

(19) 日本国特許庁 (JP)

(12) 特 許 公 報 (B2)

(11) 特許番号

特許第6119535号
(P6119535)

(45) 発行日 平成29年4月26日 (2017. 4. 26)

(24) 登録日 平成29年4月7日 (2017. 4. 7)

(51) Int.Cl. F I
G 0 6 F 11/36 (2006.01)
 G 0 6 F 11/36 1 3 6
 G 0 6 F 11/36 1 4 4

請求項の数 8 (全 31 頁)

(21) 出願番号	特願2013-203365 (P2013-203365)	(73) 特許権者	000005223
(22) 出願日	平成25年9月30日 (2013. 9. 30)		富士通株式会社
(65) 公開番号	特開2015-69437 (P2015-69437A)		神奈川県川崎市中原区上小田中4丁目1番1号
(43) 公開日	平成27年4月13日 (2015. 4. 13)	(74) 代理人	100092152
審査請求日	平成28年6月6日 (2016. 6. 6)		弁理士 服部 毅巖
		(72) 発明者	中川 隆
			神奈川県川崎市中原区上小田中4丁目1番1号 富士通株式会社内
		(72) 発明者	鈴木 保利
			神奈川県川崎市中原区上小田中4丁目1番1号 富士通株式会社内
		(72) 発明者	山本 博之
			神奈川県川崎市中原区上小田中4丁目1番1号 富士通株式会社内

最終頁に続く

(54) 【発明の名称】 トレース方法、処理プログラム、および情報処理装置

(57) 【特許請求の範囲】

【請求項 1】

プログラムの実行中に、トレースコードに基づいてトレース処理を実行するための処理プログラムであって、

コンピュータに、

前記プログラムに定義された関数内のトレース位置に記述された第1の命令を、前記トレースコードの実行を指示する第2の命令に書き換えると共に、前記第1の命令を記憶手段に格納し、

前記関数実行中の前記トレース位置の実行時に、前記第2の命令に基づいて前記トレースコードを実行し、

前記トレースコード内に前記関数を呼び出す第3の命令が含まれる場合、前記第3の命令の実行時には、前記関数内の前記トレース位置の前記第2の命令を、前記記憶手段に格納した前記第1の命令に置き換えて、前記関数を実行する、

処理を実行させることを特徴とする処理プログラム。

【請求項 2】

前記トレース位置が前記関数の先頭である場合、

前記第1の命令を格納する際には、前記第1の命令に続けて、前記関数における前記トレース位置の次の命令へのジャンプを指示する第4の命令を、前記記憶手段に格納し、

前記トレースコード実行中の前記第3の命令の実行時には、前記記憶手段に格納された前記第1の命令と第4の命令とを順に実行する、

10

20

ことを特徴とする請求項 1 記載の処理プログラム。

【請求項 3】

前記トレース位置が前記関数の先頭である場合、前記第 1 の命令を格納する際には、前記第 3 の命令の呼び出し先となる関数のアドレスを示す情報を、前記第 1 の命令のアドレスに変更する、

ことを特徴とする請求項 2 記載の処理プログラム。

【請求項 4】

前記第 1 の命令を格納する際には、前記第 1 の命令に続けて、前記関数におけるトレース位置の次の命令へのジャンプを指示する第 4 の命令を、前記記憶手段に格納し、

前記トレースコードの実行中の前記第 3 の命令の実行時には、前記第 3 の命令に基づいて前記関数の先頭から実行を開始し、前記関数の実行位置が前記トレース位置になると、前記記憶手段に格納された前記第 1 の命令と第 4 の命令とを順に実行する、

ことを特徴とする請求項 1 記載の処理プログラム。

【請求項 5】

前記関数の実行位置が前記トレース位置となり、前記トレースコードの実行を開始したときに、前記プログラムを実行している処理単位の識別情報を前記記憶手段に格納し、

前記関数の実行位置が前記トレース位置となったときに、前記プログラムを実行している処理単位の識別情報が既に保存されている場合、前記トレースコードの実行に代えて、前記記憶手段に格納された前記第 1 の命令と第 4 の命令とを順に実行することを特徴とする請求項 4 記載の処理プログラム。

【請求項 6】

前記第 1 の命令を保存する際には、前記第 1 の命令に続けて、前記関数における前記トレース位置の次の命令へのジャンプを指示する第 4 の命令を、前記記憶手段 1 2 に格納し、

前記トレースコードの実行を開始する際に、前記トレース位置が前記関数の先頭か否かを判断し、

前記トレース位置が前記関数の先頭の場合、前記トレースコード実行中の前記第 3 の命令の実行時には、前記記憶手段に格納された前記第 1 の命令と第 4 の命令とを順に実行する、

前記トレース位置が前記関数の先頭ではない場合、前記トレースコードの実行中の前記第 3 の命令の実行時には、前記第 3 の命令に基づいて前記関数の先頭から実行を開始し、前記関数の実行位置が前記トレース位置になると、前記記憶手段に格納された前記第 1 の命令と第 4 の命令とを順に実行する、

ことを特徴とする請求項 1 記載の処理プログラム。

【請求項 7】

プログラムの実行中に、トレースコードに基づいてトレース処理を実行するトレース方法であって、

コンピュータが、

前記プログラムに定義された関数内のトレース位置に記述された第 1 の命令を、前記トレースコードの実行を指示する第 2 の命令に書き換えると共に、前記第 1 の命令を記憶手段に格納し、

前記関数実行中の前記トレース位置の実行時に、前記第 2 の命令に基づいて前記トレースコードを実行し、

前記トレースコード内に前記関数を呼び出す第 3 の命令が含まれる場合、前記第 3 の命令の実行時には、前記関数内の前記トレース位置の前記第 2 の命令を、前記記憶手段に格納した前記第 1 の命令に置き換えて、前記関数を実行する、

トレース方法。

【請求項 8】

プログラムの実行中に、トレースコードに基づいてトレース処理を実行する情報処理装置であって、

前記プログラムに定義された関数内のトレース位置に記述された第1の命令を、トレースコードの実行を指示する第2の命令に書き換えると共に、前記第1の命令を記憶手段に格納する書き換え手段と、

前記関数実行中の前記トレース位置の実行時に、前記第2の命令に基づいて前記トレースコードを実行し、前記トレースコード内に前記関数を呼び出す第3の命令が含まれる場合、前記第3の命令の実行時には、前記関数内の前記トレース位置の前記第2の命令を、前記記憶手段に格納した前記第1の命令に置き換えて、前記関数を実行する実行手段と、
を有する情報処理装置。

【発明の詳細な説明】

【技術分野】

10

【0001】

本発明は、プログラム間の呼び出し制御を行うトレース方法、処理プログラ、および情報処理装置に関する。

【背景技術】

【0002】

近年、OS (Operating System) の機能強化・拡張に伴い、OS の規模が肥大化し、OS において生じる障害は複雑化している。OS の障害の調査手法の一つにトレース技術がある。トレース技術では、プログラム中の特定のコードを実行する時点での、コンピュータの状態に関する情報を採取する技術である。トレース技術では、例えば特定のコードを実行するときのレジスタの内容などの情報が採取される。トレース技術は特に原因解明の困難な複雑化した障害に対して用いられ、今後さらに発展し複雑化していくOS の障害調査にはトレース技術を進展させることが重要である。

20

【0003】

障害調査に有用な技術としては、例えばプログラムの同一の場所で例外が発生した場合であっても、プロセス毎に例外処理を変えることを可能とするデバッグ方法がある。

【先行技術文献】

【特許文献】

【0004】

【特許文献1】特開2006-268298号公報

【発明の概要】

30

【発明が解決しようとする課題】

【0005】

トレース処理において、OS で用意されたAPI (Application Program Interface) などのプログラムを呼び出して処理を実行することができれば、トレース処理の自由度が高まる。例えばトレース処理中にストレージ装置内のファイル操作のAPIを呼び出すことができれば、トレースで採取した情報をストレージ装置に退避させることができる。なお、OS で用意されたAPIなどのプログラムで障害が発生する場合がある。その場合、APIなどのプログラムをトレース対象として、トレース処理を実施することとなる。

【0006】

しかし、従来のトレース技術では、トレース処理中に呼び出されるAPIなどのプログラムをトレース対象とすると、トレース処理内でそのプログラムが再帰的に呼び出されてしまい、処理が無限ループに陥る。そのため、トレース処理中で呼び出されるプログラムをトレース対象とすることができない。

40

【0007】

1つの側面では、本件は、トレース処理を実行することによる処理のループの発生を抑制することを目的とする。

【課題を解決するための手段】

【0008】

1つの案では、プログラムの実行中に、トレースコードに基づいてトレース処理を実行するための処理プログラムであって、コンピュータに以下の処理を実行させる処理プログラ

50

ラムが提供される。当該プログラムを実行するコンピュータは、プログラムに定義された関数内のトレース位置に記述された第1の命令を、トレースコードの実行を指示する第2の命令に書き換えると共に、第1の命令を記憶手段に格納する。またコンピュータは、関数実行中のトレース位置の実行時に、第2の命令に基づいてトレースコードを実行する。トレースコード内に関数を呼び出す第3の命令が含まれる場合、コンピュータは、第3の命令の実行時には、関数内のトレース位置の第2の命令を、記憶手段に格納した第1の命令に置き換えて、関数を実行する。

【発明の効果】

【0009】

1態様によれば、トレース処理を実行することによる処理のループの発生が抑止される 10

。【図面の簡単な説明】

【0010】

【図1】第1の実施の形態に係る情報処理装置の一例を示す図である。

【図2】本実施の形態に用いるコンピュータのハードウェアの一構成例を示す図である。

【図3】トレース機能の一例を示すブロック図である。

【図4】トレースの処理の概要を示す図である。

【図5】ハンドラにおいてトレースポイントを含む関数を呼び出さない場合の命令の一例を示す図である。

【図6】ハンドラにおいてトレースポイントを含む関数を呼び出すことによるループ発生 20
例を示す図である。

【図7】第2の実施の形態におけるハンドラからのトレース対象のプログラムの呼び出し例を示す図である。

【図8】トレース設定ファイルの一例を示す図である。

【図9】トレース設定処理の手順の一例を示すフローチャートである。

【図10】トレース管理テーブルの一例を示す図である。

【図11】トレース管理テーブル作成処理の手順を示すフローチャートである。

【図12】トレース設定ファイル読み込み処理の手順の一例を示すフローチャートである

。【図13】IAT書き換え処理の手順の一例を示すフローチャートである。 30

【図14】APIリストの一例を示す図である。

【図15】トラップ例外処理の手順の一例を示すフローチャートである。

【図16】トレースデータ採取処理の手順の一例を示すフローチャートである。

【図17】先頭以外の箇所をトレースポイントとした場合のループ発生例を示す図である

。【図18】第3の実施の形態におけるハンドラからのトレース対象の関数の呼び出し例を示す図である。

【図19】第3の実施の形態におけるトレース設定ファイルの一例を示す図である。

【図20】第3の実施の形態におけるトレース設定処理の手順の一例を示すフローチャートである。 40

【図21】第3の実施の形態におけるトラップ例外処理の手順の一例を示すフローチャートである。

【図22】TID管理テーブルの一例を示す図である。

【図23】スレッド判別処理の手順の一例を示すフローチャートである。

【図24】第3の実施の形態におけるトレースデータ採取処理の手順の一例を示すフローチャートである。

【図25】第4の実施の形態におけるトレース管理テーブルの一例を示す図である。

【図26】第4の実施の形態におけるトレース設定ファイル読み込み処理の手順の一例を示すフローチャートである。

【図27】トラップ例外処理の手順の一例を示すフローチャートである。 50

【図 28】トレースアドレス判別処理の手順の一例を示すフローチャートである。

【発明を実施するための形態】

【0011】

以下、本実施の形態について図面を参照して説明する。なお各実施の形態は、矛盾のない範囲で複数の実施の形態を組み合わせる実施することができる。

〔第 1 の実施の形態〕

図 1 は、第 1 の実施の形態に係る情報処理装置の一例を示す図である。第 1 の実施の形態に係る情報処理装置 10 は、プログラム 1 の実行中に、トレースコード 2 に基づいてトレース処理を実行する。トレース処理は、例えば、プログラム 1 実行中の所定の時点でのメモリやレジスタの内容を取得する処理である。情報処理装置 10 は、トレース処理を実現するために、書き換え手段 11、記憶手段 12、および実行手段 13 を有する。情報処理装置 10 は、例えばプロセッサやメモリを有するコンピュータである。

