(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5717678号
(P5717678)
(45)発行日　平成27年5月13日(2015.5.13)　(24)登録日　平成27年3月27日(2015.3.27)

(51)Int.Cl.　　F I
G06F 11/28 (2006.01)　　G06F 11/28 310B
G06F 11/34 (2006.01)　　G06F 11/34 S

請求項の数 6　(全 16 頁)

(21)出願番号　特願2012-51773(P2012-51773)
(22)出願日　平成24年3月8日(2012.3.8)
(65)公開番号　特開2013-186731(P2013-186731A)
(43)公開日　平成25年9月19日(2013.9.19)
審査請求日　平成26年4月16日(2014.4.16)

(73)特許権者　000006013
三菱電機株式会社
東京都千代田区丸の内二丁目７番３号
(74)代理人　100094916
弁理士　村上　啓吾
(74)代理人　100073759
弁理士　大岩　増雄
(74)代理人　100127672
弁理士　吉澤　憲治
(74)代理人　100088199
弁理士　竹中　岑生
(72)発明者　真鍋　恭
東京都千代田区丸の内二丁目７番３号　三菱電機株式会社内

最終頁に続く

(54)【発明の名称】コンピュータシステムおよびコンピュータシステムを用いたネットワークシステム

(57)【特許請求の範囲】
【請求項１】
コンピュータシステムにおいて、
ｆｔｒａｃｅが装備されているＬｉｎｕｘ（登録商標）カーネルを有し、
アプリケーションの共有ライブラリを有するトレースライブラリ部と、
上記トレースライブラリ部の関数に対して、その入り口と出口において、上記関数の引数あるいは戻り値を引数に含むとともにエラーとなるシステムコールを呼び出すことで、上記ｆｔｒａｃｅを用いて上記関数の引数および戻り値を含めたｆｔｒａｃｅ出力トレースログの出力を行うトレースコマンド機能部、および、上記ｆｔｒａｃｅ出力トレースログおよび上記アプリケーションの関数情報を結合して上記関数の入り口および出口を含めたアプリケーショントレースログを作成するトレース解析機能部を有するトレースプログラム部とを備えるコンピュータシステム。
【請求項２】
上記トレースコマンド機能部は、複数のプロセスを対象に上記アプリケーショントレースログを出力する請求項１記載のコンピュータシステム。
【請求項３】
上記トレースコマンド機能部は、上記アプリケーショントレースログの出力を自動開始するためのトレース条件を上記トレースライブラリ部に設定する請求項１または請求項２記載のコンピュータシステム。
【請求項４】

