性能(%) の割合分だけ演算に割り当てる．例えば，性能が10% の場合は図 6 のようになる．

このとき，1 つのタイムスロットは1ms であることから，10ms で周期性があることがわかる．また，図5 と表 4 から，割り当てたプロセッサ性能に基づいて処理が行われていることがわかる．例えば，10%のプロセッサ性能を割り当てているプロセス a0019 の場合，

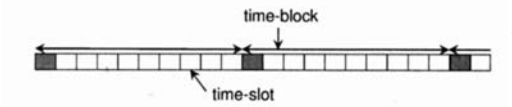

図 6 演算性能 10% の場合のタイムスロットの割当

1 周期中に 1 回タイムスロットが割り当てられている．

これらのことから，図5 に示す動作は，各プロセスに割り当てた性能に基づいた処理が行われていることがわかる．従って，時間変化に合わせたOS の処理の流れを可視化表示することができ，OS のプロセススケジューリングの学習を支援できるといえる．

また，割込が発生する動作の表示例を図 7，割込が発生してる箇所を拡大した図を図 8 に示す．この図は，プロセスを生成するプログラムを実行したときの様子を表示した図である．この例では，割込処理が発生し，それに伴いシステムコール発行処理が行われている．その後，プロセスa0009 により，新たにプロセスが生成されている．このとき， flowid が 0xef000e の割込処理は，プロセス生成前の数十ミリ秒の間に頻繁に発生している．実際に処理を行う場合，キーボードからコマンドを入力してプログラムを実行する必要がある．そして，この例において実行したプログラムはプロセスを生成するプログラムである．

これらのことから，発生した割込処理は，コマンド入力時のキーボード割込であることがわかる．従って，割込を考慮した処理の流れを示すことができ，割込の発生を含めた動作の学習を支援できるといえる．

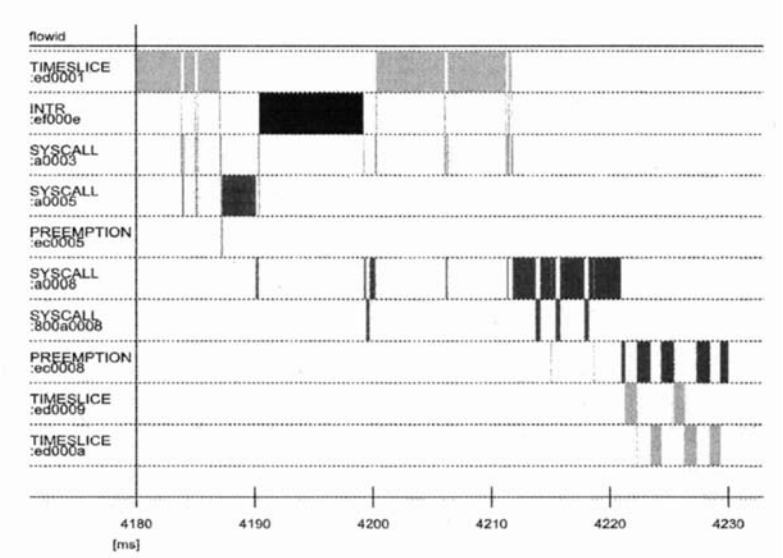

図 7 表示の事例 (割込)

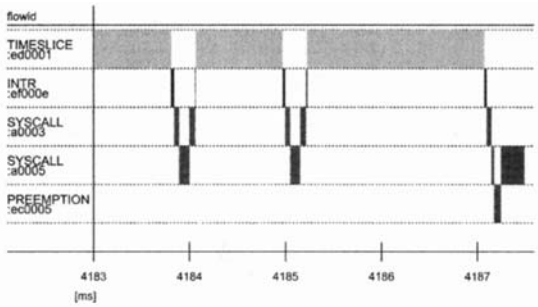

図 8 表示の事例 (割込) 拡大図

6. おわりに

OS 動作の可視化に向けた情報収集機構の方式について述べた．割込を考慮した処理の流れを明確化するために，処理開始契機を分類し，処理開始契機と処理を関連付けて記録する方法について述べた．この提案方式に基づき，割込を考慮した情報収集機構を ***Tender*** に実現した．そして，収集した情報のうち，流れ識別子の情報を基にして，各処理を処理契機により分類して解析，表示を行う方式を提案し，実現した．

また，実現した情報収集機構により実際に動作の記録を取り，割込やタイムスライスといったOS 処理を含めた動作の可視化表示を行った．

残された課題として，転送部，解析部を含めた可視化機能全体の構築，資源呼出を基本とした詳細な動作の可視化方法の検討がある．

参考文献

1) Bryan M. Cantrill, Michael W. Shapiro, Adam H. Leventhal : Dynamic Instrumentation of Production Systems, USENIX Annual Technical Conference, pp.15-28, 2004.
2) Vara Prasad, William Cohen, Frank Ch. Eigler, Martin Hunt, Jim Keniston, Brad Chen : Locating System Problems Using Dynamic Instrumentation, In Proceedings of the Linux Symposium, Vol.2, pp.49-64, 2005.
3) 西野洋介，早川栄一，高橋延匡：可視化によるOS 教育支援環境の実現と評価，情報処理学会研究報告，2001-OS-88, Vol.2001, No.78, pp.107-114, 2001.
4) 谷口秀夫，青木義則，後藤真孝，村上大介，田端利宏：資源の独立化機構による***Tender*** オペレーティングシステム，情報処理学会論文誌, Vol.41, No.12, pp.3363-3374, 2000.
5) 河原太介，谷口秀夫：***Tender*** における OS 動作の可視化に向けた情報収集機構の実現，情報処理学会研究報告, 2004-OS-97, Vol.2004, No.82, pp.41-48, 2004.
6) W3C : Scalable Vector Graphics (SVG), http://www.w3.org/Graphics/SVG/.
7) 田端利宏，谷口秀夫 :***Tender*** オペレーティングシステムの資源「演算」によるプログラム実行速度調整機能の実現と評価，情報処理学会論文誌, Vol.40, No.6, pp.2523-2533, 1999.