10

【0012】

書き換え手段 11 は、プログラム 1 に定義された関数 1 a 内のトレース位置に記述された第 1 の命令 4 を、トレースコード 2 の実行を指示する第 2 の命令 5 に書き換える（ステップ S 1）。例えばトレースコード 2 が例外処理を実行するハンドラ内に記述されたコードの場合、第 2 の命令 5 は例外命令である。命令の書き換えは、メモリ上で行われる。例えばプログラム 1 が展開されたメモリ内の第 1 の命令 4 が格納された記憶領域に、第 2 の命令 5 が書き込まれる。これにより、プログラム 1 内にトレースコードをフック（処理を追加すること）することができる。

20

【0013】

トレースコード 2 には、プログラム 1 における第 1 の命令 4 の実行時に採取すべき情報の採取手順が記述されている。ここでトレースコード 2 において高度な処理を行うために、プログラム 1 内の関数 1 a の処理が有効な場合がある。その場合、トレースコード 2 内に関数 1 a を呼び出す第 3 の命令 6 が記述される。

【0014】

また書き換え手段 11 は、第 2 の命令 5 に書き換えられた第 1 の命令 4 を、記憶手段 12 に格納する（ステップ S 2）。例えば書き換え手段 11 は、第 1 の命令 4 に続けて、プログラム 1 におけるトレース位置の次の命令へのジャンプを指示する第 4 の命令 7 を記述したスタブ 3 を生成する。そして書き換え手段 11 は、スタブ 3 を記憶手段 12 に格納する。

30

【0015】

記憶手段 12 は、第 1 の命令 4 を記憶する。例えば記憶手段 12 は、第 1 の命令 4 を含むスタブ 3 を記憶する。

実行手段 13 は、関数 1 a 実行中のトレース位置の実行時に、第 2 の命令 5 に基づいてトレースコード 2 を実行する（ステップ S 3）。例えば第 2 の命令 5 が例外命令であれば、実行手段 13 は、例外ハンドラとして定義されているトレースコード 2 を呼び出し、呼び出したトレースコード 2 を実行する。

【0016】

実行手段 13 は、トレースコード 2 内に関数 1 a を呼び出す第 3 の命令 6 が含まれる場合、第 3 の命令 6 の実行時には、関数 1 a 内のトレース位置の第 2 の命令 5 を、記憶手段 12 に格納した第 1 の命令 4 に置き換えて、関数 1 a を実行する。

40

【0017】

例えば実行手段 13 は、第 3 の命令 6 に基づいて、プログラム 1 内の関数 1 a を呼び出して、関数 1 a を実行する（ステップ S 4）。そして実行手段 13 は、関数 1 a の実行中に第 2 の命令 5 の実行順になると、第 2 の命令 5 に示されているトレースコード 2 の実行に代えて、スタブ 3 を実行する（ステップ S 5）。

【0018】

スタブ 3 には、第 1 の命令 4 が記述されている。そのため、スタブ 3 を実行することにより、第 2 の命令 5 の実行に代えて、第 1 の命令 4 が実行されることとなる。第 1 の命令

50

4の実行後、実行手段13がスタブ3に記述されている第4の命令7を実行することにより、関数1aにおけるトレース位置の次の命令に処理が進められる(ステップS6)。

【0019】

その後、実行手段13により関数1aのトレース位置の次の命令から順に処理が実行される。関数1aの実行が終了すると、第3の命令6に基づくサブルーチンの処理が終了したこととなり、トレースコード2の第3の命令6の次の命令に処理が進められる(ステップS7)。実行手段13は、トレースコード2の実行が終了すると、次にスタブ3を実行する(ステップS8)。トレースコード2の後にスタブ3を実行することにより、第2の命令5によって書き換えられた第1の命令4が実行される。スタブ3の実行が終了すると、関数1a内のトレース位置の次の命令に処理が進められる(ステップS9)。

10

【0020】

このようにして、プログラム1内の関数1aの実行中に挿入するトレースコード2において関数1aの呼び出しが可能となる。すなわち、トレースコード2においてプログラムを呼び出しても、処理のループが発生しない。その結果、トレースコード2に課される制約が抑止され、トレースコード2で実行する処理の自由度が高くなる。

【0021】

例えばプログラム実行中の障害解析を行う場合、デバッグ対象のプログラムは様々である。障害解析に関する処理をトレースコード2で記述しておき、プログラム1内の関数1aの実行中に障害が発生したときに、その関数1a内の位置をトレース位置に指定して障害解析を行う場合がある。このとき第1の実施の形態に示すようなループの発生を抑止する処理を実施しないと、トレースコード2中で呼び出す関数1aについては、障害の解析ができなくなってしまう。コンピュータの障害解析において、解析ができないプログラムは少ない方がよく、そのためには、障害解析の処理が記述されたトレースコード2において、他のプログラムの呼び出しを行わないようにすることとなる。ところが、障害解析で利用できるプログラムに制限があると、高度な障害解析が難しくなってしまう。特にOSで提供される処理(例えばファイルアクセス)を用いるには、その処理に対応するAPIなどの関数の呼び出しを行うことになる。解析処理においてそのようなAPIの呼び出しができなければ、OSの機能を利用した高度な障害解析が困難となる。

20

【0022】

他方、第1の実施の形態に示す処理を実施することで、障害解析の手順を記述したトレースコード2中で、APIなどの各種関数の呼び出しが可能となる。その結果、高度な障害解析が可能となる。

30

【0023】

しかも、関数1a内のトレース位置に、トレースコード2をフックするようにすることで、トレースコード2を関数1aと同じスレッドで実行することができる。同じスレッドで実行させることで、トレースコード2の実行開始前後でコンテキストスイッチを行わずに済む。コンテキストスイッチを行わないことで、トレース位置での関数1aの実行状態を変化させずに済み、正しい情報のトレースが可能となる。

【0024】

さらに、第1の命令4の第2の命令5への書き換えを、メモリ上で行うことで、プログラム1の実行ファイルの内容を編集せずに済む。その結果、再コンパイルなしに、トレース処理を行うことができる。

40

【0025】

なお、図1に示す書き換え手段11と実行手段13とは、例えば情報処理装置10が有するプロセッサにより実現することができる。また、記憶手段12は、例えば情報処理装置10が有するメモリにより実現することができる。

【0026】

また、図1に示した各要素間を接続する線は通信経路の一部を示すものであり、図示した通信経路以外の通信経路も設定可能である。

[第2の実施の形態]

50

次に第２の実施の形態について説明する。第２の実施の形態は、ＯＳで用意されたＡＰＩの実行開始時におけるコンピュータの状態を示す情報をトレース（データ採取）するものである。ＡＰＩには、ＯＳの機能を利用するための関数などのプログラムが含まれており、アプリケーションプログラムなどを実行中にＡＰＩを呼び出すことで、ＯＳの機能を利用することができる。

【００２７】

図２は、本実施の形態に用いるコンピュータのハードウェアの一構成例を示す図である。コンピュータ１００は、プロセッサ１０１によって装置全体が制御されている。プロセッサ１０１には、バス１０９を介してメモリ１０２と複数の周辺機器が接続されている。プロセッサ１０１は、マルチプロセッサであってもよい。プロセッサ１０１は、例えばＣ
ＰＵ（Central Processing Unit）、ＭＰＵ（Micro Processing Unit）、またはＤＳＰ（
Digital Signal Processor）である。プロセッサ１０１の機能の少なくとも一部を、ＡＳ
ＩＣ（Application Specific Integrated Circuit）、ＰＬＤ（Programmable Logic Devi
ce）などの電子回路で実現してもよい。

10

【００２８】

メモリ１０２は、コンピュータ１００の主記憶装置として使用される。メモリ１０２には、プロセッサ１０１に実行させるＯＳのプログラムやアプリケーションプログラムの少なくとも一部が一時的に格納される。また、メモリ１０２には、プロセッサ１０１による処理に必要な各種データが格納される。メモリ１０２としては、例えばＲＡＭ（Random Access Memory）などの揮発性の半導体記憶装置が使用される。

20

【００２９】

バス１０９に接続されている周辺機器としては、ＨＤＤ（Hard Disk Drive）１０３、グラフィック処理装置１０４、入力インタフェース１０５、光学ドライブ装置１０６、機器接続インタフェース１０７およびネットワークインタフェース１０８がある。

【００３０】

ＨＤＤ１０３は、内蔵したディスクに対して、磁気的にデータの書き込みおよび読み出しを行う。ＨＤＤ１０３は、コンピュータ１００の補助記憶装置として使用される。ＨＤＤ１０３には、ＯＳのプログラム、アプリケーションプログラム、および各種データが格納される。なお、補助記憶装置としては、フラッシュメモリなどの不揮発性の半導体記憶装置を使用することもできる。

30

【００３１】

グラフィック処理装置１０４には、モニタ２１が接続されている。グラフィック処理装置１０４は、プロセッサ１０１からの命令に従って、画像をモニタ２１の画面に表示させる。モニタ２１としては、ＣＲＴ（Cathode Ray Tube）を用いた表示装置や液晶表示装置などがある。

【００３２】

入力インタフェース１０５には、キーボード２２とマウス２３とが接続されている。入力インタフェース１０５は、キーボード２２やマウス２３から送られてくる信号をプロセッサ１０１に送信する。なお、マウス２３は、ポインティングデバイスの一例であり、他のポインティングデバイスを使用することもできる。他のポインティングデバイスとしては、タッチパネル、タブレット、タッチパッド、トラックボールなどがある。

40

【００３３】

光学ドライブ装置１０６は、レーザ光などを利用して、光ディスク２４に記録されたデータの読み取りを行う。光ディスク２４は、光の反射によって読み取り可能なようにデータが記録された可搬型の記録媒体である。光ディスク２４には、ＤＶＤ（Digital Versatile Disc）、ＤＶＤ－ＲＡＭ、ＣＤ－ＲＯＭ（Compact Disc Read Only Memory）、ＣＤ－Ｒ（Recordable）／ＲＷ（ReWritable）などがある。

【００３４】

機器接続インタフェース１０７は、コンピュータ１００に周辺機器を接続するための通信インタフェースである。例えば機器接続インタフェース１０７には、メモリ装置２５や

50

メモリーリーダーライタ 26 を接続することができる。メモリー装置 25 は、機器接続インタフェース 107 との通信機能を搭載した記録媒体である。メモリーリーダーライタ 26 は、メモリーカード 27 へのデータの書き込み、またはメモリーカード 27 からのデータの読み出しを行う装置である。メモリーカード 27 は、カード型の記録媒体である。

【0035】

ネットワークインタフェース 108 は、ネットワーク 20 に接続されている。ネットワークインタフェース 108 は、ネットワーク 20 を介して、他のコンピュータまたは通信機器との間でデータの送受信を行う。

【0036】

以上のようなハードウェア構成によって、第 2 の実施の形態の処理機能を実現することができる。なお、第 1 の実施の形態に示した情報処理装置 10 も、図 2 に示したコンピュータ 100 と同様のハードウェアにより実現することができる。

【0037】

コンピュータ 100 は、例えばコンピュータ読み取り可能な記録媒体に記録されたプログラムを実行することにより、第 2 の実施の形態の処理機能を実現する。コンピュータ 100 に実行させる処理内容を記述したプログラムは、様々な記録媒体に記録しておくことができる。例えば、コンピュータ 100 に実行させるプログラムを HDD 103 に格納しておくことができる。プロセッサ 101 は、HDD 103 内のプログラムの少なくとも一部をメモリ 102 にロードし、プログラムを実行する。またコンピュータ 100 に実行させるプログラムを、光ディスク 24、メモリー装置 25、メモリーカード 27 などの可搬型記録媒体に記録しておくこともできる。可搬型記録媒体に格納されたプログラムは、例えばプロセッサ 101 からの制御により、HDD 103 にインストールされた後、実行可能となる。またプロセッサ 101 が、可搬型記録媒体から直接プログラムを読み出して実行することもできる。

【0038】

図 3 は、トレース機能の一例を示すブロック図である。図 3 には、コンピュータ 100 が有する機能のうち、トレースによる情報採取に係る機能を抜き出したものである。コンピュータ 100 は、プログラム記憶部 110、コマンド部 120、ドライバ部 130、およびプール領域 140 を有する。なお、コンピュータ 100 は、図 3 に示した機能以外にも、OS により実現される機能を有している。

