(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2009-259089
(P2009-259089A)
(43)公開日　平成21年11月5日(2009.11.5)

(51)Int.Cl.		F I		テーマコード（参考）
G06F 11/28	(2006.01)	G06F 11/28	310A	5B042

審査請求 未請求　請求項の数 6　ＯＬ　（全 20 頁）

(21)出願番号　特願2008-108954 (P2008-108954)
(22)出願日　平成20年4月18日 (2008.4.18)

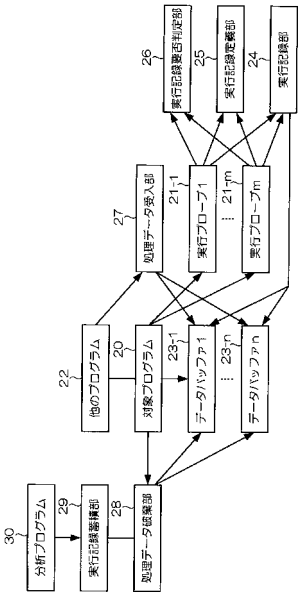
（特許庁注：以下のものは登録商標）
１．Ｌｉｎｕｘ

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(74)代理人　100106909
弁理士　棚井　澄雄
(74)代理人　100134544
弁理士　森　隆一郎
(74)代理人　100148873
弁理士　渡辺　浩史
(74)代理人　100150197
弁理士　松尾　直樹
(72)発明者　狩野　秀一
東京都港区芝五丁目７番１号　日本電気株式会社内

Ｆターム(参考)　5B042 HH25　HH30　LA02　LA05　LA09
LA24　MA05　MA08　MC12　MC28

(54)【発明の名称】プログラム実行経路追跡装置、プログラム実行経路追跡方法、及びプログラム

(57)【要約】
【課題】特定データの処理経路を選択的に記録／読み出し可能とする。
【解決手段】実行プローブ２１－１～２１－mは、対象プログラム２０内の特定の位置が実行されたときに、それを検出する。実行記録定義部２５は、ユーザによって設定される、実行記録として保持すべき情報の種類等を保持する。実行記録要否判定部２６は、実行プローブ２１－１～２１－mから得たプログラム内の位置と、処理データの内容及び、これまでの処理記録と、上記実行記録定義部２５の判定条件とにより、該当処理データの処理記録が、所望のものであるかどうかを検査する。実行記録部２４は、要否判定に従って、実行プローブ２１－１～２１－mからの要求に従って、データバッファ２３－１～２３－ｎの所定位置に実行記録を書き込む。
【選択図】図１



【特許請求の範囲】
【請求項１】
対象プログラムの実行経路を記録するプログラム実行経路追跡装置であって、
前記対象プログラムの実行に伴う処理追跡情報を記録するか否かを判定するための処理追跡情報記録条件を記憶する条件記憶手段と、
前記対象プログラムの実行に伴う入出力データと処理追跡情報とを検出する検出手段と、
前記条件記憶手段に記憶されている処理追跡情報記録条件に基づいて、前記検出手段により検出された処理追跡情報を記録すべきか否かを判定する実行記録要否判定手段と、
分割された複数の記録領域を有し、前記実行記録要否判定手段により記録すべきと判定された場合、前記対象プログラムの実行に伴う入出力データに対応付けて前記処理追跡情報を前記複数の記録領域のいずれかに保持する保持手段と
を備えることを特徴とするプログラム実行経路追跡装置。
【請求項２】
前記保持手段は、
前記入出力データに対応付けられる処理追跡情報を保持する第１の保持手段と、
前記入出力データと独立して処理追跡情報を保持する第２の保持手段とを備え、
前記対象プログラム実行中、前記入出力データに対応付けられる処理追跡情報を、前記第１の保持手段に保持し、前記入出力データ破棄時には、記録要の入出力データを前記第２の保持手段に移動させる記録制御手段を更に備えることを特徴とする請求項１記載のプログラム実行経路追跡装置。
【請求項３】
前記入出力データが分割される場合、分割された入出力データの各々に、前記処理追跡情報の複製を対応付けて、前記保持手段に保持する記録制御手段を更に備えることを特徴とする請求項１記載のプログラム実行経路追跡装置。
【請求項４】
前記入出力データを連結する場合、それぞれの入出力データを移動させることなく、それぞれの入出力データに対応する複数の記録領域を連結することで、前記入出力データの連結を実現する記録制御手段を更に備えることを特徴とする請求項１記載のプログラム実行経路追跡装置。
【請求項５】
対象プログラムの実行経路を記録するプログラム実行経路追跡方法であって、
前記対象プログラムの実行に伴う処理追跡情報を記録するか否かを判定するための処理追跡情報記録条件を記憶するステップと、
前記対象プログラムの実行に伴う入出力データと処理追跡情報とを検出するステップと、
前記記憶されている処理追跡情報記録条件に基づいて、前記検出手段により検出された処理追跡情報を記録すべきか否かを判定するステップと、
記録すべきと判定された場合、前記対象プログラムの実行に伴う入出力データに対応付けて前記処理追跡情報を、複数の記録領域のいずれかに保持するステップと
を含むことを特徴とするプログラム実行経路追跡方法。
【請求項６】
対象プログラムの実行経路を記録するプログラム実行経路追跡装置のコンピュータに、
前記対象プログラムの実行に伴う処理追跡情報を記録するか否かを判定するための処理追跡情報記録条件を記憶するステップと、
前記対象プログラムの実行に伴う入出力データと処理追跡情報とを検出するステップと、
前記記憶されている処理追跡情報記録条件に基づいて、前記検出手段により検出された処理追跡情報を記録すべきか否かを判定するステップと、
記録すべきと判定された場合、前記対象プログラムの実行に伴う入出力データに対応付

けて前記処理追跡情報を、複数の記録領域のいずれかに保持するステップと
を実行させることを特徴とするプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、特定の処理データに注目した場合の通信ソフトウエアの実行経路を追跡するプログラム実行経路追跡装置、プログラム実行経路追跡方法、及びプログラムに関する。
【背景技術】
【０００２】
従来、プログラムが意図通りに動作しない場合の原因の分析や、性能最適化のための情報収集の手段として、通信プログラムの実行経路を追跡する技術が提案されている（例えば、特許文献１、２参照）。
【０００３】
また、通信装置内でのソフトウエアの実行経路を追跡する、ソフトウエア実行追跡方法の１つに次のようなものがある。通常、ソフトウエアは、データ及びプログラムから構成されており、このうち、プログラムのある位置の処理が実行されると、その事象を記録するような命令（プローブ）を、プログラム内に埋め込む。以降、プログラムが実行されると、それを記録する処理を実行プローブ処理と呼ぶ。
【０００４】
プログラムの中の、実行を記録したい場所に上記プローブを埋め込み、プログラムを実行して実行記録を収集し、該記録より上記目的のための分析を行う。プローブにより記録する情報としては、
・ソースコード上の位置
・該当位置の実行時刻（時計から得た値や、ＣＰＵのクロックサイクルカウンタの値など）
・重要な変数の値
が典型的な例である。
【０００５】
プローブは、もっとも原始的な手段としては、ｐｒｉｎｔｆ（）ルーチンなどにより、記録をシステムコンソール等に表示したり、システムログに記録したりすることにより実現できる。但し、ｐｒｉｎｔｆは、文字列への変換が伴うなど、実行コストが高いため、高度な追跡方法では、記録用のバッファが用意されており、バッファに記録データを蓄積し、プログラム実行後に該データを読み出すようになっている。これにより、同様の手段を効率的に実現できる。
【０００６】
また、特に、プローブ自体の処理が複雑になる場合には、プローブの実現にも種類があり、例えば、以下のような方法が使われている。
（１）対象プログラム内の記録点に直接命令を埋め込んでこれを実行させる。
（２）実行プローブをサブルーチンとして用意し、対象プログラム内の記録点に該サブルーチンを呼び出すコードを埋め込む。
（３）ＣＰＵのデバッグ機構（ブレークポイント等）を利用して、対象プログラムを変更せずにサブルーチンへ制御を移す。
【０００７】
これらの方法による実行記録は、例えば、次のような形式になっている。
・実行点の識別子１：実行記録情報１（時刻、変数の値など）
・実行点の識別子２：実行記録情報２（時刻、変数の値など）
【０００８】
また、少数の実行プローブにより実行記録を効率的に集める方法としてスタックトレースを用いる方法がある。あるルーチンから他のルーチンがコールされると、引数、戻り番地等が、呼出しのネストの順にスタックに積まれる。実行プローブ内で、この情報を記録