上記トレース解析機能部は、上記ｆｔｒａｃｅ出力トレースログのフィルタリングを行う請求項１ないし請求項３のいずれか１項に記載のコンピュータシステム。
【請求項５】
上記トレースプログラム部は、上記アプリケーショントレースログをグラフ化形状に変更する可視化機能部を備える請求項１ないし請求項４のいずれか１項に記載のコンピュータシステム。
【請求項６】
ネットワークに接続された複数の端末を有し、上記一の端末には請求項１ないし請求項５のいずれか１項に記載のコンピュータシステムを有し、
上記他の端末は、当該他の端末のアプリケーショントレースログの記録の開始および終了を指示するトレースクライアント機能部を有し、
上記一の端末の上記トレースプログラム部は、トレースサーバ機能部を有し、
上記トレースサーバ機能部は、上記トレースクライアント機能部からの指示により、上記ネットワークおよび上記トレースクライアント機能部を介して、上記他の端末の上記アプリケーショントレースログを作成するコンピュータシステムを用いたネットワークシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
この発明は、ｆｔｒａｃｅが装備されているＬｉｎｕｘ（登録商標）カーネルにおいて、通常の汎用アプリケーションの関数の入り口および出口のアプリケーショントレースログを柔軟に取得でき、より効率的な障害解析、アプリケーションの稼動状態の確認や障害原因の切り分けを可能とするコンピュータシステムおよびコンピュータシステムを用いたネットワークシステムに関するものである。
【背景技術】
【０００２】
ビジネスや社会生活において、各種のサービスを実現するコンピュータシステムへの期待は、ますます高まってきている。また、近年のいくつかの大規模システム障害の発生により、コンピュータシステムの品質が個人のみならず社会に大きな影響を与えることが強く認識されてきている。一方、近年のコンピュータシステムは、ますます複雑化の傾向にあり、その規模も大きくなるばかりである。そのため、効率的な障害解析（アプリケーションの稼動状態の確認や障害原因の切り分け）の方法が、より一層求められている。現在、アプリケーションの振る舞いを確認するための手段としては、デバッガとトレーサとの２つのツールが主に用いられている。
【０００３】
デバッガは、ＣＰＵを止めて特定プロセスの状態（メモリ、レジスタ、変数）を調べることができる。また、デバッガの一種で、関数ごとの実行頻度や実行時間などといったプロセスの振る舞いを測定することができるプロファイラと呼ばれるツールもある。ただ、これらのツールを利用するためには、アプリケーションのコンパイル／リンク時に－ｐｇオプションを付ける必要があり、リソース等が非常に限られた環境以外では、この－ｐｇオプションにてコンパイル／リンクが行われるのが通常である。一方、トレーサは、一般にＣＰＵを止めずに、システム全体で実行した命令とその結果を実行順に出力するツールであり、プロセスを実行させながら、プロセスの振る舞いを可視化することができることから、近年、脚光を浴びている。
【０００４】
Ｌｉｎｕｘ（登録商標）カーネルでは、Ｌｉｎｕｘ（登録商標）カーネル２．６．２７以降、ｆｔｒａｃｅと呼ばれるトレーサが標準装備されている。ｆｔｒａｃｅは、トレース時に実行するモジュールを動的に組み込む動的トレーサと異なり、カーネルコード中に直接トレース呼び出しが記述されている静的トレーサであるため、動的トレーサに比べて高速に動作するのが特徴である。ここで、ｆｔｒａｃｅの動作について説明する。Ｌｉｎ

ｕｘ（登録商標）カーネル上で動作するプロセスは、元となるアプリケーションと、Ｌｉｎｕｘ（登録商標）カーネルから提供されたランタイムライブラリとからなり、プロセス内からＬｉｎｕｘ（登録商標）カーネルに対して様々なシステムコールが呼び出される。一方、Ｌｉｎｕｘ（登録商標）カーネルでは、ｆｔｒａｃｅの起動／終了を制御することができ、ｆｔｒａｃｅが起動している場合には、システムコールＩ／Ｆによって、システムコールが呼び出される度に、そのシステムコール名と引数、戻り値をＬｉｎｕｘ（登録商標）カーネル上のメモリに格納し、ｆｔｒａｃｅが終了されるタイミングで、ｆｔｒａｃｅ出力トレースログとして、ファイルに保存される。

【０００５】

このようなｆｔｒａｃｅを用いることによって、Ｌｉｎｕｘ（登録商標）カーネル上のシステムコールの振る舞いを監視することができ、カーネルレベルの稼動状態の確認や障害原因の切り分けを行うのに、非常に効果的である。また、ｆｔｒａｃｅは、アーキテクチャ非依存であり、プラグインによって柔軟に拡張することが可能である。プラグインには、全てのカーネル関数の入り口と出口をトレースすることができるｆｕｎｃｔｉｏｎ＿ｇｒａｐｈ（引数や戻り値は取得できない）等がある。しかしながら、プラグインを含めたとしても、ｆｔｒａｃｅでは、汎用アプリケーションの関数レベルのトレースログを、引数や戻り値も含めて取得することができない。

【０００６】

このことを解決するために、カーネルやアプリケーションに専用の改良を加えることで、関数レベルのトレースログを出力する手法が提案されている（例えば、特許文献１および特許文献２参照）。