【0039】

プログラム記憶部 110 は、トレース対象のプログラムを記憶する。例えばプログラム記憶部 110 は、メモリ 102 内の記憶領域である。

コマンド部 120 は、ユーザから、トレース指示のコマンドの入力を受け付ける。そしてコマンド部 120 は、トレース要求をドライバ部 130 に送信する。またコマンド部 120 は、ドライバ部 130 からトレース結果を取得し、トレース結果を含むログファイルを出力する。なおコマンド部 120 は、OS のユーザモードで動作する。

【0040】

ドライバ部 130 は、コマンド部 120 からのトレース要求に応じて、プログラム記憶部 110 に格納されたトレース対象のプログラムのトレースを行う。ドライバ部 130 は、トレースにより取得した情報を、プール領域 140 に格納する。またドライバ部 130 は、トレースに用いるスタブも、プール領域 140 に格納する。またドライバ部 130 は、トレースが完了すると、トレース結果をコマンド部 120 に送信する。なおドライバ部 130 は、OS のカーネルモードで動作する。

【0041】

プール領域 140 は、トレースデータやスタブの記憶領域である。例えばプール領域 140 は、メモリ 102 内の記憶領域である。

なお、コマンド部 120 は、例えば「trace.exe」というファイル名の実行形式のファイルをコンピュータ 100 が実行することで実現する機能である。またドライバ部 130 は、例えば「TraceDriver.sys」というファイル名の実行形式のファイルをコンピュータ

10

20

30

40

50

100が実行することで実現する機能である。

【0042】

このような機能によりトレースが実行される。

図4は、トレースの処理の概要を示す図である。ユーザは、キーボード22などの入力デバイスを操作し、トレースの実行を指示するコマンド31を入力する。コマンド31では、例えばトレース対象のプログラム111が指定される。またコマンド31では、トレースの実行条件が定義されたトレース設定ファイル32が指定される。

【0043】

入力されたコマンド31は、ユーザモードで動作するコマンド部120で受け取られる。コマンド部120は、例えばコマンド31で指定されたトレース設定ファイル32を参照し、トレースの実行条件を認識する。そしてコマンド部120は、トレースの実行条件を含むトレース要求を出力する。

10

【0044】

トレース要求は、OSのI/Oマネージャ151を介して、カーネルモードで動作するドライバ部130に入力される。ドライバ部130は、トレース要求に従って、トレース処理を実行する。トレース処理は、トレース要求取得時に実行するトレース設定処理と、トレースポイントにおける例外発生時に実行するトレースデータ収集処理とに大別される。図4では、トレース設定処理を実線の矢印で示し、トレースデータ収集処理を破線の矢印で示している。

20

【0045】

トレース設定処理では、ドライバ部130は、トレース対象のプログラム111のトレースポイントとなる位置の命令を、割り込み命令111aに書き換える。またドライバ部130は、IDT (Interrupt Descriptor Table) における例外割り込み用のハンドラ131のアドレスを、ドライバ部130が有するハンドラ131のアドレスに書き換える。さらにドライバ部130は、プログラム111の書き換えによって上書きされた命令を実行するスタブ141を生成し、プール領域140に格納する。

【0046】

その後、コンピュータ100においてプログラム111の実行が開始され、実行位置がトレースポイントに達すると、割り込み命令111aが実行される。割り込み命令111aが実行されると、OSによってIDT152が参照され、その割り込み命令111aに対応する例外割り込み用のハンドラ131のアドレスが取得される。そしてハンドラ131の実行が開始される。ハンドラ131は、トレースデータを収集し、収集したトレースデータ142をプール領域140に格納する。トレースデータの収集が完了すると、ハンドラ131は、スタブ141を起動する。するとコンピュータ100においてスタブ141が実行される。スタブ141の実行後、コンピュータ100は、プログラム111内の割り込み命令111aより後の命令を実行する。

30

【0047】

ドライバ部130は、トレースデータ142をI/Oマネージャ151経由でコマンド部120に送信する。するとコマンド部120は、トレースデータ142を含むログファイル33を生成する。ログファイル33は、例えばHDD103に格納される。

40

【0048】

このような手順でプログラム実行中の所定のトレースポイントにおけるコンピュータの状態を示す情報（レジスタの値など）が収集される。どのような情報を収集するかは、ハンドラ131に定義されている。トレースポイントとしては、例えばプログラム111内に定義されたAPIなどの関数に含まれる命令を指定することができる。ここで、ハンドラ131において、トレースポイントとして指定された命令を含む関数を呼び出すか否かにより、トレースの手順が異なる。

【0049】

図5は、ハンドラにおいてトレースポイントを含む関数を呼び出さない場合の命令の一例を示す図である。図5の例では、プログラム40内に「FunctionA」という関数40ー

50

1 が定義されている。そして関数 40-1 の実行開始時にトレースを実行するものとする。この場合、関数 40-1 の先頭の命令 41 が、メモリ 102 内で例外命令 42 に書き換えられる。元の命令 41 は、メモリ 102 のアドレス「fffff800`014e6610」から「4883ec10」と記述されている。この命令 41 をニーモニックで表すと「sub rsp,10h」となる。書き換え後の例外命令 42 は、メモリ 102 のアドレス「fffff800`014e6610」から「cc」と記述されている。この例外命令 42 をニーモニックで表すと「int 3」となる。

【0050】

例外命令 42 は、書き換え元の命令 41 よりも命令長が短い。そのため、命令 41 が格納されたメモリ 102 内の領域の一部が例外命令 42 で上書きされ、他の領域には不完全な命令 43 が残される。そこで、元の命令 41 を実行するためのスタブ 141a が生成される。スタブ 141a には、例外命令 42 によって上書きされた元の命令 41 と同じ内容の命令 44 が含まれている。またスタブ 141a の最後には、元のプログラム 40 において、書き換えられた命令 41 の次に実行することになっていた命令 46 へのジャンプ命令 45 が記述されている。

10

【0051】

命令書き換え後のプログラム 40a 内の関数 40a-1 が実行されると、まず例外命令 42 が実行される。例外命令 42 が実行されることで、例外処理に対応するハンドラ 131a が起動される。そしてハンドラ 131a が実行されることによって、データが採取される。ハンドラ 131a の実行が終了すると、スタブ 141a が実行される。すると、スタブ 141a 内に記述された、書き換え元の命令 41 と同じ内容の命令 44 が実行される。次に、プログラム 40a 内の命令 46 へのジャンプ命令 45 が実行される。その後、プログラム 40a の命令 46 以降の各命令が実行される。

20

【0052】

このようにして、プログラム 40 をトレース対象として、プログラム 40 の開始時点での状態をトレースすることができる。ただし、図 5 の例では、ハンドラ 131a 内に、トレースポイントを含む関数 40a-1 の呼出命令が含まれていない。ハンドラ 131a 内に、関数 40a-1 の呼出命令が含まれている場合、図 5 と同様の処理を行うと、処理のループが発生する。

【0053】

図 6 は、ハンドラにおいてトレースポイントを含む関数を呼び出すことによるループ発生例を示す図である。図 6 の例では、トレース処理の内容が記述されたハンドラ 131b 内に、トレースポイントを有する関数 40a-1 を呼び出す命令 51 が含まれている。この場合、トレース実施中に関数 40a-1 が実行されると、まず例外命令 42 が実行され、ハンドラ 131b が呼び出される。呼び出されたハンドラ 131b 内の命令が先頭から実行される。そして命令 51 が実行されることで、関数 40a-1 が呼び出され、先頭の例外命令 42 が実行される。すると、再度例外処理が発生し、ハンドラ 131b が呼び出される。

30

【0054】

このように、トレース対象のプログラムをハンドラ 131b 内で呼び出している場合、ループ処理が繰り返される。処理が何度もループしていると、再帰的な割り込みの繰り返しにより、処理の呼び出しを管理するスタックがオーバーフローする。その結果、トレース処理がエラー終了する。

40

【0055】

第 2 の実施の形態では、ハンドラ 131b 内でトレース対象の関数 40a-1 を呼び出す場合の呼び出し先をスタブ 141a に変更することで、図 6 に示すようなループの発生を回避する。ハンドラ 131b 内での呼び出し先の変更は、例えば IAT (Import Address Table) の内容を変更することで実現できる。IAT は、外部ファイルで定義された API の先頭アドレスが記述されたデータテーブルである。

【0056】

図 7 は、第 2 の実施の形態におけるハンドラからのトレース対象のプログラムの呼び出

50

し例を示す図である。ドライバ部130は、データセクション132、コードセクション133、IAT134などを有している。データセクション132は、ドライバ部130の処理の実行に使用するデータが格納される。コードセクション133は、ドライバ部130が実行する処理内容を示すプログラムコードが格納される。なおハンドラ131bも、コードセクション133に含まれる。IAT134には、ドライバ部130が処理の実行に使用するAPIの名称に対応付けて、そのAPIを定義する外部ファイルの先頭アドレスが設定されている。

【0057】

関数40a-1をトレース対象としてトレースを実施する場合において、ハンドラ131b内に関数40a-1の呼び出し命令(call命令)があると、ドライバ部130は、自身
10 自身が有するIATの書き換えを行う。例えばドライバ部130は、関数40a-1の名前「FunctionA」の先頭アドレス「ffff800`014e6610」を、スタブ141aの先頭アドレスに変更する。

【0058】

その後、関数40a-1が実行されると、例外命令42によりハンドラ131bが呼び出される。そしてハンドラ131bに基づいて、データが収集される。ハンドラ131bにおける関数40a-1呼び出しの命令51が実行されると、IAT134が参照され、命令51で示されるAPIの名称に対応付けられたアドレスのプログラムが呼び出される。該当アドレスは、予めスタブ141aの先頭アドレスに変更されているため、スタブ141aが呼び出され、実行される。
20

【0059】

スタブ141aを実行することで、例外命令42で上書きされた命令41と同じ内容の命令44が実行され、関数40a-1の命令46に処理が遷移する。その後、関数40a-1の命令46以降の処理が実行される。関数40a-1が最後まで実行されると、呼出元のハンドラ131bに処理が遷移する。

【0060】

ハンドラ131bからの呼び出しによる関数40a-1の処理が終了すると、ハンドラ131bの残りの処理が実行される。ハンドラ131bの処理が完了すると、スタブ141a内の命令44が実行される。次に、ジャンプ命令45により、関数40a-1の命令46にジャンプする。その後、関数40a-1の命令46以降の処理が実行される。
30

【0061】

次に、トレース設定処理について詳細に説明する。トレース設定処理は、トレース設定ファイル32に基づいて行われる。

図8は、トレース設定ファイルの一例を示す図である。トレース設定ファイル32には、例えばデータバッファサイズやトレースポイントが指定される。図8の例では、[SETTING]の欄に、採取したデータを格納するデータバッファのサイズが設定されている。また[TRACEPOINT]の欄に、トレースポイントが指定されている。トレースポイントは、例えばトレース対象のプログラムの名称(シンボル名)で指定される。トレースポイントがシンボル名のみで示された場合、そのシンボル名に対応するプログラムの先頭の位置がトレースポイントとなる。また、トレースポイントをメモリ内でのアドレスで指定することもできる。
40

【0062】

次に、トレース設定処理の手順について説明する。

図9は、トレース設定処理の手順の一例を示すフローチャートである。

[ステップS101] ドライバ部130は、トレース管理テーブルを作成する。トレース管理テーブルは、トレースポイントにおける命令と、その命令を実行するためのスタブアドレスとを管理するデータテーブルである。トレース管理テーブル作成処理の詳細は後述する(図11参照)。

【0063】

[ステップS102] ドライバ部130は、IATの書き換えを行う。IAT書き換え
50

処理の詳細は後述する（図 1 3 参照）。なお書き換えられた I A T の内容は、トレース処理を伴う情報解析が終了したときに、元に戻される。

【0064】

〔ステップ S 1 0 3〕ドライバ部 1 3 0 は、ハンドラを置き換える。例えばドライバ部 1 3 0 は、例外処理が発生したときに、トレース処理を行うハンドラ 1 3 1 b が実行されるように、O S の設定内容を更新する。

【0065】

〔ステップ S 1 0 4〕ドライバ部 1 3 0 は、トレースポイントとなっているアドレスに記述されている命令を、例外命令に置き換える。

このような手順でトレースの実行環境が整えられる。トレースポイントは、トレース管理テーブルに示される。

10

【0066】

図 1 0 は、トレース管理テーブルの一例を示す図である。トレース管理テーブル 6 0 には、トレースアドレス、元の命令、およびスタブアドレスの欄が設けられている。トレースアドレスの欄には、トレースポイントとなる命令が格納されたメモリ 1 0 2 のアドレス（トレースアドレス）が設定される。元の命令の欄には、例外命令に書き換える前に、トレースアドレスに述されていた命令が設定される。スタブアドレスの欄には、元の命令を実行するためのスタブの先頭アドレスが設定される。