すると、実行プローブを含まないルーチンの実行記録（コールフローや引数など）も取得することができる。

【０００９】

実行プローブのサブルーチンは、実行点毎に記録すべき情報が異なる等の場合には、別々のものの場合もあるし、どの実行点でも同じ情報を記録する等の場合には、単一のサブルーチンを共用することもある。

【００１０】

これらの実行追跡方法を、通信ソフトウエアに適用する場合を考えてみると、次のようになる。通信ソフトウエアは、通常、アプリケーションプログラムと伝送路との間で、所定のプロトコルに従って、対向装置との間で送受信データを交換する処理を行うプログラムとして実現されている。また、ＴＣＰ／ＩＰなどの現代の通信ソフトウエアは、図１４に示すように、概念的に複数のプロトコルの実装が層状に繋がっているように表現できる。すなわち、最下位層には、Ｅｔｈｅｒｎｅｔ(登録商標) Ｄｒｉｖｅｒ１、Ｆｉｒｅｗｉｒｅ Ｄｒｉｖｅｒ２、ＰＰＰ Ｄｒｉｖｅｒ３、Ｌｏｏｐｂａｃｋ Ｄｒｉｖｅｒ４、その上層に、順次、Ｎｅｔｗｏｒｋ Ｉｎｔｅｒｆａｃｅ５、ＡＲＰ６、ＩＰｖ４（７）、ＩＰｖ６（８）、ＩＣＭＰｖ４（９）、ＵＤＰ１０、ＴＣＰ１１、ＩＣＭＰｖ６（１２）、Ｓｏｃｋｅｔ ＡＰＩ１３、Ａｐｐｌｉｃａｔｉｏｎ１４、１４、…というように構成されている。

【００１１】

また、より具体的な例として、ＢＳＤ系ＯＳのＴＣＰ／ＩＰスタックのうち、ＵＤＰ受信処理部分の構成を図１５に示す。図１５において、破線の間が該当部分である。すなわち、通信データは、伝送路とアプリケーションとの間の、プロトコルを実装したプログラム（ＩＰｖ４、ＵＤＰ、Ｓｏｋｅｔ ＡＰＩ）の中を通ってくることになる。

【００１２】

ここで、通信処理の分析については、特定のプロトコル実装などの振る舞いに着目したい場合もあるが、特定の通信データに着目したい場合もある。例えば、特定の受信パケットに含まれるデータが、どのモジュールに、どの時刻に処理されたかを追跡することで、該当データの伝送遅延の発生源を計測することができる。

【特許文献１】特開２００２－０７８２００号公報

【特許文献２】特開２００７－３２８５９７号公報

【発明の開示】

【発明が解決しようとする課題】

【００１３】

しかしながら、上述した従来技術による、実行プローブ毎に実行記録を残す方法では、次のような問題があった。

（１）着目したもの以外のデータを処理する場合であっても、該当プローブを通れば実行記録が発生する。したがって、着目したデータに関する実行記録を切り出すためには、どのデータの処理を行っているか逐一記録しなければならず、その記録のためのプログラムの作成コストが高くなる。また、例えば、処理対象のデータが暗号化されていたりすると、そもそも着目したデータなのかどうか、事後に識別できないこともある。

【００１４】

ここで、着目したいデータの条件が、例えば、次のような内容だったとする。

・トランスポートプロトコルの番号：６（ＴＣＰ）
・トランスポートプロトコルの宛先ポート番号：５０１３
・宛先ＩＰアドレス：１９２．１６８．１．１５
・ネットワーク層処理要件
　－ＥＳＰにより暗号化されていること
　－アウターヘッダの宛先ＩＰアドレス：１３３．２０１．５．１５

【００１５】

これらの条件を満たすパケットが運ぶデータのみ、実行経路の追跡を行うとする。しか

しながら、例えば、ＩＰ受信処理において、ある受信パケットが上記要件を満たすものかどうかを判定することはできない。これは、該当パケットがＥＳＰにより暗号化されていなければならず、かつ、その暗号化されたデータの条件が指定されているためである。

【００１６】

従って、この場合、ＩＰ受信処理では、上記要件の一部、具体的には、

・ＥＳＰにより暗号化されていること

・アウターヘッダの宛先ＩＰアドレス：１３３．２０１．５．１５

であることしか判別できない。この条件に合うパケットを処理するときは、実行プローブで常に実行記録を残すことが必要になる。

【００１７】

（２）着目したデータに対する処理なのかどうか、ある実行プローブの時点では、識別が困難だとすると、とりあえず実行記録を残すことになる。実行記録は、システムログなどに記録されるか、実行記録用のバッファ等に保存される。システムログに記録する場合、処理対象データが多くなると、システムログの出力量が増えて、その表示／記録等の処理量が多くなり、他の処理の性能を低下させるという問題がある。

【００１８】

一方、バッファに記録する場合、高速なメモリ等に記録するので、記録コストは低く実現できる場合もあるが、データが多量であるために、バッファに入りきらない場合が多くなる。この場合、バッファからデータを読み出して他の記憶装置に保存するなどの作業を間欠的に行う必要が発生し、その処理量が無視できなくなる。これらにより、記録処理の負荷により、本来実行を記録すべき処理の性能が低下し、実行記録を取らない場合とは異なる動作をする場合が無視できなくなり、その場合、初期の目的を果たせないことになる。

【００１９】

（３）通常、プロトコル処理は、イベント駆動、かつマルチスレッドになっている。すなわち、パケットの受信、タイマのタイムアウト等を契機として処理が始まり、別の契機により開始された処理は、論理的には独立して実行される。単一のデータの処理であっても、例えば、パケット受信契機で行われるプロトコルの処理の後、該当データがバッファに格納され、タイマにより駆動される別の処理が引き継ぐような場合がある。このような処理では、イベント毎に異なるスレッドが実行されるため、スタックもイベント毎に独立になる。この結果、実行プローブからスタックを見たとしても、該当イベントに関する実行記録しか得られず、該当データについての一貫した処理記録を取得できない場合があるという問題がある。

【００２０】

例えば、図１６に示すようなプログラムがあるとする。図１６において、ｒｏｕｔｉｎｅ１～４があり、各々、ｄａｔａ　ｄを処理する。ｒｏｕｔｉｎｅ１は、ｒｏｕｔｉｎｅ２を呼び出し、ｒｏｕｔｉｎｅ３は、ｒｏｕｔｉｎｅ４を呼び出す。ｒｏｕｔｉｎｅ２は、データの処理を中断し、ある条件が満たされると、スケジューラがｒｕｔｉｎｅ３を呼び出して同一データの処理を続けるようになっている。

【００２１】

この処理のシーケンスを図１７に示す。図１７には、時間の経過とともに、ルーチンの呼出しによりスタックが伸縮する様子を示している。このような場合、ｐｒｏｂｅをｒｏｕｔｉｎｅ４に入れても、次のようなスタックトレースしか得られない。