【先行技術文献】

【特許文献】

【０００７】

【特許文献１】特開２００１－３５６９０６号公報

【特許文献２】特開２００８－０５９４０２号公報

【発明の概要】

【発明が解決しようとする課題】

【０００８】

従来のＬｉｎｕｘ（登録商標）カーネルでは、装備されているｆｔｒａｃｅやそのプラグインを用いても、システムコールやカーネル関数レベルのトレースログしか取得することができず、汎用アプリケーションの関数レベルのトレースログを、引数や戻り値も含めて取得することができなかった。また、上記に示した各特許文献では、その課題の解決のために、カーネルやアプリケーションに専用の改良を加える必要があるという問題点があった。

【０００９】

この発明は上記のような課題を解決するためになされたものであり、装備されているｆｔｒａｃｅの仕組みをそのまま利用し、カーネルやアプリケーションに専用の改良を加えることなく、アプリケーションの関数レベルのアプリケーショントレースログを、引数や戻り値も含めて取得できるコンピュータシステムおよびコンピュータシステムを用いたネットワークシステムを提供することを目的とする。

【課題を解決するための手段】

【００１０】

この発明のコンピュータシステムは、
コンピュータシステムにおいて、
ｆｔｒａｃｅが装備されているＬｉｎｕｘ（登録商標）カーネルを有し、
アプリケーションの共有ライブラリを有するトレースライブラリ部と、
上記トレースライブラリ部の関数に対して、その入り口と出口において、上記関数の引数あるいは戻り値を引数に含むとともにエラーとなるシステムコールを呼び出すことで、上記ｆｔｒａｃｅを用いて上記関数の引数および戻り値を含めたｆｔｒａｃｅ出力トレース

ログの出力を行うトレースコマンド機能部、および、上記ｆｔｒａｃｅ出力トレースログおよび上記アプリケーションの関数情報を結合して上記関数の入り口および出口を含めたアプリケーショントレースログを作成するトレース解析機能部を有するトレースプログラム部とを備える
ものである。

【００１１】

また、この発明のコンピュータシステムを用いたネットワークシステムは、
ネットワークに接続された複数の端末を有し、上記一の端末には請求項１ないし請求項５のいずれか１項に記載のコンピュータシステムを有し、
上記他の端末は、当該他の端末のアプリケーショントレースログの記録の開始および終了を指示するトレースクライアント機能部を有し、
上記一の端末のトレースプログラム部は、トレースサーバ機能部を有し、
上記トレースサーバ機能部は、上記トレースクライアント機能部からの指示により、上記ネットワークおよび上記トレースクライアント機能部を介して、上記他の端末の上記アプリケーショントレースログを作成する
ものである。

【発明の効果】

【００１２】

この発明のコンピュータシステムによれば、
アプリケーションの関数の入り口および出口のアプリケーショントレースログを、引数や戻り値も含めて取得できる。

【００１３】

また、この発明のコンピュータシステムを用いたネットワークシステムは、
ネットワークに接続された複数の端末を有し、上記一の端末には請求項１ないし請求項５のいずれか１項に記載のコンピュータシステムを有し、
上記他の端末は、当該他の端末のアプリケーショントレースログの記録の開始および終了を指示するトレースクライアント機能部を有し、
上記一の端末のトレースプログラム部は、トレースサーバ機能部を有し、
上記トレースサーバ機能部は、上記トレースクライアント機能部からの指示により、上記ネットワークおよび上記トレースクライアント機能部を介して、上記他の端末の上記アプリケーショントレースログを作成するので、
ネットワークに接続された一の端末から他の端末の、アプリケーションの関数の入り口および出口のアプリケーショントレースログを、引数や戻り値も含めて取得できる。

【図面の簡単な説明】

【００１４】

【図１】この発明の実施の形態１のコンピュータシステムの構成を示す図である。
【図２】図１に示したコンピュータシステムにおけるｆｔｒａｃｅ出力トレースログの一例を示した図である。
【図３】図１に示したコンピュータシステムにおけるアプリケーショントレースログの一例を示した図である。
【図４】図１に示したコンピュータシステムにおけるトレースライブラリ部のｍｏｕｎｔ（）関数の手順を示すフローチャートである。
【図５】図１に示したコンピュータシステムにおけるトレースライブラリ部のｒｅｔｕｒｎ＿ｆｕｎｃｔｉｏｎ（）関数の手順を示すフローチャートである。
【図６】図１に示したコンピュータシステムにおけるトレースプログラム部のトレースコマンド機能部の手順を示すフローチャートである。
【図７】図１に示したコンピュータシステムにおけるトレースプログラム部のトレース解析機能部の手順を示すフローチャートである。
【図８】この発明の実施の形態２におけるコンピュータシステムの構成を示す図である。
【図９】図８に示したトレースプログラム部のトレースコマンド機能部の手順を示すフロ