【0067】

次にトレース管理テーブル作成処理について詳細に説明する。

20

図 1 1 は、トレース管理テーブル作成処理の手順を示すフローチャートである。

〔ステップ S 1 1 1〕ドライバ部 1 3 0 は、トレース設定ファイル読み込み処理を行う。トレース設定ファイル読み込み処理により、トレース管理テーブル 6 0 のトレースアドレスの欄に、トレースポイントのアドレスが設定される。この処理の詳細は後述する（図 1 2 参照）。ここで、トレース管理テーブル 6 0 に設定されたトレースポイントの数を m とする（ m は 1 以上の整数）。

【0068】

〔ステップ S 1 1 2〕ドライバ部 1 3 0 は、変数 i に 0 を設定する。

〔ステップ S 1 1 3〕ドライバ部 1 3 0 は、 i の値がトレースポイント数 m と同じか否かを判断する。 i の値がトレースポイント数 m と同じ場合、トレース管理テーブル作成処理が終了する。 i の値がトレースポイント数 m と異なる場合、処理がステップ S 1 1 4 に進められる。

30

【0069】

〔ステップ S 1 1 4〕ドライバ部 1 3 0 は、トレース管理テーブル 6 0 の i 番目のレコードのトレースポイントに記述されている命令を、そのレコードの元の命令の欄に格納する。

【0070】

〔ステップ S 1 1 5〕ドライバ部 1 3 0 は、スタブの格納領域をメモリ 1 0 2 内に作成する。

〔ステップ S 1 1 6〕ドライバ部 1 3 0 は、スタブの格納領域の先頭のアドレスを、トレース管理テーブル 6 0 の i 番目のレコードのスタブアドレスの欄に格納する。

40

【0071】

〔ステップ S 1 1 7〕ドライバ部 1 3 0 は、 i の値に 1 を加算し、処理をステップ S 1 1 3 に進める。

次にトレース設定ファイル読み込み処理の手順について説明する。

【0072】

図 1 2 は、トレース設定ファイル読み込み処理の手順の一例を示すフローチャートである。

〔ステップ S 1 2 1〕ドライバ部 1 3 0 は、トレース設定ファイル 3 2 の [S E T T I N G] の欄から、データサイズを読み取る。ドライバ部 1 3 0 は、読み取ったデータサイ

50

ズ分の記憶領域を、トレースデータ142（図4参照）の格納領域としてメモリ102内に確保する。

【0073】

〔ステップS122〕ドライバ部130は、変数*i*に0を設定する。

〔ステップS123〕ドライバ部130は、*i*の値がトレースポイント数*m*と同じか否かを判断する。*i*の値がトレースポイント数*m*と同じ場合、トレース管理テーブル作成処理が終了する。*i*の値がトレースポイント数*m*と異なる場合、処理がステップS124に進められる。

【0074】

〔ステップS124〕ドライバ部130は、トレース設定ファイル32の〔TRACE POINT〕の欄に設定されている全*m*個のトレースポイントそれぞれの定義行から、*i*番目のトレースポイントの定義情報を読み取る。

10

【0075】

〔ステップS125〕ドライバ部130は、トレースポイントがシンボル名で定義されているか否かを判断する。トレースポイントがシンボル名で定義されていれば、処理がステップS126に進められる。トレースポイントがアドレスで定義されていれば、処理がステップS127に進められる。

【0076】

〔ステップS126〕ドライバ部130は、シンボル名をアドレスに変換する。例えばドライバ部130は、OSに用意されたAPIを用いて、シンボル名に対応するプログラム（関数などのAPIを含む）の先頭アドレスを、OSから取得する。そしてドライバ部130は、取得したアドレスを、*i*番目のトレースポイントのトレースアドレスと認識する。

20

【0077】

〔ステップS127〕ドライバ部130は、*i*番目のトレースポイントのトレースアドレスを、トレース管理テーブル60に登録する。

〔ステップS128〕ドライバ部130は、*i*の値に1を加算し、処理をステップS123に進める。

【0078】

図11、図12に示した処理により、トレース管理テーブル60が作成される。トレース管理テーブル60が作成されると、IATの書き換えが行われる。

30

図13は、IAT書き換え処理の手順の一例を示すフローチャートである。

【0079】

〔ステップS131〕ドライバ部130は、ドライバ部130の処理内容を記述した実行形式のファイル（TraceDriver.sys）のロードアドレス（メモリに読み込まれたファイルの先頭アドレス）を、OSから取得する。

【0080】

〔ステップS132〕ドライバ部130は、ドライバ部130の処理内容を記述した実行形式のファイルを解析し、IAT134が格納された領域のアドレスを取得する。

〔ステップS133〕ドライバ部130は、IAT134に基づいて、ハンドラ内で使用されているAPIのリスト（APIリスト）を作成する。例えばドライバ部130は、ハンドラ内で使用されているAPIの名前をアドレスに変換し、アドレス順にソートする。

40

【0081】

図14は、APIリストの一例を示す図である。APIリスト62には、API名とアドレスとの欄が設けられている。API名には、ハンドラ131b内で使用されるAPIの名称が設定される。アドレスの欄には、対応するAPIなどの関数が格納された領域のアドレスが設定される。API名とアドレスとの組は、IATから取得できる。API名とアドレスとの組からなる各レコードは、例えばアドレスが昇順となるようにソートされている。ここでAPIリスト62に登録されたAPI数を*n*（*n*は1以上の整数）とする

50

。

【0082】

図13の説明に戻る。

〔ステップS134〕ドライバ部130は、変数*i*に0を設定する。

〔ステップS135〕ドライバ部130は、*i*の値がAPI数*n*と同じか否かを判断する。*i*の値がAPI数*n*と同じ場合、IAT書き換え処理が終了する。*i*の値がAPI数*n*と異なる場合、処理がステップS136に進められる。

【0083】

〔ステップS136〕ドライバ部130は、APIリスト62の*i*番目のAPIがトレース対象か否かを判断する。例えば*i*番目のAPIのアドレスと同じアドレスが、トレース管理テーブル60のトレースアドレスの欄に登録されている場合、*i*番目のAPIがトレース対象であると判断される。*i*番目のAPIがトレース対象の場合、処理がステップS137に進められる。*i*番目のAPIがトレース対象ではない場合、処理がステップS138に進められる。

10

【0084】

〔ステップS137〕ドライバ部130は、APIリスト62の*i*番目のAPIに対応するIAT内のエントリを、そのAPIに対応するスタブのアドレスに変換する。例えばドライバ部130は、APIリスト62の*i*番目のAPIのアドレスとトレースアドレスとするレコードを、トレース管理テーブル60から取得する。次にドライバ部130は、取得したレコードからスタブアドレスを抽出する。そしてドライバ部130は、APIリスト62の*i*番目のAPIに対応するIAT内のエントリのアドレスと、抽出したスタブアドレスを書き換える。

20

【0085】

〔ステップS138〕ドライバ部130は、*i*の値に1を加算し、処理をステップS135に進める。

このようにしてIATの書き換えが行われる。その後、図9に示すように、ハンドラの置き換えとトレースポイントの命令の置き換えが行われ、トレース設定が完了する。トレース設定が完了した後に、トレース対象のAPIが呼び出されると、例外命令が実行される。その結果、トラップ例外処理が開始される。

【0086】

図15は、トラップ例外処理の手順の一例を示すフローチャートである。

30

〔ステップS141〕ドライバ部130は、トレースアドレスで発生した例外か否かを判断する。例えばドライバ部130は、トレース管理テーブル60を参照し、例外を発生させた例外命令のアドレスが、トレースアドレスとして登録されている場合、トレースアドレスで発生した例外であると判断する。トレースアドレスで発生した例外の場合、処理がステップS143に進められる。トレースアドレス以外で発生した例外の場合、処理がステップS142に進められる。

【0087】

〔ステップS142〕ドライバ部130は、トレースアドレス以外で発生した例外であれば、ステップS103（図9参照）による置き換え前の例外ハンドラを呼び出す。すると、呼び出された例外ハンドラが実行され、トレース処理を伴わない例外処理が行われる。その後、トラップ例外処理が終了する。

40

【0088】

〔ステップS143〕ドライバ部130は、ハンドラ131bに基づいて、トレースデータの採取処理を行う。この処理の詳細は後述する（図16参照）。

〔ステップS144〕ドライバ部130は、トレース対象の関数に対応するスタブを実行する。例えばハンドラ131bの最後の命令として、スタブ141aの先頭へのジャンプ命令が記載されている。ドライバ部130は、その命令を実行することで、処理対象をスタブ141aに遷移させ、スタブ141aに記述された命令を実行する。

【0089】

50

〔ステップS145〕ドライバ部130は、例外発生元に処理を戻す。これにより、例外発生元のプログラムを実行しているスレッドにより、例外発生アドレスの次の命令以降の処理が実行される。

【0090】

次に、トレースデータ採取処理について詳細に説明する。

図16は、トレースデータ採取処理の手順の一例を示すフローチャートである。

〔ステップS151〕ドライバ部130は、ハンドラ131bを呼び出し、ハンドラ131b内の次に実行する命令を解釈する。

【0091】

〔ステップS152〕ドライバ部130は、実行対象の命令が、例外発生元の関数（API）の呼び出し命令か否かを判断する。該当関数の呼び出し命令の場合、処理がステップS154に進められる。該当関数の呼び出し命令でなければ、処理がステップS153に進められる。

10

【0092】

〔ステップS153〕ドライバ部130は、ハンドラ131bの次に実行する命令が、例外発生元のAPIの呼び出し命令でなければ、その命令を実行する。その後、処理がステップS157に進められる。

【0093】

〔ステップS154〕ドライバ部130は、ハンドラ131bの次に実行する命令が、例外発生元のAPIの呼び出し命令であれば、該当関数に対応するスタブを実行する。例えばドライバ部130は、IAT134を参照し、例外発生元の関数に対応付けて設定されたスタブアドレスを取得する。そしてドライバ部130は、取得したスタブアドレスに格納されているスタブ内の命令を実行する。そのスタブには、例外発生元の関数において、例外命令に置き換えられた命令、および例外発生元の関数における例外命令の次の命令へのジャンプ命令が記述されている。従って、ドライバ部130がスタブを実行することにより、例外命令に置き換えられた命令が実行され、例外発生元の関数における例外命令の次の命令へのジャンプ命令が実行される。

20

【0094】

〔ステップS155〕ドライバ部130は、例外発生元の関数における例外発生アドレスの次の命令以降の処理を実行する。

30

〔ステップS156〕ドライバ部130は、例外発生元の関数の処理が最後まで終了すると、処理がハンドラ131bに戻す。

【0095】

〔ステップS157〕ドライバ部130は、ハンドラ131bの処理が終了したか否かを判断する。未処理の命令があれば、処理がステップS151に進められる。ハンドラ131bの処理が終了していれば、トレースデータ採取処理が終了する。

【0096】

以上のようにして、トレースデータを収集するハンドラ131b内で、トレースポイントとなっているAPIなどの関数を呼び出しても、処理のループを発生させずに済む。そのため、ハンドラ131b内でOSのAPIなどの各種の関数を利用することができる。その結果、自由度の高いトレースデータの収集処理が可能となる。例えばトレース用のハンドラ131b内でファイルアクセスのAPIを呼び出せるようになることで、メモリ102内に保存したトレースデータを、適宜、HDD103などの他の記録媒体に書き出すことが可能となる。その結果、収集可能なトレースデータのデータ量が、プール領域140の容量に依存しなくなり、大量のトレースデータの収集が可能となる。

40

【0097】

しかも第2の実施の形態では、IAT134を予め書き換えるだけで、ハンドラ131b内での関数40a-1の呼び出し命令実行時に、スタブ141aを実行させることができ、簡単な手順により例外命令42で上書きされた命令44を実行することができる。従って、トレース処理時の処理効率を低下させずに済む。

50

【0098】

さらに、トレースコードが記載されたハンドラを、プログラムヘフックすることで実行するため、同じトレースポイントを含む関数を実行するスレッドと同じスレッドでハンドラを実行できる。その結果、ハンドラ実行時にコンテキストスイッチが発生せず、レジスタやメモリの内容を、関数実行状態のまま維持できる。その結果、関数実行時の状態を正しくトレースできる。

【0099】

さらに、トレースポイントの命令を書き換える際に、メモリ上で書き換えを行うため、プログラムの実行ファイルには手を加えずに済む。そのためプログラムの再コンパイルをせずにトレース処理を行うことができる。