（スタックの先頭）
ｒｏｕｔｉｎｅ４
ｒｏｕｔｉｎｅ３
ｓｃｈｅｄｕｌｅｒ
（スタックの末尾）

【００２２】
実際には、データは、ｓｃｈｅｄｕｌｅｒの前に、ｒｏｕｔｉｎｅ２、ｒｏｕｔｉｎｅ１でも処理されているので、データの処理経路を完全に追跡することはできない。このように、特定データのみに着目してその処理経路等を記録するには、従来の実行プローブを用いる方式では十分な効果が得られず、通信ソフトウエアの開発及び性能最適化を行う上で大きな問題が存在する。
【００２３】
本発明は、このような事情を考慮してなされたものであり、その目的は、特定データの処理経路を選択的に記録／読み出すことができるプログラム実行経路追跡装置、プログラム実行経路追跡方法、及びプログラムを提供することにある。
【課題を解決するための手段】
【００２４】
本発明は、上述した課題を解決することを目的とし、対象プログラムの実行経路を記録するプログラム実行経路追跡装置であって、前記対象プログラムの実行に伴う処理追跡情報を記録するか否かを判定するための処理追跡情報記録条件を記憶する条件記憶手段と、前記対象プログラムの実行に伴う入出力データと処理追跡情報とを検出する検出手段と、前記条件記憶手段に記憶されている処理追跡情報記録条件に基づいて、前記検出手段により検出された処理追跡情報を記録すべきか否かを判定する実行記録要否判定手段と、分割された複数の記録領域を有し、前記実行記録要否判定手段により記録すべきと判定された場合、前記対象プログラムの実行に伴う入出力データに対応付けて前記処理追跡情報を前記複数の記録領域のいずれかに保持する保持手段とを備えることを特徴とするプログラム実行経路追跡装置である。
【００２５】
本発明は、上記の発明において、前記保持手段は、前記入出力データに対応付けられる処理追跡情報を保持する第１の保持手段と、前記入出力データと独立して処理追跡情報を保持する第２の保持手段とを備え、前記対象プログラム実行中、前記入出力データに対応付けられる処理追跡情報を、前記第１の保持手段に保持し、前記入出力データ破棄時には、記録要の入出力データを前記第２の保持手段に移動させる記録制御手段を更に備えることを特徴とする。
【００２６】
本発明は、上記の発明において、前記入出力データが分割される場合、分割された入出力データの各々に、前記処理追跡情報の複製を対応付けて、前記保持手段に保持する記録制御手段を更に備えることを特徴とする。
【００２７】
本発明は、上記の発明において、前記入出力データを連結する場合、それぞれの入出力データを移動させることなく、それぞれの入出力データに対応する複数の記録領域を連結することで、前記入出力データの連結を実現する記録制御手段を更に備えることを特徴とする。
【００２８】
また、上述した課題を解決するために、本発明は、対象プログラムの実行経路を記録するプログラム実行経路追跡方法であって、前記対象プログラムの実行に伴う処理追跡情報を記録するか否かを判定するための処理追跡情報記録条件を記憶するステップと、前記対象プログラムの実行に伴う入出力データと処理追跡情報とを検出するステップと、前記記憶されている処理追跡情報記録条件に基づいて、前記検出手段により検出された処理追跡情報を記録すべきか否かを判定するステップと、記録すべきと判定された場合、前記対象プログラムの実行に伴う入出力データに対応付けて前記処理追跡情報を、複数の記録領域のいずれかに保持するステップとを含むことを特徴とするプログラム実行経路追跡方法である。
【００２９】
また、上述した課題を解決するために、本発明は、対象プログラムの実行経路を記録す

るプログラム実行経路追跡装置のコンピュータに、前記対象プログラムの実行に伴う処理追跡情報を記録するか否かを判定するための処理追跡情報記録条件を記憶するステップと、前記対象プログラムの実行に伴う入出力データと処理追跡情報とを検出するステップと、前記記憶されている処理追跡情報記録条件に基づいて、前記検出手段により検出された処理追跡情報を記録すべきか否かを判定するステップと、記録すべきと判定された場合、前記対象プログラムの実行に伴う入出力データに対応付けて前記処理追跡情報を、複数の記録領域のいずれかに保持するステップとを実行させることを特徴とするプログラムである。

【発明の効果】

【００３０】

この発明によれば、対象プログラムの実行に伴う処理追跡情報を記録するか否かを判定するための処理追跡情報記録条件を記憶しておき、対象プログラムの実行に伴う入出力データと処理追跡情報とを検出し、記憶されている処理追跡情報記録条件に基づいて、検出された処理追跡情報を記録すべきか否かを判定し、記録すべきと判定された場合、対象プログラムの実行に伴う入出力データに対応付けて処理追跡情報を、複数の記録領域のいずれかに保持する。したがって、特定データの処理経路を選択的に記録／読み出すことができるという利点が得られる。

【００３１】

また、本発明によれば、対象プログラム実行中、前記入出力データに対応付けられる処理追跡情報を、前記第１の保持手段に保持し、前記入出力データ破棄時には、記録要の入出力データを前記第２の保持手段に移動させる。したがって、必要な実行記録のみ保存することにより、所望の処理データに対する実行記録だけを残すことができる。

【００３２】

また、本発明によれば、入出力データが分割される場合、分割された入出力データの各々に、処理追跡情報の複製を対応付けて保持する。したがって、分割後の処理データを、別々のものとして実行記録を保持し続けることができる。

【００３３】

また、この発明によれば、入出力データを連結する場合、それぞれの入出力データを移動させることなく、それぞれの入出力データに対応する複数の記録領域を連結することで、入出力データの連結を実現する。したがって、効率低下が遥かに大きいデータの移動を行わないので、効率的に入出力データを連結することができる。

【発明を実施するための最良の形態】

【００３４】

以下、本発明の一実施形態を、図面を参照して説明する。

【００３５】

Ａ．第１実施形態

図１は、本発明の第１実施形態による、システムの構成を示すブロック図である。図１において、本第１実施形態のシステムは、実行経路を記録する対象となる対象プログラム２０と、実行プローブ２１－１～２１－ｍと、他のプログラム２２と、データを保持するデータバッファ２３－１～２３－ｎと、実行記録部２４と、実行記録定義部２５と、実行記録要否判定部２６と、処理データ受入部２７と、処理データ破棄部２８と、実行記録蓄積部２９と、分析プログラム３０とからなる。

【００３６】

ここで、その他のプログラム２２とは、アプリケーションプログラムや、ＯＳのデバイスドライバなどが典型的な例である。対象プログラム２０は、処理データを他のプログラム２２から受け入れて所定の処理を行い、破棄または他のプログラム２２へ受け渡す。

【００３７】

実行プローブ２１－１～２１－ｍは、対象プログラム２０内の特定の位置が実行されたときに、それを検出し、所定の処理を行う機能を有する。特に、所定の処理を行うにあたり、各々の実行プローブ２１－１～２１－ｍは、プローブが実行を検知したプログラム上

の位置、プログラムが処理しているデータが格納されているメモリの位置、及び、実行プローブ２１－１～２１－ｍが検出したプログラムの実行点における、該当プログラム内から参照できる変数の値を識別できるものとする。これらの情報は、例えば、実行プローブ２１－１～２１－ｍが関数として実現されていれば、その引数として受け渡されるのが一般的である。なお、以下では、実行プローブ２１－１～２１－ｍを総称する際、実行プローブ２１とする。

【００３８】

データバッファ２３－１～２３－ｎは、対象プログラム２０が処理する処理データを保持する。ここで、処理データは、説明のために有限個の場合のみを扱うが、実際には、時間の経過により処理されるデータは入れ替わるので、有限とは限らない。但し、データバッファ２３－１～２３－ｎは、有限なので、同時に、所定の最大個数以上に使われることはない。また、データの個数が重要でない場合には、１つのデータのみについて説明する。

【００３９】

処理データは、１つ以上のデータバッファに格納される。ここで、データバッファ２３－１～２３－ｎは、固定長であり、データ本体のほかに、データの管理領域も備える。データの管理領域は、少なくとも、後続データを含むバッファの番地（後続データがない場合には、それを識別する情報)、該当データバッファのデータの種類、該当データバッファ内のデータの開始位置、該当データバッファ内のデータの量、該当データバッファ内の付加データの開始位置、及び該当データバッファ内の付加データの量を含む。なお、以下では、データバッファ２３－１～２３－ｎを総称する際、データバッファ２３とする。

【００４０】

実行記録部２４は、実行プローブ２１－１～２１－ｍからの要求に従って、データバッファ２３－１～２３－ｎの所定位置に実行記録を書き込む機能を有する。実行記録定義部２５は、ユーザによって設定される、実行記録として保持すべき情報の種類等を保持する機能を有する。実行記録要否判定部２６は、実行プローブ２１－１～２１－ｍから得たプログラム内の位置と、処理データの内容及び、これまでの処理記録と、所定の判定条件とにより、該当処理データの処理記録が、所望のものであるかどうかを検査する機能を有する。

【００４１】

ここで、判定条件は、例えば、トランスポートプロトコルの番号：６（ＴＣＰ）、トランスポートプロトコルの宛先ポート番号：５０１３、宛先ＩＰアドレス：１９２．１６８．１．１５、ネットワーク層処理要件（ＥＳＰにより暗号化されていること、アウターヘッダの宛先ＩＰアドレス：１３３．２０１．５．１５）という内容である。処理データや、変数の値を参照して、上記要件をすべて識別できれば、実行記録の要否を判定できる。判定条件は、ユーザによって定義され、実行記録要否判定部２６内に保持される。