ーチャートである。
【図１０】この発明の実施の形態３におけるコンピュータシステムの構成を示した図である。
【図１１】図１０に示したコンピュータシステムにおけるトレースライブラリ部のｍｏｕｎｔ（）関数の手順を示すフローチャートである。
【図１２】図１０に示したコンピュータシステムにおけるトレースプログラム部のトレースコマンド機能部の手順を示すフローチャートである。
【図１３】この発明の実施の形態４におけるコンピュータシステムの構成を示す図である。
【図１４】この発明の実施の形態５におけるコンピュータシステムの構成を示す図である。
【図１５】図１４に示したコンピュータシステムにおけるアプリケーショントレースグラフ（状態遷移図）の一例を示した図である。
【図１６】図１４に示したコンピュータシステムにおけるアプリケーショントレースグラフ（シーケンス図）の一例を示した図である。
【図１７】この発明の実施の形態６におけるコンピュータシステムを用いたネットワークシステムの構成を示した図である。
【発明を実施するための形態】
【００１５】
実施の形態１．
以下、本願発明の実施の形態について説明する。図１はこの発明の実施の形態１におけるコンピュータシステムの構成を示す図、図２は図１に示したコンピュータシステムにおけるｆｔｒａｃｅ出力トレースログの具体的な一例を示した図、図３は図１に示したコンピュータシステムにおけるアプリケーショントレースログの具体的な一例を示した図、図４は図１に示したコンピュータシステムにおけるｍｃｏｕｎｔ（）関数のトレースログの出力手順を示したフローチャート、図５は図１に示したコンピュータシステムにおけるｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数のトレースログの出力手順を示したフローチャート、図６は図１に示したコンピュータシステムにおけるトレースコマンド機能部の手順を示したフローチャート、図７は図１に示したコンピュータシステムにおけるトレース解析機能部の手順を示したフローチャートである。
【００１６】
図において、－ｐｇオプション付きのアプリケーション１０では、プロファイラ等に使用されるプロファイル情報を保存するために、各関数の入り口で、ｍｃｏｕｎｔ（）というｍｃｏｕｎｔ（）関数５を呼び出す。この発明に係わるアプリケーショントレース機能では、このｍｃｏｕｎｔ（）関数５を、アプリケーショントレース用に置き換えるため、Ｌｉｎｕｘ（登録商標）カーネル１のＬＤ＿ＰＲＥＬＯＡＤ６と呼ばれる機能を利用する。Ｌｉｎｕｘ（登録商標）カーネル１では、同じ名前の関数を持つ複数のライブラリがダイナミックリンクされている場合、最初に見つかった関数が使用される性質がある。そのため、環境変数として、あらかじめＬＤ＿ＰＲＥＬＯＡＤ６で共有ライブラリ作成し、指定しておくことで、システム標準のランタイムライブラリ４の関数を、別の関数に置き換えることができる。
【００１７】
よって、アプリケーショントレース機能では、環境変数をＬＤ＿ＰＲＥＬＯＡＤ６として、トレースライブラリ部１１を指定しておくことで、Ｌｉｎｕｘ（登録商標）カーネル提供のランタイムライブラリ４のｍｃｏｕｎｔ（）関数５の代わりに、トレースライブラリ部１１のｍｃｏｕｎｔ（）関数１４およびｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５が実行されるようにする。また、トレースプログラム部１２は、トレースコマンド機能部１７とトレース解析機能部１８とを備えた単独のアプリケーションとして提供される。トレースプログラム部１２のトレースコマンド機能部１７は、アプリケーショントレース記録の開始／終了を行う。アプリケーショントレース記録を開始する場合、トレースコマン