10

【0100】

またトレース対象の関数として、例えばOSで用意されたAPIのようなカーネルで動作する関数を指定できるため、カーネルで動作する関数についても、実行状態をトレースすることができる。

【0101】

〔第3の実施の形態〕

次に第3の実施の形態について説明する。第3の実施の形態では、APIなどの関数中の任意の箇所をトレースポイントとすることを可能とするものである。以下、第3の実施の形態における、第2の実施の形態と相違する部分について説明する。

【0102】

20

図17は、先頭以外の箇所をトレースポイントとした場合のループ発生例を示す図である。図17の例では、図5に示した関数40-1の先頭から2番目の命令をトレースポイントとしている。この場合、関数40-1の2行目の命令が、例外命令42に書き換えられる。命令書き換え後のプログラム40b内の関数40b-1が実行されると、例外命令42によりハンドラ131bが呼び出される。ハンドラ131b内には、関数40b-1を呼び出す命令51が含まれている。この場合、命令51が実行されると関数40b-1が呼び出され、再度、例外命令42が実行される。その結果、処理のループが発生する。

【0103】

なお、ハンドラ131bはプログラム40bにフックする形式で実行されており、図17に示すような処理は、1つのスレッドで実行される。すなわち、関数40b-1が繰り返し呼び出され、例外命令42が何度も実行されても、その例外命令42を実行するスレッドのスレッドID(TID)は同じである。そこで第3の実施の形態では、ドライバ部130は、例外処理のハンドラ131bを呼び出したスレッドのTIDに基づいて、同じスレッドからの呼び出し回数をカウントする。そしてドライバ部130は、特定のスレッドからのハンドラの呼び出し回数に応じた処理を行うことで、ループの発生を抑止する。

30

【0104】

図18は、第3の実施の形態におけるハンドラからのトレース対象の関数の呼び出し例を示す図である。関数40b-1の実行が開始され、実行箇所が例外命令42に達すると、例外が発生し、ハンドラ131cが呼び出される。

【0105】

40

ハンドラ131cの先頭にはTIDのチェック命令71が記述されている。TIDのチェック命令71の実行により、ハンドラ131cを呼び出したスレッド(プログラム40bを実行しているスレッド)のTIDが取得される。そして取得したTIDがメモリ102に記憶される。さらに、以前に同じTIDを取得しているかどうか判定される。

【0106】

1度目の呼び出しでは、同じTIDを以前に取得していないと判定される。この場合、ハンドラ131c内のTIDのチェック命令71以降の命令が実行される。そして関数40b-1を呼び出す命令51の実行により、関数40b-1が呼び出される。

【0107】

関数40b-1が呼び出されると、関数40b-1が先頭から実行される。そして例外

50

命令 4 2 に達すると、再度、例外が発生し、ハンドラ 1 3 1 c が呼び出される。ハンドラ 1 3 1 c が呼び出されると、まず T I D のチェック命令 7 1 が実行される。T I D のチェック命令 7 1 の実行により、呼出元のスレッドの T I D に基づいて、関数 4 0 b - 1 からの 2 度目の呼び出しであると判断される。2 度目の呼び出しの場合、ハンドラ 1 3 1 c に示されたデータ採取処理に代えて、スタブ 1 4 1 b が呼び出される。

【0 1 0 8】

スタブ 1 4 1 b には、例外命令 4 2 により書き換えられた命令 7 2 と、例外命令 4 2 の次の命令へのジャンプ命令 7 3 とが記述されている。従って、スタブ 1 4 1 b が実行されることにより、命令 7 2 が実行された後、関数 4 0 b - 1 の例外命令 4 2 の次の命令に処理がジャンプする。そして、関数 4 0 b - 1 のジャンプ先の命令以降の処理が実行される。

10

【0 1 0 9】

関数 4 0 b - 1 の処理が最後まで終了すると、関数 4 0 b - 1 の呼び出し元（ハンドラ 1 3 1 c の命令 5 1）に処理が戻される。そしてハンドラ 1 3 1 c 内の命令 5 1 以降の処理が実行され、データ採取が行われる。ハンドラ 1 3 1 c の処理が終了すると、スタブ 1 4 1 b 内の命令 7 2 が実行される。その後、ジャンプ命令 7 3 により、関数 4 0 b - 1 の例外命令 4 2 の次の命令に処理がジャンプする。そして、関数 4 0 b - 1 のジャンプ先の命令以降の処理が実行される。

【0 1 1 0】

このように、T I D を管理することで、同一スレッドからのハンドラ 1 3 1 c の呼び出し回数が判別可能となる。そして、2 度目の呼び出しではハンドラ 1 3 1 c 内のデータ採取処理を省略することで、ループの発生が抑止される。

20

【0 1 1 1】

なお、第 3 の実施の形態では、トレース設定ファイルにおいて、関数 4 0 b - 1 の先頭以外の部分を、トレースポイントとして指定できる。

図 1 9 は、第 3 の実施の形態におけるトレース設定ファイルの一例を示す図である。トレース設定ファイル 3 2 a では、トレース対象の関数の名称と、その関数中でのトレースポイントの先頭からのオフセット（シンボル名 + オフセット）により、トレースポイントを指定可能となっている。この場合、トレース対象の関数中の、先頭からオフセットで指定された位置が、トレースポイントとなる。

30

【0 1 1 2】

次に、トレース設定処理について説明する。

図 2 0 は、第 3 の実施の形態におけるトレース設定処理の手順の一例を示すフローチャートである。第 3 の実施の形態では、トレース管理テーブル作成処理（ステップ S 2 0 1）、例外ハンドラの置き換え処理（ステップ S 2 0 2）、およびトレースポイントの命令の例外命令への置き換え処理（ステップ S 2 0 3）が、ドライバ部 1 3 0 によって行われる。ステップ S 2 0 1 ~ S 2 0 3 の各処理の詳細は、それぞれ、図 9 に示した第 2 の実施の形態におけるトレース設定処理のステップ S 1 0 1, S 1 0 3, S 1 0 4 と同様である。すなわち、第 3 の実施の形態におけるトレース設定処理は、第 2 の実施の形態のトレース設定処理から、I A T の書き換え処理（ステップ S 1 0 2）を除いた処理となる。

40

【0 1 1 3】

次に、例外処理発生時に実行されるトラップ例外処理の手順について説明する。

図 2 1 は、第 3 の実施の形態におけるトラップ例外処理の手順の一例を示すフローチャートである。

【0 1 1 4】

〔ステップ S 2 1 1〕ドライバ部 1 3 0 は、トレースアドレスで発生した例外か否かを判断する。トレースアドレスで発生した例外の場合、処理がステップ S 2 1 3 に進められる。トレースアドレス以外で発生した例外の場合、処理がステップ S 2 1 2 に進められる。

【0 1 1 5】

50

〔ステップS 2 1 2〕ドライバ部1 3 0は、トレースアドレス以外で発生した例外であれば、ステップS 2 0 2（図2 0参照）による置き換え前の例外ハンドラを呼び出す。すると、呼び出された例外ハンドラが実行され、トレース処理を伴わない例外処理が行われる。その後、トラップ例外処理が終了する。

【0 1 1 6】

〔ステップS 2 1 3〕ドライバ部1 3 0は、スレッド判別処理を行う。スレッド判別処理では、例外発生元のスレッドが、ハンドラ1 3 1 cの実行により再帰的に呼び出されたスレッドか否かの判定が行われる。スレッド判別処理の詳細は後述する（図2 3参照）。

【0 1 1 7】

なおドライバ部1 3 0は、スレッド判別処理の過程で、T I D管理テーブルを作成する。T I D管理テーブルは、例外を発生させたスレッドのT I Dを管理するデータテーブルである。

【0 1 1 8】

図2 2は、T I D管理テーブルの一例を示す図である。図2 2の例では、T I D管理テーブル6 3には、トレースアドレスとT I Dとの欄が設けられている。トレースアドレスの欄には、トレースアドレスが設定される。T I Dの欄には、対応するトレースアドレスで例外を発生させた1以上のスレッドのT I Dが設定される。例えば、異なるスレッドそれぞれにおいて、トレース対象のA P Iが個別に呼び出されることがある。このような場合には、1つのトレースアドレスに、複数のT I Dが対応付けられる。

【0 1 1 9】

以下、図2 1の説明に戻る。

〔ステップS 2 1 4〕ドライバ部1 3 0は、ステップS 2 1 3により、再帰的に呼ばれたスレッドであると判定されたか否かを判断する。再帰的に呼ばれたスレッドであると判定された場合、処理がステップS 2 1 7に進められる。再帰的に呼ばれたスレッドではないと判定された場合、処理がステップS 2 1 5に進められる。

【0 1 2 0】

〔ステップS 2 1 5〕ドライバ部1 3 0は、再帰的に呼ばれたスレッドでなければ、トレースデータ採取処理を行う。トレースデータ採取処理の詳細は後述する（図2 4参照）。

【0 1 2 1】

〔ステップS 2 1 6〕ドライバ部1 3 0は、T I D管理テーブル内の、例外発生元のトレースアドレスに対応付けられているT I Dのリストの中から、例外発生元となっているトレース対象の関数を実行しているスレッドのT I Dを削除する。

【0 1 2 2】

〔ステップS 2 1 7〕ドライバ部1 3 0は、トレース対象の関数に対応するスタブを実行する。

〔ステップS 2 1 8〕ドライバ部1 3 0は、例外発生元に処理を戻す。これにより、例外発生元の関数を実行しているスレッドにより、例外発生アドレスの次の命令以降の処理が実行される。

【0 1 2 3】

次に、スレッド判別処理について詳細に説明する。

図2 3は、スレッド判別処理の手順の一例を示すフローチャートである。

〔ステップS 2 2 1〕ドライバ部1 3 0は、T I D管理テーブル6 3から、例外発生元のトレースアドレスを検索する。

【0 1 2 4】

〔ステップS 2 2 2〕ドライバ部1 3 0は、例外発生元のトレースアドレスがT I D管理テーブル6 3に登録されているか否かを判断する。トレースアドレスが登録されている場合、処理がステップS 2 2 4に進められる。トレースアドレスが登録されていない場合、処理がステップS 2 2 3に進められる。

【0 1 2 5】

〔ステップS 2 2 3〕ドライバ部1 3 0は、T I D管理テーブル6 3へ、例外発生元のトレースアドレスを登録する。その後、処理がステップS 2 2 6に進められる。

〔ステップS 2 2 4〕ドライバ部1 3 0は、T I D管理テーブル6 3に、例外発生元のトレースアドレスに対応付けて、今回の例外を発生させたスレッドのT I Dが登録されているか否かを判断する。該当するT I Dが登録されていれば、処理がステップS 2 2 5に進められる。該当するT I Dが登録されていなければ、処理がステップS 2 2 6に進められる。

【0 1 2 6】

〔ステップS 2 2 5〕ドライバ部1 3 0は、例外を発生させたスレッドのT I Dが、例外発生元のトレースアドレスに対応付けて登録されていれば、例外を発生させたスレッドは、再帰的に呼び出されたスレッドであると判別する。その後、スレッド判別処理が終了する。

10

【0 1 2 7】

〔ステップS 2 2 6〕ドライバ部1 3 0は、例外を発生させたスレッドのT I Dが、例外発生元のトレースアドレスに対応付けて登録されていない場合、T I D管理テーブル6 3へ、例外発生元のトレースアドレスに対応付けて、例外を発生させたスレッドのT I Dを登録する。

【0 1 2 8】

〔ステップS 2 2 7〕ドライバ部1 3 0は、例外を発生させたスレッドは、再帰的に呼び出されたスレッドではないと判別する。その後、スレッド判別処理が終了する。

20

次に、第3の実施の形態におけるトレースデータ採取処理の手順について説明する。

【0 1 2 9】

図2 4は、第3の実施の形態におけるトレースデータ採取処理の手順の一例を示すフローチャートである。

〔ステップS 2 3 1〕ドライバ部1 3 0は、ハンドラ1 3 1 cを呼び出し、ハンドラ1 3 1 b内の命令を次に実行する命令を解釈する。

【0 1 3 0】

〔ステップS 2 3 2〕ドライバ部1 3 0は、実行対象の命令が、例外発生元の関数（A P I）の呼び出し命令か否かを判断する。該当関数の呼び出し命令の場合、処理がステップS 2 3 4に進められる。該当関数の呼び出し命令でなければ、処理がステップS 2 3 3に進められる。