【００４２】

処理データ受入部２７、及び処理データ破棄部２８は、各々、外部のプログラムからの処理データの受け取りと、外部のプログラムへ処理データを渡したあとのデータの破棄を実行する。具体的には、処理データ受入部２７は、空いているデータバッファを用意して、所定のインタフェースを介し、データが該当データバッファ２３に入るように調整する。処理データ破棄部２８は、所定のインタフェースを介し、データバッファ２３内のデータが外部のプログラムへ渡るようにし、さらにその後、該当データバッファ２３として使われていたメモリを未使用な状態に戻す。ここで、処理データ破棄部２８では、データバッファ２３のデータを破棄する際に、該当データバッファ２３に処理記録が書き込んであり、かつ、その記録が必要と判定されていれば、該当記録を実行記録バッファへ書き写す機能を有する。

【００４３】

実行記録蓄積部２９は、実行記録を処理データとは分離された状態で保持する機能と、蓄積した実行記録を外部のプログラムに読み出させる機能を有する。分析プログラム３０

は、実行記録蓄積部２９から実行記録データを読み出して該当データを加工分析する機能を有する。分析プログラム３０は、上記他のプログラムが兼ねても良い。

【００４４】

次に、上述した第１実施形態の動作について説明する。

まず、実行プローブ以外で処理データがどのように扱われるか説明する。

（１）バッファの用意

パケット交換を行う通信装置でパケットを受信する処理を例に説明する。

パケットが到着すると、ハードウエアから割り込みがかかり、割り込みハンドラ、デバイスドライバが順に呼ばれる。デバイスドライバでは、パケットを処理データとしてメモリに読み込む。具体的には、処理データ受入部２７で、格納用のデータバッファ２３－ｉ（ｉ＝１～ｎ）を用意し、該データバッファ２３－ｉに処理データを読み込む。これは、例えば、ＢＳＤ系オペレーティングシステムでは、ｍ＿ｄｅｖｇｅｔ（）がこの処理を行う。

【００４５】

データバッファ２３は、通常、複数個連結できるようになっており、データが１つのバッファに入りきらなければ、複数のバッファに分けて格納される。ここで、データバッファ２３は、固定長であるので、バッファのどの部分にデータが配置されているかを示すために、管理領域にデータ開始位置、データ量を保持する。一例として、ＢＳＤ系のオペレーティングシステムでは、ｍｂｕｆ構造体というデータバッファがこれに相当する。

【００４６】

図２及び図３は、上記ｍｂｕｆの表現を示す概念図である。単一のｍｂｕｆには、１００バイトまでのデータを格納でき、複数のデータバッファを連結することが可能である。この構造は、プロトコルヘッダの付け外しを効率的に行うのに好適である。データ量が少ないときは、図２に示すように格納し、多量のデータを扱うときは、図３に示すように、外部バッファ（ｍｂｕｆ　ｃｌｕｓｔｅｒと呼ぶ）２３０を用いる。同様の構造のバッファは、他のＯＳでも用いられており、ｌｉｎｕｘにおいては、ｓｋ＿ｂｕｆｆ構造体、Ｓｏｌａｒｉｓ等のＳＴＲＥＡＭＳを備えるＯＳでは、ｍｓｇｂ構造体がこれに相当する。

【００４７】

（２）データの処理によるバッファの組み替え

データバッファ２３に処理データが用意できると、プログラムは、所定の手順で該当データを処理する。処理手順は、本発明とは無関係のためここでは説明しないが、一般にプロトコル処理では、次のような特徴を有する処理が行われる。

【００４８】

（ａ）あるプロトコルのヘッダが単一のバッファに収まるようにする。処理データは、複数のデータバッファ２３に分かれて格納されていることがあり、その場合、データは、連続したものとして扱えない（例えば、型キャストにより構造体にマップするなど）。これを解決するために、ある領域が単一のバッファに収まるようにデータの移し替えとバッファの組み替えとを必要に応じて行う。ＢＳＤ系のオペレーティングシステムでは、この操作はｍ＿ｐｕｌｌｕｐ（）関数により行われる。

【００４９】

（ｂ）データバッファ２３の先頭領域に空きを作る。あるいは、先頭領域からデータを取り去る。通常、プロトコルデータは、先頭から見て下位層ヘッダから上位層ヘッダへ、プロトコルヘッダが繋がった形式になっている。受信時には、このヘッダを順に処理して取り除き、送信時には、ペイロードの先頭にプロトコルヘッダを組み立てる、という処理をすることになる。このため、処理データの先頭部分にデータを付けたり、取り除いたりする操作が典型的である。ＢＳＤ系のオペレーティングシステムでは、この操作を各々Ｍ＿ＰＲＥＰＥＮＤ（）マクロ、及びｍ＿ａｄｊ（）関数により行われる。

【００５０】

（３）他のプログラムへの受け渡しによるデータの破棄

処理データをプログラムが処理し終わると、一般に、次のいずれかの処理が行われる。

（ａ）他のプログラムに渡す。受信したパケットは、最終的にアプリケーションプログラムにより処理される。また、送信すべきパケットは、最終的には物理的な伝送路に繋がったハードウエアに渡すために、デバイスドライバに受け渡される。

【００５１】

これらの他のプログラム２２は、処理データの管理方法や、メモリのアドレス空間が異なる等のために、実行追跡対象プログラムのデータバッファ２３をそのまま使い回すことが一般にはできないようになっている。このため、データを受け渡すインタフェースが予め定義されており、そのインタフェースを経由してバッファの中身のデータを渡す。例えば、ＢＳＤ系のオペレーティングシステムでは、アプリケーション（ユーザープロセス）にデータを渡すにはｃｏｐｙｏｕｔ（）関数が使われる。その後、バッファ内のデータは不要なので破棄し、バッファは未使用状態として管理される。

【００５２】

（ｂ）データを破棄する。資源が不足している場合、あるいは、データが異常であって処理を続けられない場合には、他のプログラムに渡さずにデータを破棄することがある。この場合も、データバッファ２３を未使用状態に戻す。

【００５３】

このように、いずれの場合も、バッファを未使用状態に戻す処理が行われる。ＢＳＤ系のオペレーティングシステムでは、ｍ＿ｆｒｅｅｍ（）関数がこの処理を行う。

【００５４】

［実行プローブの動作］

次に、実行プローブ２１の動作を説明する。前述したように、実行プローブ２１は、
（１）対象プログラム２０内の記録点に直接命令を埋め込んでこれを実行させる、
（２）実行プローブ２１をサブルーチンとして用意し、対象プログラム内の記録点に該サブルーチンを呼び出すコードを埋め込む、
（３）ＣＰＵのデバッグ機構（ブレークポイント等）を利用して、対象プログラム２０を変更せずにサブルーチンへ制御を移す、
などの方法で、対象プログラム２０の実行中に実行される。

【００５５】

図４及び図５は、実行プローブの動作の説明するためのフローチャートである。実行プローブ２１では、まず、記録要否判定部２６にて本プローブ内での記録の要否を検査する（ステップＳａ１）。より具体的には、実行プローブ２１では、実行記録要否判定部２６に、プログラム位置、処理データの格納位置、及び処理記録の格納位置を渡し、実行記録の要否を調べる。実行記録要否判定部２６からは、記録要、記録不要、あるいは不明のいずれかが、判定結果として戻される。ここで、記録要／不要の場合には、その結果をデータバッファ２３の付加データ内に書き込む。一方、記録要または不明の場合には、実行記録をデータバッファ２３に書き込む処理を行う。

【００５６】

次に、上記判定結果に従って記録不要であるか否かを判定する（ステップＳａ２）。そして、記録不要以外の場合には（ステップＳａ２のＮＯ）、記録情報の収集を行う（ステップＳｂ３）。より具体的には、実行記録として格納すべき情報を集める。該情報は、実行記録定義部２５に保持されているので、それを参照して情報を収集する。実行プローブ２１では、予め、追跡対象のプログラム２０から、上述した情報、すなわち、プローブが実行を検知したプログラム上の位置、プログラムが処理しているデータが格納されているメモリの位置、及び、実行プローブが検出したプログラムの実行点における、該当プログラム内から参照できる変数の値を取得できるようになっている。また、実行記録定義部２５に格納された情報の収集手段も予め実行プローブ２１に用意されている。

【００５７】

非常に単純な実行記録データの例としては、例えば、図６に示すようなものが典型的である。図６において、タイプ（ｔｙｐｅ）は、この実行記録の種類、データ長（ｌｅｎｇｔｈ）は、この実行記録の長さ（バイト数）、シーケンス番号（ｓｅｑｕｅｎｃｅ　ｎｕ