ド機能部１７は、対象となるアプリケーションの起動とともにアプリケーショントレース記録を開始するか、すでに起動しているアプリケーションのプロセスに対して、アプリケーショントレース記録を開始するように要求する。

【００１８】

また、アプリケーショントレース記録を終了する場合、トレースコマンド機能部１７は、対象となっている、すでにアプリケーショントレース記録中のプロセスに対して、アプリケーショントレース記録を終了するように要求する。もちろん、アプリケーションのプロセスが終了した際に、自動的にアプリケーショントレース記録を終了することも可能である。こうして、アプリケーショントレース記録を終了した際に、ｆｔｒａｃｅ出力トレースログ９をファイルに保存する。一方、トレース解析機能部１８は、アプリケーショントレース記録が終了した際に呼び出される。トレース解析機能部１８は、ｆｔｒａｃｅ８が出力するｆｔｒａｃｅ出力トレースログ９と、アプリケーション１０の関数情報を結合して、最終的な関数の入り口および出口（関数レベル）を有するアプリケーショントレースログ２４を出力する。例えば、図２に示すようなｆｔｒａｃｅ出力トレースログ９の場合、アプリケーショントレースログ２４は、図３のようになる。

【００１９】

次に上記のように構成された実施の形態１のコンピュータシステムの動作について説明する。まず、ｍｃｏｕｎｔ（）関数１４の入り口を示すトレースログの出力手順を図４に基づき説明する。トレースライブラリ部１１のｍｃｏｕｎｔ（）関数１４が呼び出されると、最初に、トレースライブラリ部１１内のトレースフラグ１３がＯＮになっているか否かを確認する（図４のステップＳ１０００）。そして、ＯＦＦになっている場合（Ｎｏ）には、アプリケーショントレースログの出力を行わないまま、ここで処理を終了する。また、ＯＮになっている場合（Ｙｅｓ）には、ｍｃｏｕｎｔ（）関数１４を呼び出した元の関数のスタック情報を調べ、呼び出し元関数が終了した際のリターンアドレスを、別メモリに記憶する（図４のステップＳ１０１０）。次に、上記リターンアドレスを、トレースライブラリ部１１のｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５の開始アドレスに置き換える（図４のステップＳ１０２０）。

【００２０】

よって、呼び出し元関数が終了した際に、必ずトレースライブラリ部１１のｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５が呼び出されるようになる。次に、同じくスタック情報から、呼び出し元関数の各引数の値を取得する（図４のステップＳ１０３０）。最後に、ｆｔｒａｃｅ８に関数の入り口のトレースログを残すため、あえて明示的にエラーとなるｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７をシステムコールＩ／Ｆ２が呼び出す（図４のステップＳ１０４０）。ここでは、なるべくエラーの処理が短く、システムコールの引数が多いｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７を用いる（尚、ｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７は、６個の引数を持っているものである）。そして、ｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７の引数（６個）として、必ずシステムコールがエラーとなり、後に、ここで埋め込んだことがわかるようなエラー値と、呼び出し元関数のアドレス位置と、呼び出し元関数の引数を最大４個までとを指定して、ｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７を実行する。以上で、トレースライブラリ部１１のｍｃｏｕｎｔ（）関数１４の処理は終了である。

【００２１】

次に、ｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５の出口を示すトレースログの出力手順を図５に基づいて説明する。まず、トレースライブラリ部１１のｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５が呼び出されると、最初に、ｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５を呼び出した元の関数のスタック情報を調べ、呼び出し元関数の戻り値を取得する（図５のステップＳ１１００）。次に、ｆｔｒａｃｅ８に関数出口のトレースログを残すため、あえて明示的にエラーとなるｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７をシステムコールＩ／Ｆ２が呼び出す（図５のステップＳ１１１０）。ここでは、上記において説明した場合と同様に、ｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７を用いる。