30

【0 1 3 1】

〔ステップS 2 3 3〕ドライバ部1 3 0は、ハンドラ1 3 1 cの次に実行する命令が、例外発生元のA P Iの呼び出し命令でなければ、その命令を実行する。その後、処理がステップS 2 4 1に進められる。

【0 1 3 2】

〔ステップS 2 3 4〕ドライバ部1 3 0は、ハンドラ1 3 1 cの次に実行する命令が、例外発生元のA P Iの呼び出し命令であれば、呼び出し命令に従って、例外発生元の関数を再帰的に呼び出す。

【0 1 3 3】

40

〔ステップS 2 3 5〕ドライバ部1 3 0は、呼び出した関数内の命令を先頭から順に解釈する。

〔ステップS 2 3 6〕ドライバ部1 3 0は、例外発生元の関数内の次に実行する命令が、例外命令か否かを判断する。例外命令であれば、処理がステップS 2 3 7に進められる。例外命令でなければ、処理がステップS 2 3 8に進められる。

【0 1 3 4】

〔ステップS 2 3 7〕ドライバ部1 3 0は、実行する処理が例外命令であれば、例外を発生させる。これにより、図2 1に示したトラップ例外処理が実行される。その後、処理がステップS 2 3 9に進められる。

【0 1 3 5】

50

〔ステップS 2 3 8〕ドライバ部1 3 0は、次に実行する命令が例外命令でなければ、その命令を実行する。

〔ステップS 2 3 9〕ドライバ部1 3 0は、例外発生元の関数（A P I）の処理が終了したか否かを判断する。処理が終了した場合、処理がステップS 2 4 0に進められる。処理が終了していなければ、処理がステップS 2 3 5に進められ、例外発生元の関数中の次に実行する命令が解釈される。

【0 1 3 6】

〔ステップS 2 4 0〕ドライバ部1 3 0は、例外発生元の関数の処理が終了すると、処理をハンドラ1 3 1 cに戻し、例外発生元の関数の呼び出し命令の次の命令を、次の実行対象の命令とする。

【0 1 3 7】

〔ステップS 2 4 1〕ドライバ部1 3 0は、ハンドラ1 3 1 cの処理が終了したか否かを判断する。未処理の命令があれば、処理がステップS 2 3 1に進められる。ハンドラ1 3 1 cの処理が終了していれば、トレースデータ採取処理が終了する。

【0 1 3 8】

このようにして、トレース対象のA P Iなどの関数の先頭以外がトレースポイントであっても、データのトレース処理を記述したハンドラ内に、トレース対象の関数の呼び出し命令を含めることが可能となる。従って、柔軟性の高いトレースデータの採取処理が可能となる。

【0 1 3 9】

なお第3の実施の形態では、トレースポイントがトレース対象の関数の先頭以外の場合の例を示しているが、トレースポイントがトレース対象の関数の先頭であっても、同様の処理でデータのトレースが可能である。

【0 1 4 0】

〔第4の実施の形態〕

次に、第4の実施の形態について説明する。第4の実施の形態は、第2の実施の形態と第3の実施の形態とを組み合わせたものである。第4の実施の形態では、トレース対象のプログラムである複数の関数にトレースポイントをセットし、各関数内に設定するトレースポイントが1箇所である場合に、トレースポイントが関数の先頭か否かに応じて、トレースデータ採取処理を切り替える。例えば第4の実施の形態では、トレースポイントがトレース対象の関数の先頭であれば、図7に示した第2の実施の形態の処理によりトレースデータを採取する。また第4の実施の形態では、トレースポイントがトレース対象の関数の先頭以外であれば、図18に示した第3の実施の形態の処理によりトレースデータを採取する。

【0 1 4 1】

第4の実施の形態では、トレースポイントが、トレース対象の関数の先頭か否かを、トレース管理テーブルを用いて管理する。

図25は、第4の実施の形態におけるトレース管理テーブルの一例を示す図である。第4の実施の形態におけるトレース管理テーブル64には、トレースアドレス、元の命令、スタブアドレス、およびフラグの欄が設けられている。トレースアドレス、元の命令、およびスタブアドレスの欄には、図10に示した第2の実施の形態のトレース管理テーブル60における同名の欄と同種の情報が設定される。

【0 1 4 2】

フラグの欄には、トレースアドレスで示されるトレースポイントが、トレース対象の関数の先頭かを示すフラグが設定される。例えば、トレースポイントがトレース対象の関数の先頭であれば、フラグの欄に「TRUE」と設定される。またトレースポイントがトレース対象の関数の先頭でなければ、フラグの欄に「FALSE」と設定される。

【0 1 4 3】

第4の実施の形態におけるトレース設定処理は、図9に示した第2の実施の形態のトレース設定処理と同様の手順で行われる。ただしトレース管理テーブル作成処理内のトレース

10

20

30

40

50

ス設定ファイル読み込み処理（ステップS101）が、第2の実施の形態と異なる。またIATの書き換え処理（ステップS102）が適用されるのは、トレースポイントがトレース対象の関数（API）の先頭の場合である。なお第4の実施の形態では、図19に示した第3の実施の形態のトレース設定ファイル32aと同様に、トレースポイントを、シンボル名+オフセットで指定可能である。

【0144】

図26は、第4の実施の形態におけるトレース設定ファイル読み込み処理の手順の一例を示すフローチャートである。なお図26に示した処理のうち、ステップS301～S307、S312の処理は、それぞれ図12に示したステップS121～S128の処理と同様である。以下、第2の実施の形態と異なる処理（ステップS308～S311）につ

10

【0145】

〔ステップS308〕ドライバ部130は、トレース管理テーブル64に登録したトレースアドレスを、「API名+オフセット」の形式に変換する。API名は、例えばトレース対象の関数を示すシンボル名である。アドレスから「API名+オフセット」への変換は、例えばOSに用意されたAPIなどを利用して行うことができる。

【0146】

〔ステップS309〕ドライバ部130は、オフセットが0か否かを判断する。例えばトレースポイントがトレース対象の関数の先頭の場合、オフセットが0となる。オフセットが0であれば、処理がステップS311に進められる。オフセットが0でなければ、処

20

【0147】

〔ステップS310〕ドライバ部130は、オフセットが0でなければ、トレース管理テーブル64における、ステップS304で読み取ったトレースポイントに対応するフラグを、「FALSE」にセットする。すなわち、トレースポイントが、トレース対象の関数の先頭ではないことが、フラグに設定される。その後、処理がステップS312に進められる。

【0148】

〔ステップS311〕ドライバ部130は、オフセットが0であれば、トレース管理テーブル64における、ステップS304で読み取ったトレースポイントに対応するフラグを、「TRUE」にセットする。すなわち、トレースポイントが、トレース対象の関数の先頭であることが、フラグに設定される。その後、処理がステップS312に進められる。

30

【0149】

このようにして、トレースポイントが関数の先頭か否かを示すフラグを含むトレース管理テーブル64が作成される。そしてトラップ例外が発生すると、トレース管理テーブル64に基づいて、トレースポイントが関数の先頭か否かが判断される。そして判断結果に応じた手順でトレースデータの採取が行われる。以下、第4の実施の形態におけるトラップ例外処理について詳細に説明する。

【0150】

40

図27は、トラップ例外処理の手順の一例を示すフローチャートである。なお図27に示した処理のうち、ステップS321、S322、S326～S331の処理は、それぞれ図21に示したステップS211～S218の処理と同様である。以下図27に示す処理のうち、第2または第3の実施の形態と相違する処理（ステップS323～S325）について説明する。

【0151】

〔ステップS323〕ドライバ部130は、トレースアドレス判別処理を行う。トレースアドレス判別処理では、例外発生元のトレースアドレスが、トレース対象の関数の先頭か否か判別される。なおトレースアドレス判別処理の詳細は後述する（図28参照）。

【0152】

50

〔ステップS 3 2 4〕ドライバ部1 3 0は、トレースアドレスがトレース対象の関数の先頭か否かを判断する。トレースアドレスがトレース対象の関数の先頭であれば、処理がステップS 3 2 5に進められる。またトレースアドレスがトレース対象関数の先頭でなければ、処理がステップS 3 2 6に進められ、第3の実施の形態と同様の手順でトレースデータの採取が行われる。

【0 1 5 3】

〔ステップS 3 2 5〕ドライバ部1 3 0は、第2の実施の形態と同様の手順で、トレースデータを採取する。この処理の詳細は、図1 6に示した処理と同様である。その後、処理がステップS 3 3 0に進められる。

【0 1 5 4】

次に、トレースアドレス判別処理について詳細に説明する。

図2 8は、トレースアドレス判別処理の手順の一例を示すフローチャートである。

〔ステップS 3 4 1〕ドライバ部1 3 0は、トレース管理テーブル6 4から、現在実行している例外処理の発生元のトレースアドレスに対応するフラグを取得する。

【0 1 5 5】

〔ステップS 3 4 2〕ドライバ部1 3 0は、取得したフラグが「TRUE」か否かを判断する。フラグが「TRUE」であれば、処理がステップS 3 4 3に進められる。フラグが「FALSE」であれば、処理がステップS 3 4 4に進められる。

【0 1 5 6】

〔ステップS 3 4 3〕ドライバ部1 3 0は、フラグが「TRUE」の場合、トレースポイントがトレース対象の関数の先頭であると判定し、トレースアドレス判別処理を終了する。

【0 1 5 7】

〔ステップS 3 4 4〕ドライバ部1 3 0は、フラグが「FALSE」の場合、トレースポイントがトレース対象の関数の先頭ではないと判定し、トレースアドレス判別処理を終了する。

【0 1 5 8】

このようにして、トレースポイントがトレース対象の関数の先頭かどうかにより、データ採取の手法を変更することができる。ここで、トレースポイントがトレース対象の関数の先頭であれば、スレッドのT I Dを管理せずに済み、簡便な処理でデータを採取でき、負荷が少なく済む。従って、第4の実施の形態によれば、トレースポイントが関数の先頭でない場合でもデータの採取を可能としながらも、第3の実施の形態よりも少ない処理負荷でデータのトレースを行うことができる。

【0 1 5 9】

以上、実施の形態を例示したが、実施の形態で示した各部の構成は同様の機能を有する他のものに置換することができる。また、他の任意の構成物や工程が付加されてもよい。さらに、前述した実施の形態のうちの任意の2以上の構成（特徴）を組み合わせたものであってもよい。

【符号の説明】

【0 1 6 0】

- 1 プログラム
- 1 a 関数
- 2 トレースコード
- 3 スタブ
- 4 第1の命令
- 5 第2の命令
- 6 第3の命令
- 7 第4の命令
- 1 0 情報処理装置
- 1 1 書き換え手段