ｍｂｅｒ）は、実行記録のシーケンス番号、プログラムカウンタ（ｐｒｏｇｒａｍ　ｃｏｕｎｔｅｒ）は、実行プローブ通過時のプログラムカウンタの値、クロックサイクルカウンタ（ｃｌｏｃｋ　ｃｙｃｌｅ　ｃｏｕｎｔｅｒ）は、実行プローブ通過時のクロックサイクルカウンタの値である。

【００５８】

上記形式例に従った、具体的なデータの例（１６進ダンプ）を示すと、「０１　１０　００　０５　ＦＥ　ＦＦ　３０　５４　００　００　００　００　０１　０４　Ｅ１　４２」となる。これを項目毎に分解すると、ｔｙｐｅ「０１」、ｌｅｎｇｔｈ「１０」、ｓｅｑｕｅｎｃｅ　ｎｕｍｂｅｒ「００　０５」、ｐｒｏｇｒａｍ　ｃｏｕｎｔｅｒ「ＦＥ　ＦＦ　３０　５４」、ｃｌｏｃｋ　ｃｙｃｌｅ　ｃｏｕｎｔｅｒ「００　００　００　００　０１　０４　Ｅ１　４２」となる。

【００５９】

シーケンス番号（ｓｅｑｕｅｎｃｅ　ｎｕｍｂｅｒ）により、処理データバッファ内の実行記録を記録順に並べることができる。また、プログラムカウンタ（ｐｒｏｇｒａｍ　ｃｏｕｎｔｅｒ）の値から、この記録がプログラム内のどのコードの実行時のものかが識別できる。また、クロックサイクルカウンタ（ｃｌｏｃｋ　ｃｙｃｌｅ　ｃｏｕｎｔｅｒ）の値から、この記録をとった時点をＣＰＵの動作クロックの精度で記録できる。これらを用いて、処理データに蓄積される実行記録を順に並べ、あるデータが通過したプログラム内の実行点間の処理量を算出できる。

【００６０】

次に、データバッファ２３上に格納領域を確保し（ステップＳｂ４）、データを書き込んで管理情報を更新する（ステップＳａ５）。より具体的には、まず、データバッファ２３内に十分な空き領域があるかどうか、すなわち、収集した情報をデータバッファ２３に格納できるか否かを調べる。つまり、処理データを格納するデータバッファ２３を調べ、収集した実行記録を格納するだけの空き領域があるかどうかを検査する。

【００６１】

最も簡単な方法は、データバッファ２３のリストを先頭から調べ、データの開始位置と長さから、処理データの前後の領域の長さを調べ、さらに、付加データの開始位置と長さから、空き領域の長さを調べる、というものである。以下、図５を参照して詳細に説明する。まず、記録データ長と先頭バッファを取得し（ステップＳｂ１）、バッファ長と保持データより、空き領域の大きさを計算する（ステップＳｂ２）。

【００６２】

例えば、図７に示すように、処理データを格納しているデータバッファ２３－１があるとする。このとき、先頭のデータバッファ２３－１の空き領域の長さは４０ｂｙｔｅｓ、２番目のデータバッファ２３－２の空き領域は無し、３番目のデータバッファ２３－３の空き領域の長さは２０ｂｙｔｅｓとなる。但し、この方法では、実行記録のバッファ内での位置関係としての並び順と、記録した時間的順序とが一致しないことがある。そのような順序を保存する必要がある場合には、例えば、実行記録にシーケンス番号を付けるようにし、最後に記録した実行記録のシーケンス番号と、それが記録されているデータバッファ２３の位置とを先頭の付加データに入れておく。そして、新しい実行記録を格納する場合には、上記最終実行記録があるデータバッファ２３から、空き領域の検査を始める。

【００６３】

次に、記録データ長が空き領域長以下であるか否かを判定し（ステップＳｂ３）、空き領域があれば、上述したステップＳａ５で、その領域に実行記録情報を書き込み、管理領域の付加データ関連情報を更新する。一方、空き領域がなければ、次のデータバッファ２３があるか否かを判定し（ステップＳｂ４）、次のデータバッファ２３があれば（ステップＳｂ４のＹＥＳ）、新しいデータバッファ２３を用意し（ステップＳｂ５）、上述したステップＳａ５で、そのデータバッファ２３に実行記録情報を書き込む。一方、次のデータバッファ２３が無い場合には（ステップＳｂ４のＮＯ）、新しいデータバッファ２３を用意し（ステップＳｂ６）、上述したステップＳａ５で、そこに実行記録を格納する。

【００６４】
新しいデータバッファ２３の用意は、処理データ受入部２７で行うのと同じ手順で行う。まず、データバッファ２３内に実行記録を書き込み、当該データバッファ２３の管理情報を更新する。次に、該データバッファ２３を、処理データが格納されたデータバッファの末尾に連結する。例えば、図８に示すように、先頭のデータバッファ２３－１には４０ｂｙｔｅの空き領域、２番目のデータバッファ２３－２には空き領域がなく、３番目のデータバッファ２３－３には２０ｂｙｔｅの空き領域がある状況において、バッファリストに４８ｂｙｔｅｓの実行記録を格納したい場合、どのデータバッファも空き領域が十分にないため、図９に示すように、別バッファとして４番目のデータバッファ２３－４を３番目のデータバッファ２３－３に連結して実行記録を格納する。

【００６５】
［データバッファ解放時の動作］
次に、データバッファの解放時の動作について説明する。
（１）処理データの一部を破棄する場合
特定のプロトコルヘッダだけを削除する場合など、一部のデータバッファ２３の中の処理データのみを破棄する場合について説明する。破棄する処理データを含むデータバッファ２３は、不要になるため、処理データを格納するバッファのリストから切り離すことになる。このとき、切り離すデータバッファ２３中に実行記録があれば、当該処理データが記録されたデータバッファ２３、または、処理データに連結されたデータバッファ２３にデータを移す。このとき、実行記録を書き込む場合と同様の手順で、データバッファ２３の空き領域を調べ、空きがあれば、そこに実行記録を書き移す。一方、空きがなければ、処理データを消去した、該当データバッファ２３が付加データのみを含むようにして、処理データバッファに連結する。

【００６６】
（２）処理データの全部を破棄する場合
他のプログラム２２に処理データを渡して該当データの処理を終える場合や、エラーのために、これ以上処理を続けられない場合など、処理データを全部破棄する場合について説明する。この場合、データバッファ２３内の付加データの記録要否を調べ、記録要となっている場合のみ、実行記録蓄積部２９にデータバッファ２３内の付加データをコピーする。その後、データバッファ２３をすべて未使用として処理する。

【００６７】
このようにすることで、プログラムがある処理データを受け入れてから削除するまで、そのプログラム内の実行プローブ２１が置かれた実行点の実行記録を処理データに保持することができる。

【００６８】
また、データの処理が進んで、実行記録の要否が判明した時点で、それを処理データとともに保持することにより、処理データ毎に実行記録の要否を識別することができる。

【００６９】
さらに、処理データ削除の時点で、必要な実行記録のみ実行記録蓄積部２９に保存することにより、所望の処理データに対する実行記録だけを残すことができる。

【００７０】
Ｂ．第２実施形態
次に、本発明の第２実施形態について説明する。
上述した第１実施形態では、実行記録データは、常に、実行記録蓄積部２９に保存され、実行記録を利用するプログラムは、実行記録蓄積部２９から実行記録データを別途取出す必要がある。この手順には、実行記録をまとめて取出せるなど利点もあるが、次のような課題もある。

【００７１】
（１）ある処理データに関する実行記録が完成した時点で、すぐに実行記録を取出すことが難しい。

（２）処理データと実行記録の対応付けが必要な場合には、実行記録を読み出すプログラムがその対応づけを行う必要がある。

【００７２】

これらの課題を解決するには、処理データの送受信を行うプログラム自身にて、実行記録の読み取りを、処理データの読み書きと同時に行う方式が好適である。例えば、Ｂｅｒｋｅｌｅｙ　Ｓｏｃｋｅｔ　ＡＰＩが備えるｒｅｃｖｍｓｇシステムコールでは、ｍｓｇｈｄｒ構造体のｃｏｎｔｒｏｌｌｄａｔａフィールドにて付加データの受け渡しができるようになっている。これを用いて、実行記録をユーザプロセスへ受け渡すことにより、処理データと同時にその実行記録も読み出すことができ、実行記録バッファからのデータの読み出し処理を省くことができる。