【００２２】
そして、ｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７の引数として、必ずシステムコールがエラーとなり、後に、ここで埋め込んだことがわかるようなエラー値と、呼び出し元関数のアドレス位置と、呼び出し元関数の戻り値とを指定して、ｓｙｓ＿ｓｐｌｉｃｅ（）システムコール７を実行する。最後に、先に別メモリに保存しておいた、呼び出し元関数の本来のリターンアドレスを呼び出し、ｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５のリターンアドレスの代わりに置き換える（図５のステップＳ１１２０）。このようにすれば、ｒｅｔｕｒｎ＿ｆｕｃｔｉｏｎ（）関数１５が終了した際に、呼び出し元関数を呼び出した本来の関数に処理が戻ることとなる。
【００２３】
次に、トレースプログラム１２のトレースコマンド機能部１７の手順を図６に基づいて説明する。まず、アプリケーショントレースログ記録を開始する場合は、まず、ｆｔｒａｃｅ８の開始を行う（図６のステップＳ１２００）。次に、アプリケーショントレースログを記録する対象がすでに起動済みのアプリケーションか否かを判断する（図６のステップＳ１２１０）。そして、起動済みのアプリケーションでない場合（Ｎｏ）には、対象アプリケーションのプロセスを新しく起動する（図６のステップＳ１２２０）。また、起動済みのアプリケーションである場合（Ｙｅｓ）、および、ステップＳ１２２０にてアプリケーションを起動した後に、対象プロセスにアタッチされているトレースライブラリ部１１のトレースフラグ１３をＯＮに設定する（図６のステップＳ１２３０）。このことで、アプリケーショントレースログ出力が開始となる。
【００２４】
一方、アプリケーショントレースログ出力を終了する場合は、まず、アプリケーショントレースログを記録する対象のプロセスが終了しているか否かを判断する（図６のステップＳ１２４０）。そして、起動中のプロセスである場合（Ｎｏ）には、対象プロセスにアタッチされているトレースライブラリ部１１のトレースフラグ１３をＯＦＦに設定する（図６のステップＳ１２５０）。また、起動中のプロセスでない場合（Ｙｅｓ）、および、ステップＳ１２５０にてプロセスを終了した場合には、ｆｔｒａｃｅ８のを終了を行う（図６のステップＳ１２６０）する。次に、ｆｔｒａｃｅ出力トレースログ９の出力を行う（図６のステップＳ１２７０）。このことにより、アプリケーショントレースログ記録が終了となる。
【００２５】
次に、トレースプログラム１２のトレース解析機能部１８の手順を図７に基づいて説明する。まず、トレースプログラム１２のトレースコマンド機能部１７が出力したｆｔｒａｃｅ出力トレースログ９の読み込みを行う（図７のステップＳ１３００）。次に、対象のアプリケーション１０本体から関数情報（各関数の名前とアドレス位置）を取得する（図７のステップＳ１３１０）。次に、ｆｔｒａｃｅ出力トレースログ９に出力されているダミーシステムコール（引数に呼び出し元関数のアドレスが位置が記録されている）から関数情報を検索し、ダミーシステムコールを置換する（図７のステップＳ１３２０）。次に、このようにダミーシステムコールと置き換えることによって、最終的なアプリケーショントレースログ２４を出力する（図７のステップＳ１３３０）。
【００２６】
上記のように構成された実施の形態１のコンピュータシステムによれば、トレースライブラリ部では、関数の入り口と出口で呼び出され内部でアプリケーショントレースログを埋め込むための関数を備え、トレースプログラム部では、アプリケーショントレース記録の開始／終了を制御するトレースコマンド機能部と、生成したｆｔｒａｃｅのトレースログを解析し最終的なアプリケーショントレースログを出力するトレース解析機能部とを備え、トレースライブラリ部によって、アプリケーションの各関数の入り口と出口において、あえてエラーとなるシステムコールを呼び出し、この状態でｆｔｒａｃｅによるトレースログを取得し、その後、取得したｆｔｒａｃｅのトレースログに含まれている、先ほどのダミーのシステムコールを、そのアプリケーション本体から取得した関数情報に置き換

えることによって、アプリケーショントレースログを作成しているため、ｆｔｒａｃｅ出力トレースログに、汎用アプリケーション関数レベルのトレースログを容易に追加することができる。このため、カーネルやアプリケーションに専用の改良を加えることなく、汎用アプリケーションの関数レベルのアプリケーショントレースログを、引数や戻り値も含めて取得することができるようになる。その結果、アプリケーションの稼動状態の確認や障害原因の切り分けが容易となり、より効率的に障害の解析を行うことができる。

【００２７】

実施の形態２.