10

20

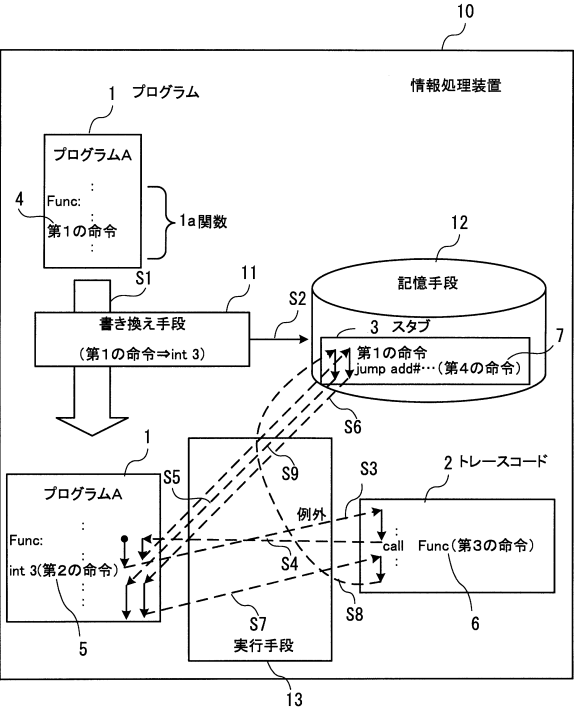
30

40

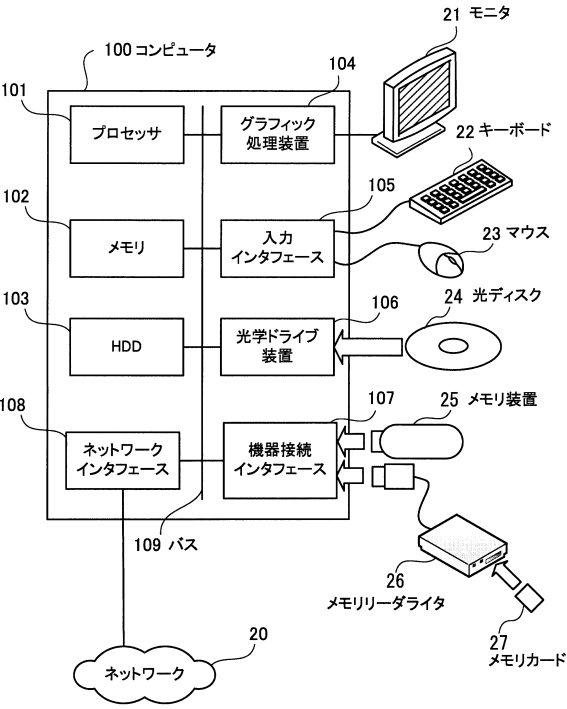
50

- 1 2 記憶手段
- 1 3 実行手段

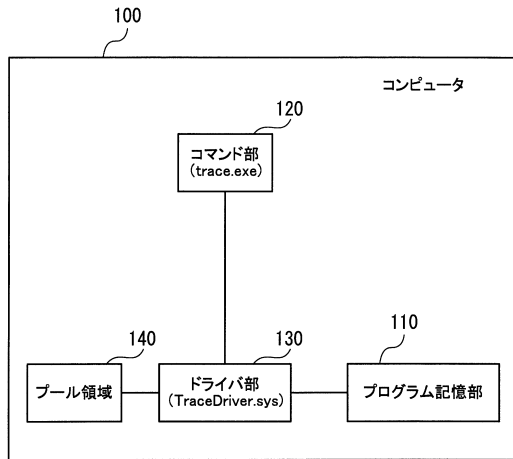
【図 1】



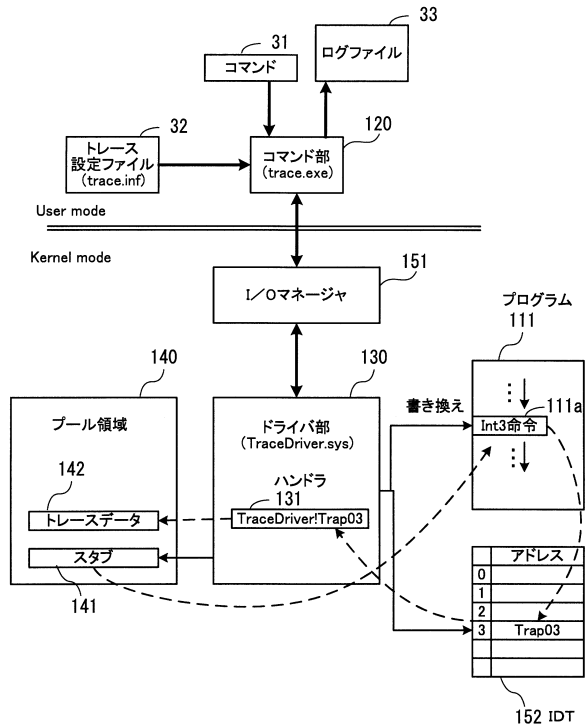
【図 2】



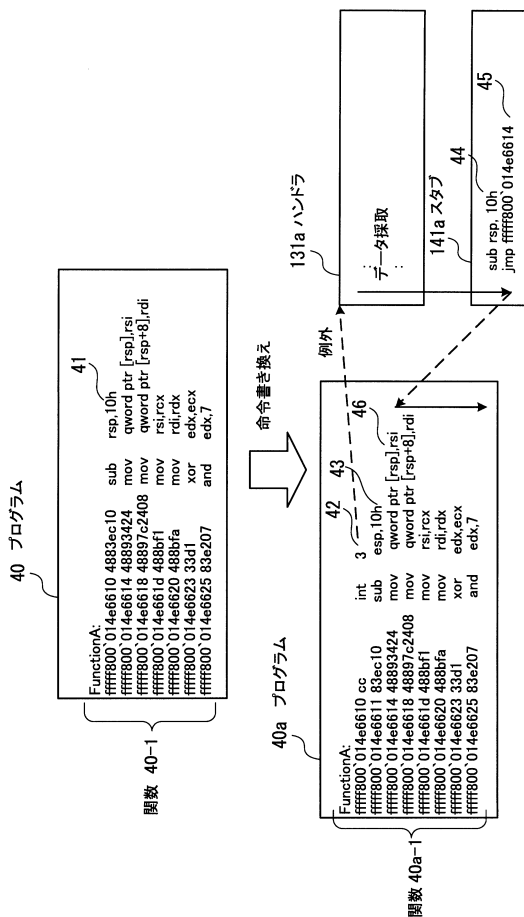
【図 3】



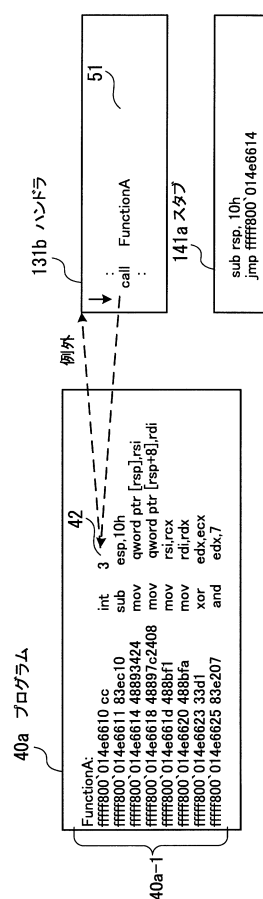
【図 4】



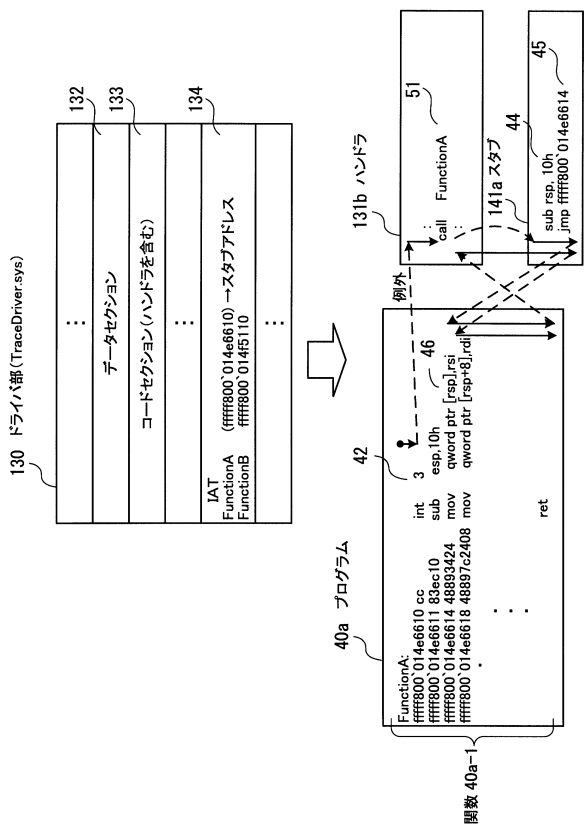
【図 5】



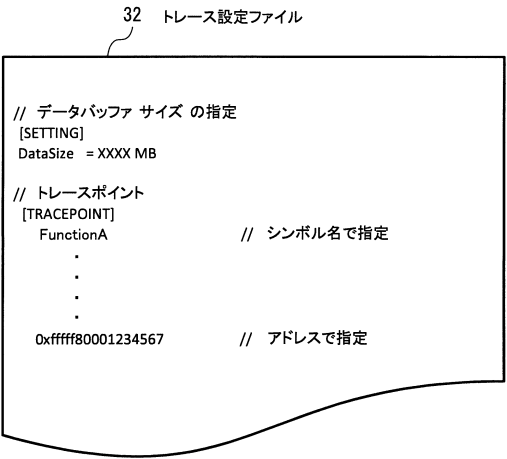
【図 6】



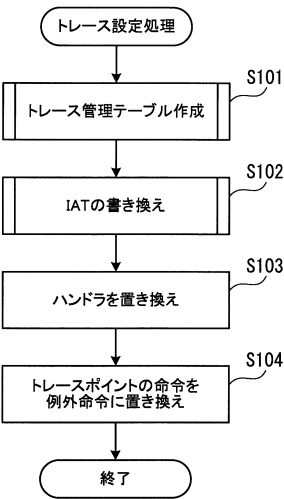
【図 7】



【図 8】



【図 9】

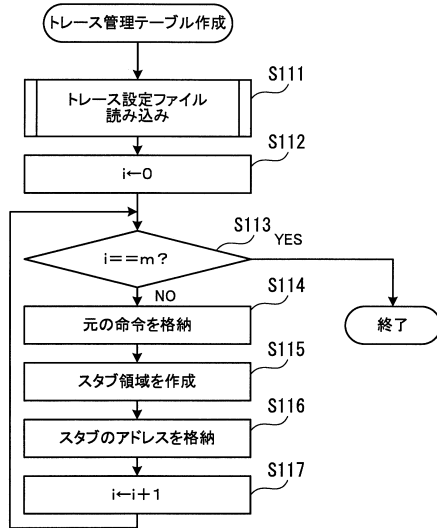


【図 10】

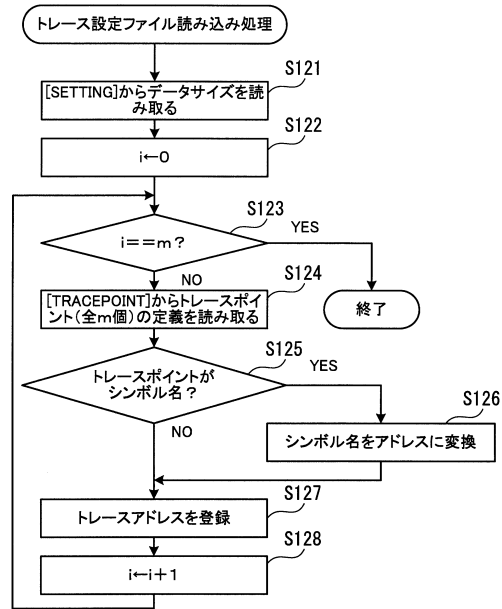
60 トレース管理テーブル

トレースアドレス	元の命令	スタブアドレス
0xAAAAAAAA	aaaa	stubA
0xBBBBBBBB	bbbb	stubB
0xCCCCCCCC	cccc	stubC
...	...	...
...	...	...
...	...	...
0xMMMMMMMM	mmmm	stubM

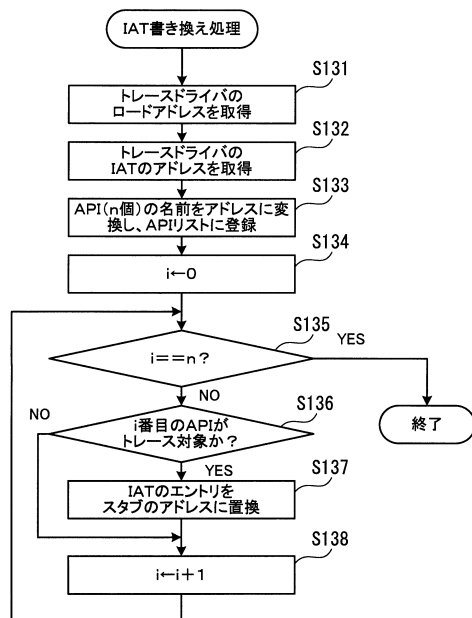
【図 1 1】



【図 1 2】



【図 1 3】

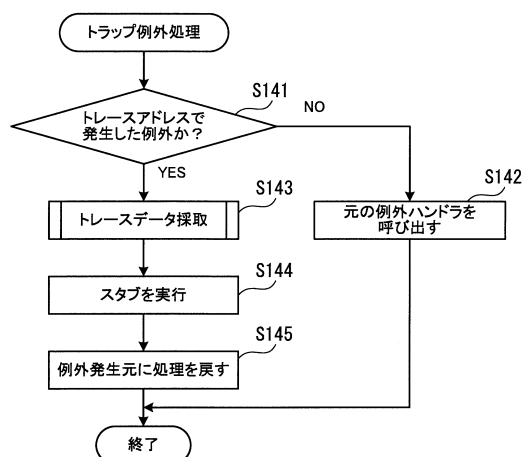


【図 1 4】

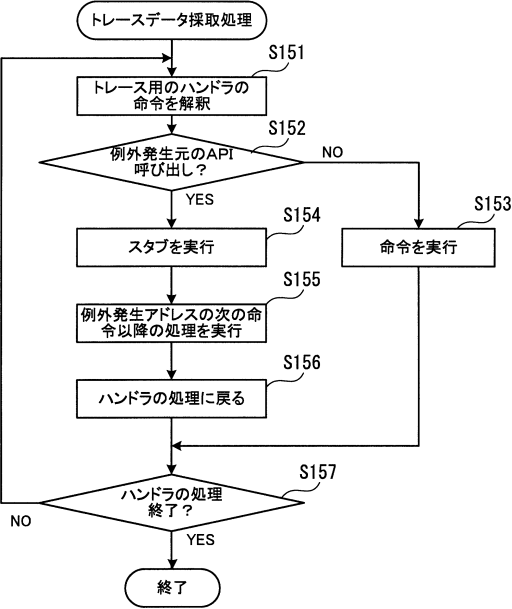
62 APIリスト

API名	アドレス
HandlerAPI_1	0xFFFFFFFF
HandlerAPI_2	0xFFFFFFFF
HandlerAPI_3	0xFFFFFFFF
...	...
...	...
HandlerAPI_N	0xFFFFFFFF