【００７３】

また、書き込み時には、ＡＰＩ引数にて実行記録を書き込むためのデータバッファ２３のポインタを渡すようにする。プログラム内では、送信データの付加データとして該当データバッファ２３へのポインタを保持し、データ送出時に実行記録を該当データバッファ２３へ書き出すようにすれば、同様の機能を実現できる。

【００７４】

Ｃ．第３実施形態

次に、本発明の第３実施形態について説明する。

上述した第１、第２実施形態では、処理データは、プロトコルヘッダに相当する一部が削られたり、付け足されたりすることはあっても、基本的に外部のプログラムから受け入れた一塊を、他の外部のプログラムへ受け渡すように処理が行われる、という条件で説明した。パケット交換などの通信システムでは、通信データは、前述したように処理されるが、ストリーム型の通信サービス、例えば、ＴＣＰのようなプロトコルの処理では、効率的にデータの伝送ができるように、処理データの分割や、あるいは、複数の処理データの連結などの操作が行われる。

【００７５】

特に、複数の処理データが連結されるような場合、各々の実行記録が混ざってしまわないように取り扱う必要がある。本第３実施形態では、ＴＣＰのように、処理データの連結、分割操作がある場合の実行記録の取り扱いについて説明する。

【００７６】

［処理データの分割］

処理データを分割する場合には、各々の分割データが、以降、異なる実行経路で処理される可能性がある。そのような場合に、支障無く実行を記録できる必要がある。これは、分割した各処理データに実行記録を各々持たせることで解決できるが、記録不要の場合には、この限りではないため、処理の場合分けが発生する。

【００７７】

記録不要としてマークされている場合には、その記録は不要なので、実行記録を削除する。それ以外の場合には、分割後の各々の処理データに実行記録を複製して添付する。これにより、分割後の処理データは、別々のものとして実行記録を保持し続けることができる。

【００７８】

［処理データの連結］

図１０は、本第３実施形態による、処理データの連結動作を示す概念図である。処理データを連結する場合には、別々の経路で処理されていたデータを、単一のものとして連結し、一連のものとして処理を続けることになる。多くの実装では、データの詰め直しによるメモリ使用効率の向上よりも、データの移動による効率低下が遥かに大きいため、処理データを連結しても、データバッファ２３－１、２３－２の中のデータの移動は行われず、データバッファ２３－１、２３－２の連結リストの組み替えのみが行われる。このため、処理データの連結により、実行記録の移動も必要ないのが通常である。

【００７９】

ここで、実行記録について注意すべきなのは下記の事項である。
（１）連結前の実行記録が混ざらないこと
（２）連結後の実行記録を効率的に保持できること
（３）実行記録の要／不要が判定できること
【００８０】
上記（１）については、実行記録に識別子を設け、由来が異なる処理データには、図１１に示すように、異なる識別子（ＩＤ＝１００、１０１）を設定するようにすることで、各々が混ざらないようにできる。
【００８１】
ここで、個々の実行記録データに識別子を記録すると、効率的でない場合もあるため、図１２に示すように、実行記録を独立したデータバッファ２３－１、２３－２に格納している場合には、データバッファ２３－１、２３－２毎に実行記録の識別子（ＩＤ＝１００、１０１）を管理しても良い。
【００８２】
また、どの実行記録が、処理データのどの部位に対するものかを識別したい場合もある。この場合には、実行記録に対象データの開始オフセットと長さとを保持させることで、処理データ連結後も個々の実行記録がそのどの部位の処理記録であるのかを識別し続けることができる。
【００８３】
次に、上記（２）については、図１３に示すように、連結前の実行記録と、連結後の実行記録とが区別できるようにし、連結後の実行記録を１つだけ保持することで、複数の実行記録への追記を避けることができる。
【００８４】
次に、上記（３）については、要不要が異なる実行記録が付いた処理データを連結するとき、無駄な情報を記録せず、必要な情報が失われないように取り扱う必要がある。連結前に不要と判定されている実行記録は削除して差し支えない。これに対して、連結前に記録要と判定されている実行記録がある場合には、以降の実行記録も必須である。よって、他に要否不明の実行記録があっても、記録要として扱って良い。
【００８５】
記録要否が不明の実行記録しかない場合には、そのまま不明として扱い、連結後も記録要否判定部で判定を続ける必要がある。記録要否判定部にて、実行記録が判定要否に使われる場合には、処理データの連結により、実行記録が複数の経路を含むことを考慮する必要がある。すなわち、連結前の実行記録と、連結後の実行記録とが繋がっているものとして扱い、すべての実行経路について要否を判定し、１つでも記録要と判定された経路があれば、該処理データの実行記録は記録要として扱う。
【産業上の利用可能性】
【００８６】
本発明によれば、通信装置等が搭載するプログラムの開発時に容易に不具合を発見し、装置の開発費用、期間を削減できる。このため、低価格な通信装置を短期間で提供できる。また、同プログラムの動作状況を観察して、性能に問題がある個所の発見が容易になるため、より高性能な装置を低価格かつ短期間で提供できる。
【図面の簡単な説明】
【００８７】
【図１】本発明の第１実施形態による、システムの構成を示すブロック図である。
【図２】本第１実施形態による、通信装置で用いられるデータバッファの典型例を示す概念図である。
【図３】本第１実施形態による、通信装置で用いられるデータバッファの典型例を示す概念図である。
【図４】本第１実施形態による、実行プローブの動作の説明するためのフローチャートである。

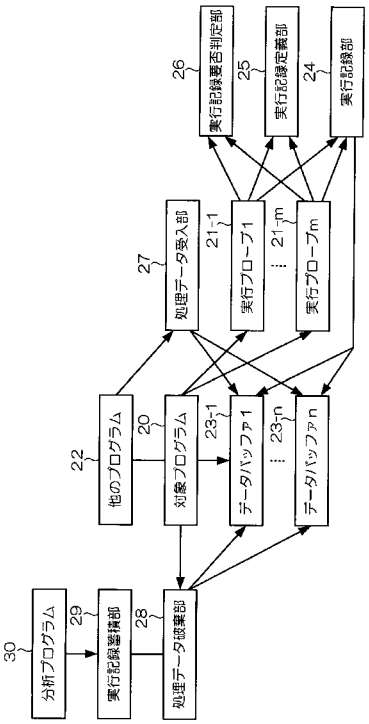
【図５】本第１実施形態による、実行プローブの記録用空き領域の算出動作の説明するためのフローチャートである。
【図６】単純な実行記録データの例を示す概念図である。
【図７】本第１実施形態による、記録領域の確保動作を示す概念図である。
【図８】本第１実施形態による、記録用空き領域の算出方法を示す概念図である。
【図９】本第１実施形態による、記録用空き領域の算出方法を示す概念図である。
【図１０】本第３実施形態による、処理データの連結動作を示す概念図である。
【図１１】本第３実施形態による、処理データ連結時の付加データの操作例を示す概念図である。
【図１２】本第３実施形態による、処理データ連結時の付加データの操作例を示す概念図である。
【図１３】本第３実施形態による、処理データ連結後の付加データの記録方式を示す概念図である。
【図１４】一般的な通信装置のソフトウエアの構成例を示すブロック図である。
【図１５】ＢＳＤ系ＯＳのＴＣＰ／ＩＰスタックのうち、ＵＤＰ受信処理部分の構成示すブロック図である。
【図１６】プログラムの一例を示す概念図である。
【図１７】通信ソフトウエアにおけるスタックの変化を説明するためのシーケンスである。

【符号の説明】

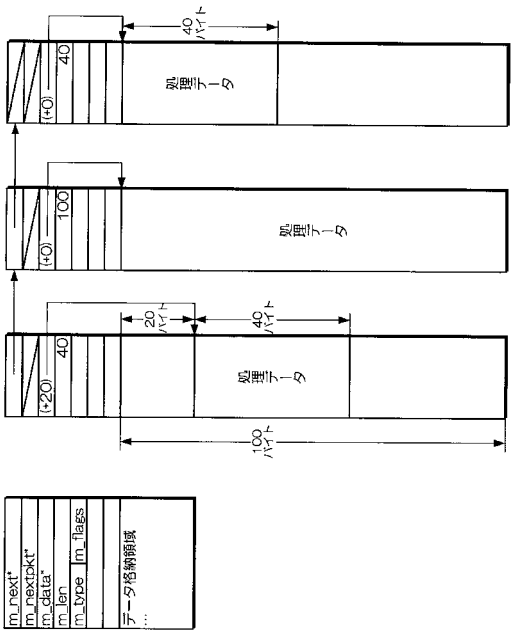
【００８８】

２０　対象プログラム
２１－１～２１－ｍ　実行プローブ
２２　他のプログラム
２３－１～２３－ｎ　データバッファ
２４　実行記録部
２５　実行記録定義部
２６　実行記録要否判定部
２７　処理データ受入部
２８　処理データ破棄部
２９　実行記録蓄積部
３０　分析プログラム