上記実施の形態１においては、単独のプロセスを対象とした場合について示したが、Ｌｉｎｕｘ（登録商標）はマルチタスクＯＳであり、複数のプロセスが、互いにどのように交じりあっているかを明らかにしたい場合がある。そこで、本実施の形態２においては、上記実施の形態１に加えて、複数のプロセスについてのアプリケーショントレースログを取得できるようにした場合について説明する。

【００２８】

図８はこの発明の実施の形態２におけるコンピュータシステムの構成を示す図、図９は図８に示したコンピュータシステムにおけるトレースコマンドの手順を示したフローチャートである。図において、上記実施の形態１とは異なり、対象となるアプリケーション１０が複数となっている。尚、複数のプロセス３についてのアプリケーショントレースログ２４を取得する場合、それぞれのプロセス３のアプリケーショントレースログ２４をひとつのファイルにまとめることも、また、プロセス３ごとにアプリケーショントレースログ２４のファイルを分けることも両方場合が可能である。

【００２９】

次に、上記のように構成された実施の形態２のコンピュータシステムの動作について説明する。トレースプログラム１２のトレースコマンド機能部１７の動作手順について図９に基づいて説明する。まず、アプリケーショントレースログ記録を開始する場合は、すでにトレース中のプロセスがあるか否かを確認する（図９のステップＳ１２８０）。そして、トレース中のプロセスがない場合（Ｎｏ）は、ｆｔｒａｃｅ８の開始を行う（図９のステップＳ１２００）。また、トレース中のプロセスがある場合（Ｙｅｓ）、および、ステップＳ１２００にてｆｔｒａｃｅ８が開始されると、すでに起動されているプロセスがアプリケーショントレースログを出力する対象であるか否かを判断する（図９のステップＳ１２１０）。そして、起動中のプロセスが対象でない場合（Ｎｏ）は、対象アプリケーションのプロセスを新しく起動する（図９のステップＳ１２２０）。また、起動中のプロセスが対象である場合（Ｙｅｓ）、および、ステップＳ１２２０にてプロセスが起動されると、対象プロセスにアタッチされているトレースライブラリ部１１のトレースフラグ１３をＯＮに設定する（図９のステップＳ１２３０）。このことにより、アプリケーショントレースログ記録が開始される。

【００３０】

一方、アプリケーショントレースログ出力を終了する場合は、アプリケーショントレースログを出力する対象のプロセスが終了しているか否かを判断する（図９のステップＳ１２４０）。そして、対象とするプロセスが終了していない場合（Ｎｏ）は、対象プロセスにアタッチされているトレースライブラリ部１１のトレースフラグ１３をＯＦＦに設定する（図９のステップＳ１２５０）。次に、今回の対象プロセス以外にもトレース中のプロセスがあるか否かを確認する（図９のステップＳ１２９０）。そして、トレース中のプロセスがない場合（Ｎｏ）は、ｆｔｒａｃｅ８の終了を行う（図９のステップＳ１２６０）。次に、ｆｔｒａｃｅ出力トレースログ９の出力して（図９のステップＳ１２７０）、アプリケーショントレースログ記録が終了する。また、ステップＳ１２９０においてトレース中のプロセスがある場合（Ｙｅｓ）にも、終了する。

【００３１】

上記のように構成された実施の形態２のコンピュータシステムによれば、上記実施の形態１と同様の効果を奏するのはもちろんのこと、複数のプロセスを対象に、アプリケーシ

ョントレースログを出力することが可能となり、複数プロセスが互いにどのように交じりあっているかの状態をトレースすることができるようになる。

【００３２】

実施の形態３．

上記各実施の形態においては、トレースプログラムから明示的にアプリケーショントレースの開始を指示する必要があったため、通常はアプリケーショントレースログを記録しないが、本実施の形態３においては、何らかの問題が発生した場合にだけ、アプリケーショントレースログを記録する場合について説明する。本実施の形態３では、上記各実施の形態に加えて、新たにトレース条件１６を設定できるようにすることで、何か問題が発生したときに、自動的にアプリケーショントレースログ記録の開始を制御できる機能を有するものである。