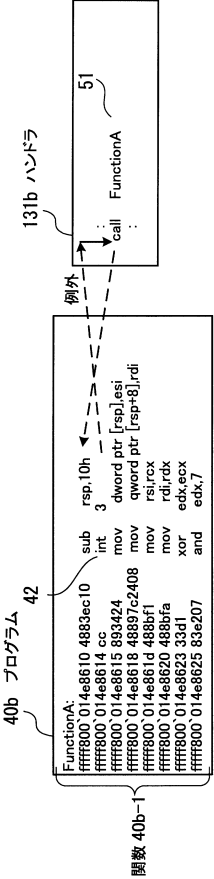
【図 1 5】



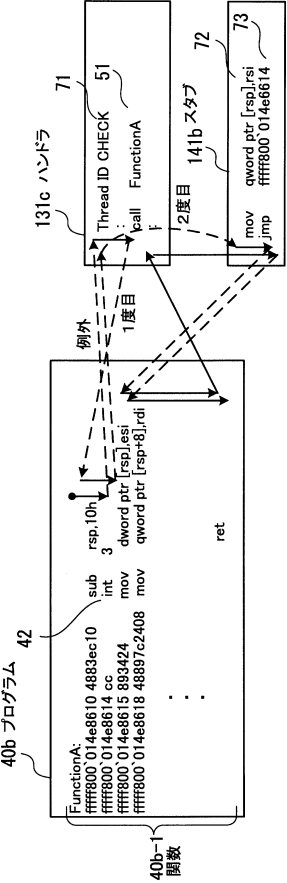
【図 16】



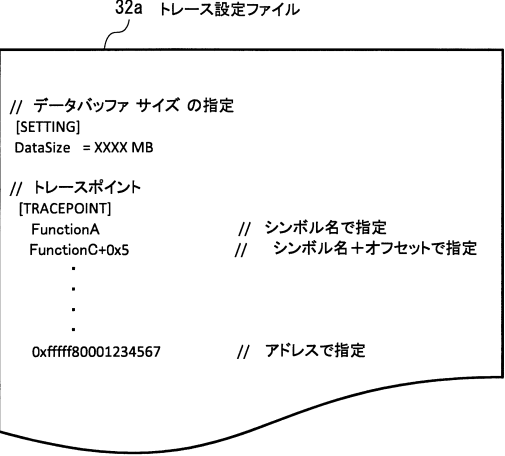
【図 17】



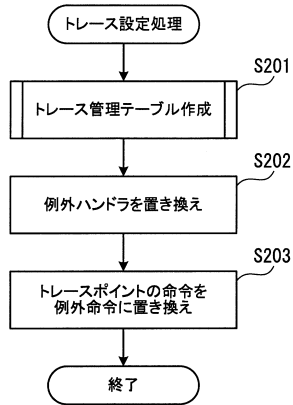
【図 18】



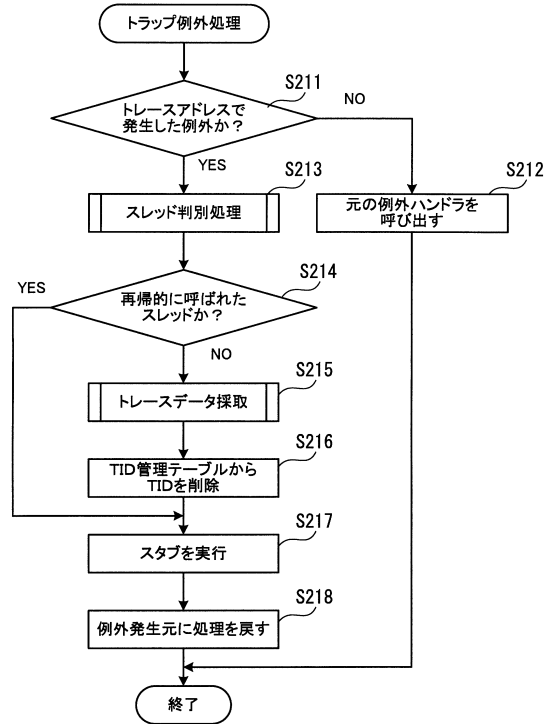
【図 19】



【図 20】



【図 21】

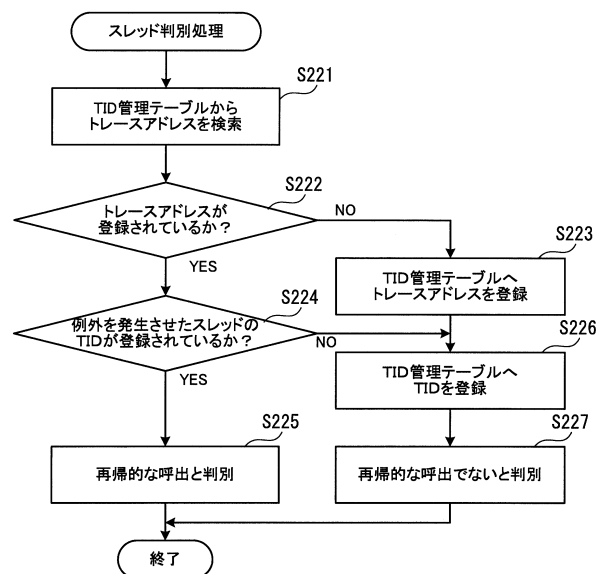


【図 22】

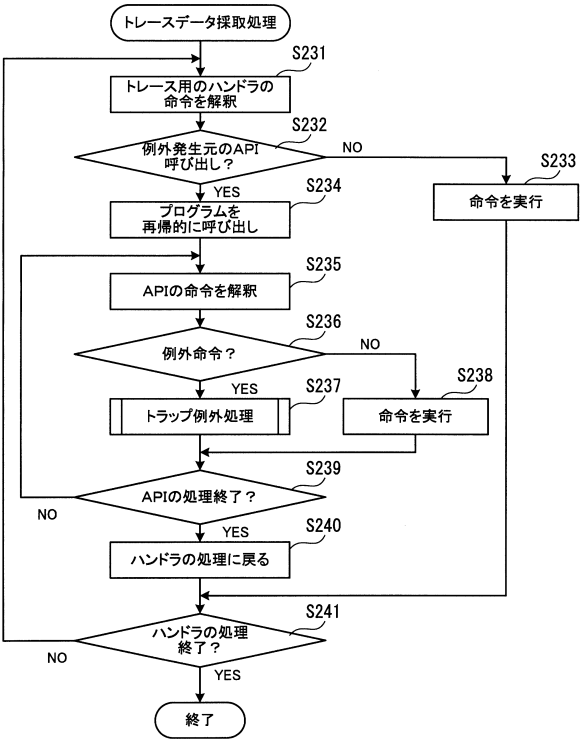
63 TID管理テーブル

トレースアドレス	TID
0xAAAAAAAA	aaa1, aaa2, aaa3...
0xBBBBBBBBB	bbb1, bbb2, bbb3...
0xCCCCCCCC	ccc1, ccc2, ccc3...
0xDDDDDDDD	ddd1, ddd2, ddd3...
...	...
...	...

【図 23】



【図 2 4】

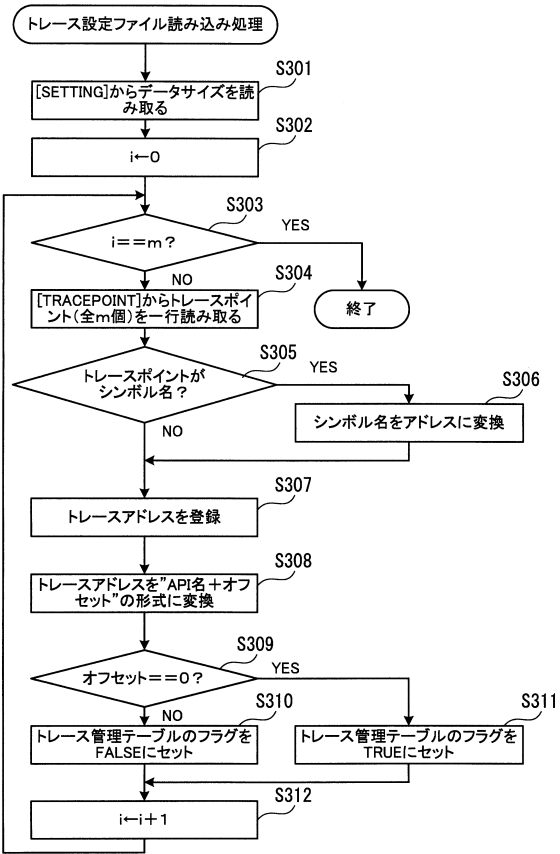


【図 2 5】

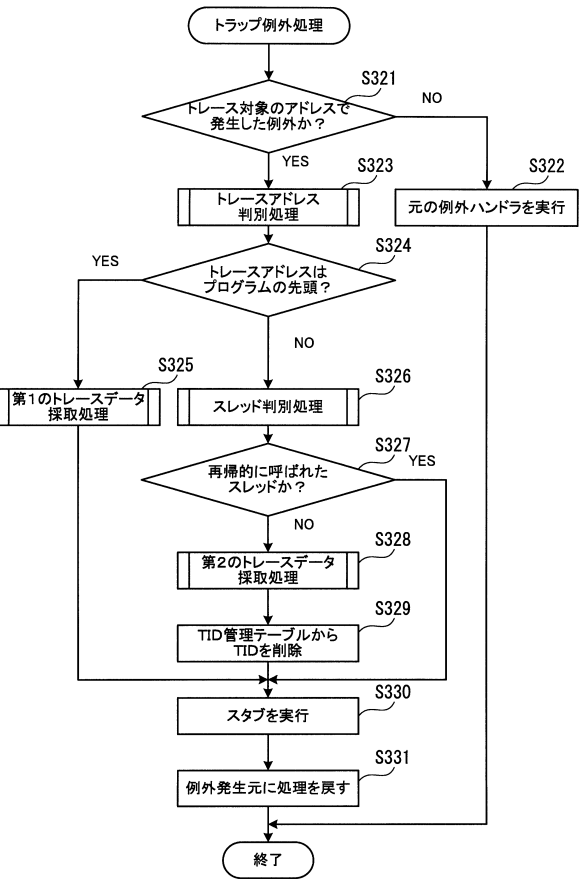
64 トレース管理テーブル

トレースアドレス	元の命令	スタブアドレス	フラグ
0xAAAAAAAA	aaaa	stabA	TRUE
0xBBBBBBBB	bbbb	stabB	FALSE
0xCccccccc	cccc	stabC	FALSE
0xDDDDDDDD	dddd	stabD	TRUE
...	...	...	...
...	...	...	...

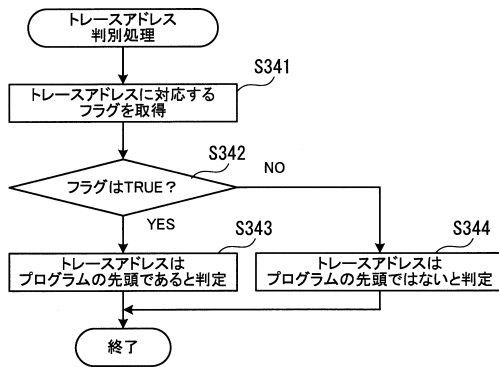
【図 2 6】



【図 2 7】



【図 28】



フロントページの続き

(72)発明者 今枝 一英

神奈川県川崎市中原区上小田中4丁目1番1号 富士通株式会社内

審査官 多胡 滋

(56)参考文献 特開2005-301382 (JP, A)

(58)調査した分野(Int.Cl., DB名)

G06F 11/36