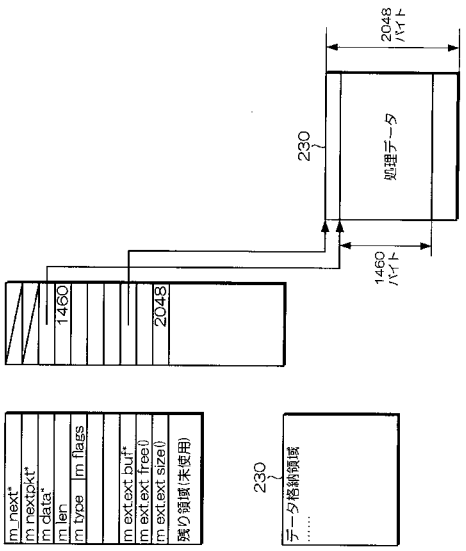
【図１】


26
実行記録要否判定部
25
実行記録定義部
24
実行記録部
27
処理データ受入部
21-1
実行プローブ1
21-m
実行プローブm
22
他のプログラム
20
対象プログラム
23-1
データバッファ1
23-n
データバッファn
30
分析プログラム
29
実行記録蓄積部
28
処理データ破棄部

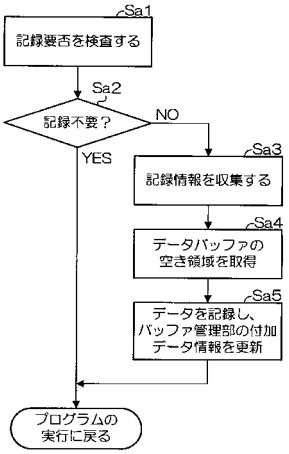
【図２】


40バイト
40
(+0)
処理データ
100
処理データ
20バイト
(+20)
処理データ
100バイト
m_next*
m_nextpkt*
m_data*
m_len
m_type
m_flags
データ格納領域

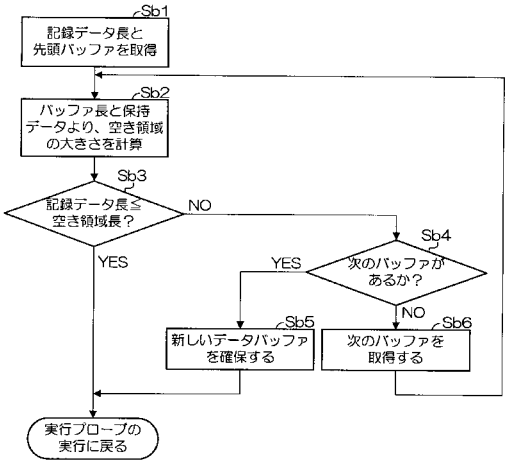
【図３】


2048バイト
230
処理データ
1460バイト
1460
2048
m_next*
m_nextpkt*
m_data*
m_len
m_type
m_flags
m_ext.ext_buf*
m_ext.ext_free()
m_ext.ext_size()
残り領域(未使用)
230
データ格納領域

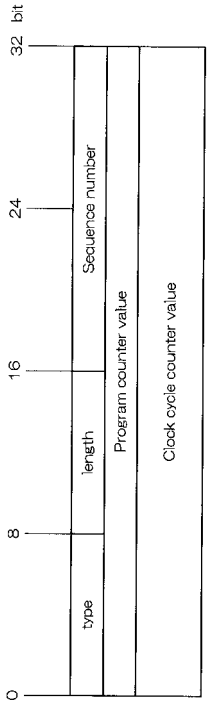
【図４】


Sa1
記録要否を検査する
Sa2
記録不要？
NO
YES
Sa3
記録情報を収集する
Sa4
データバッファの
空き領域を取得
Sa5
データを記録し、
バッファ管理部の付加
データ情報を更新
プログラムの
実行に戻る

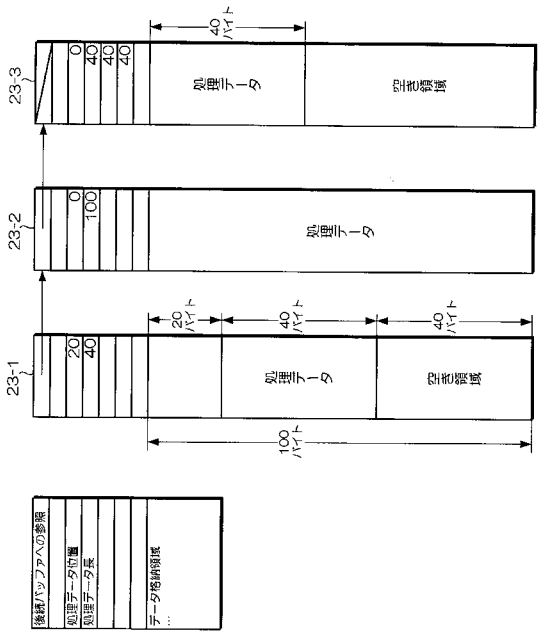
【図５】


Sb1
記録データ長と
先頭バッファを取得
Sb2
バッファ長と保持
データより、空き領域
の大きさを計算
Sb3
記録データ長≦
空き領域長？
NO
YES
Sb4
次のバッファが
あるか？
YES
NO
Sb5
新しいデータバッファ
を確保する
Sb6
次のバッファを
取得する
実行プローブの
実行に戻る

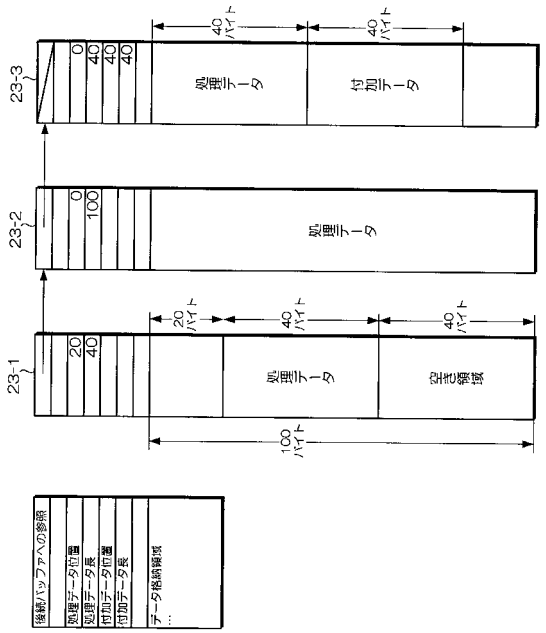
【図６】


32 bit
24
16
8
0
type
length
Sequence number
Program counter value
Clock cycle counter value

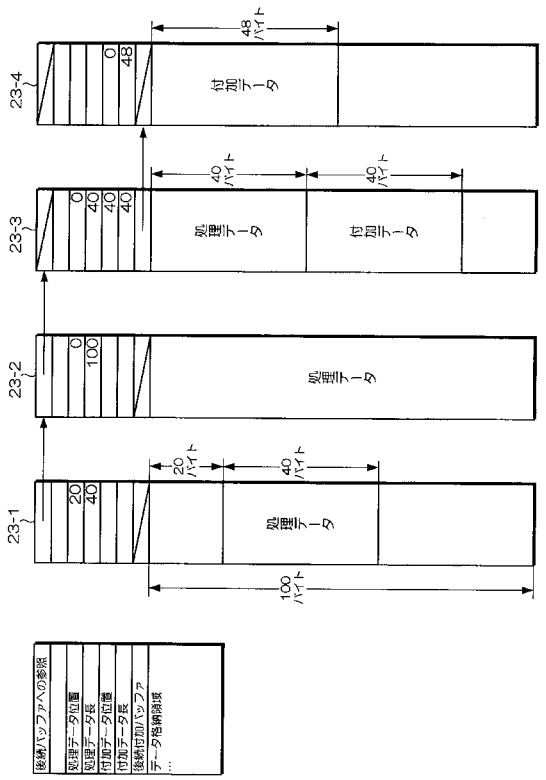
【図７】


23-3
23-2
23-1
0
40
40
40
0
100
20
40
40バイト
20バイト
100バイト
処理データ
空き領域
後続バッファへの参照
処理データ位置
処理データ長
データ格納領域

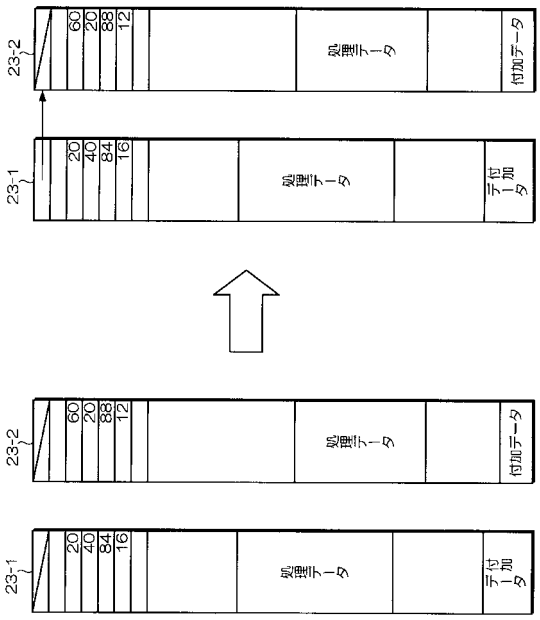
【図８】


23-3
23-2
23-1
0
40
40
40
0
100
20
40
40バイト
20バイト
100バイト
処理データ
付加データ
空き領域
後続バッファへの参照
処理データ位置
処理データ長
付加データ位置
付加データ長
データ格納領域

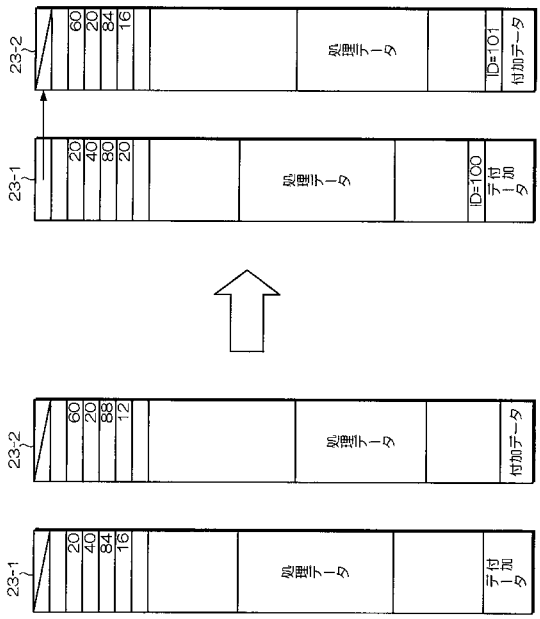
【図９】



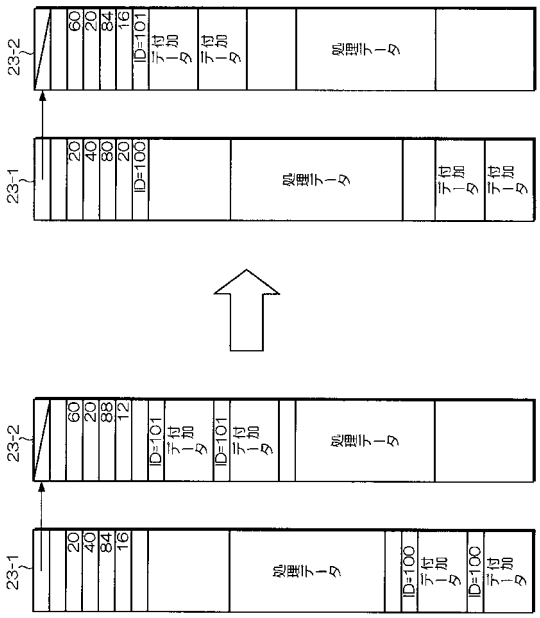
【図１０】



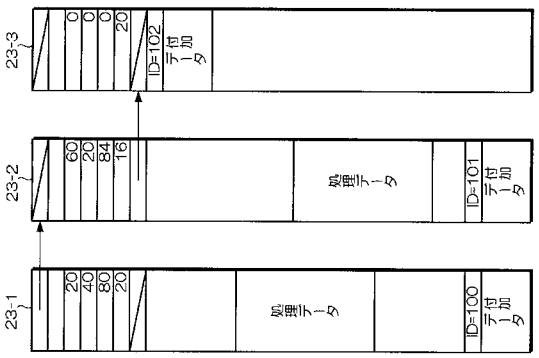
【図１１】



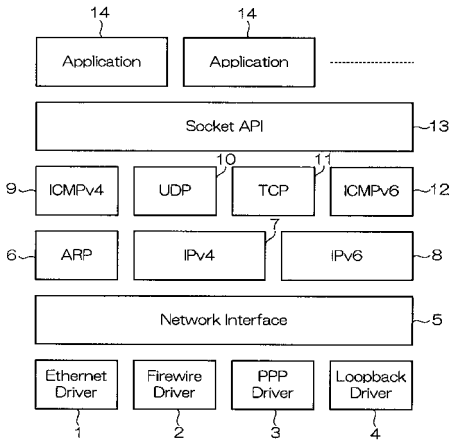
【図１２】



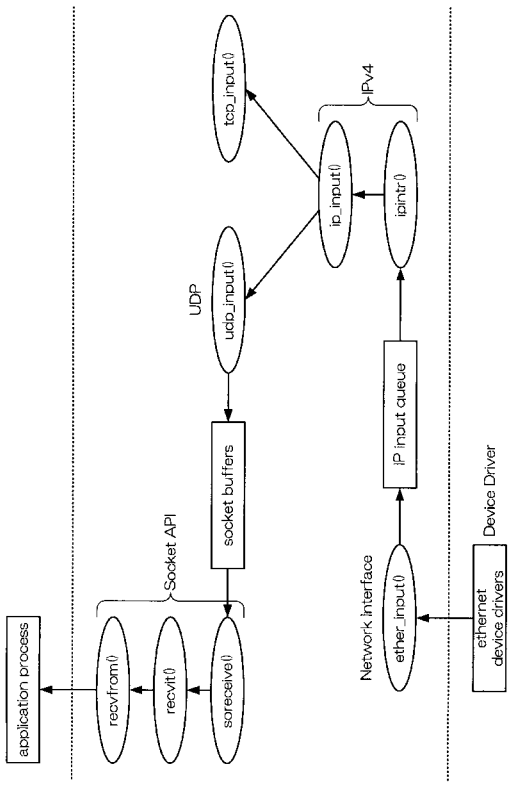
【図１３】



【図１４】



【図１５】



【図１６】

```
routine1(data d)
{
        routine2(d);
}

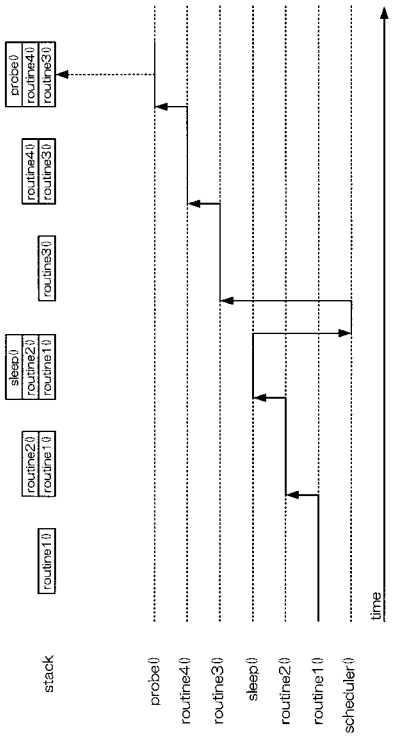
routine2(data d)
{
        /* suspend the task for data d */
        sleep();
}

scheduler()
{
        ...
        if (some condition) {
                /* resume the task for data d */
                routine3(d);
        }
        ...
}

routine3(data d)
{
        routine4(d);
}

routine4(data d)
{
        probe(routine4, d);
}
```

【図１７】


stack
routine1()
routine2()
routine1()
sleep()
routine2()
routine1()
routine3()
routine4()
routine3()
probe()
routine4()
routine3()
probe()
routine4()
routine3()
routine2()
routine1()
sleep()
routine3()
routine4()
scheduler()
time