【００３３】

図１０はこの発明の実施の形態３におけるコンピュータシステムの構成を示した図である。上記各実施の形態と比べて、新たにトレースライブラリ部１１の中に、トレース条件１６が追加されている。このトレース条件１６には、アプリケーションから取得した関数情報からその関数のアドレス位置を取得することで、例えば、ある関数が呼び出された場合にアプリケーショントレースログ出力を開始するなどの設定を行うことができる。また、トレース条件１６は、トレースプログラム１２のトレースコマンド機能部１７によって、外部からトレース条件データ２２が読み込まれ自由に変更することができる。

【００３４】

ここで、トレース条件１６を変更するためのデータが、トレース条件データ２２であり、例えば以下のような情報が設定できる。

＊指定した関数の入り口／出口で、アプリケーショントレースログ記録を開始／終了する。また、その関数の引数や戻り値に応じて、アプリケーショントレースログ記録を開始／終了する。

＊指定した時間に、アプリケーショントレースログ記録を開始／終了する。

＊指定した関数が、ある一定回数以上呼び出された際に、アプリケーショントレースログ記録を開始／終了する。

【００３５】

次に上記のように構成された実施の形態３のコンピュータシステムの動作について説明する。この発明の実施の形態３によるトレースライブラリ部１１のｍｃｏｕｎｔ（）関数１４の動作について図１１に基づいて説明する。まず、上記各実施の形態と同様に、トレースフラグがＯＮになっているか否かを判断する（図１１のステップＳ１０００）。そして、ＯＦＦの場合（Ｎｏ）には、処理を終了する。また、ＯＮの場合（Ｙｅｓ）には、トレース条件１６を満たしているか否かを判断する（図１１のステップＳ１０５０）。そして、満たしていない場合（Ｎｏ）には、処理を終了する。また、トレース条件１６を満たしている場合（Ｙｅｓ）には、すでにｆｔｒａｃｅが開始されているか否かを判断する（図１１のステップＳ１０６０）。そして、ｆｔｒａｃｅが開始されていない場合（Ｎｏ）には、ｆｔｒａｃｅの開始を行う（図１１のステップＳ１２００）。また、ｆｔｒａｃｅが開始されている場合（Ｙｅｓ）、および、ステップＳ１２００にてｆｔｒａｃｅを開始した場合には、上記各実施の形態と同様に、ステップＳ１０１０からステップＳ１０４０の工程を行う。

【００３６】

次に、トレースプログラム１２のトレースコマンド機能部１７の手順を図１２に基づいて説明する。まず、アプリケーショントレースログ記録を開始する場合は、ｆｔｒａｃｅ８はすでに開始されているため、上記各実施の形態のような、ｆｔｒａｃｅの開始の工程は省略され、以下、上記各実施の形態と同様に、ステップＳ１２１０ないしステップＳ１２３０の工程を行う。また、終了する場合には、上記各実施の形態と同様の工程にて行う。

【００３７】

上記のように構成された実施の形態３によれば、上記各実施の形態と同様の効果を奏するのはもちろんのこと、ユーザが明示的にアプリケーショントレースの開始を指示しなくても、あらかじめ設定しておいた条件によって、自動的にアプリケーショントレースログの記録ができるようになる。

【００３８】

実施の形態４.

上記各実施の形態においては、対象とするアプリケーション１０の全ての関数レベルのアプリケーショントレースログをファイルに出力している。そのため、アプリケーション１０によっては、非常に膨大のアプリケーショントレースログが出力されてしまい、ログが溢れてしまう恐れがある。そこで、本実施の形態４は、上記各実施の形態に加えて、ｆｔｒａｃｅ出力トレースログ９に対して、フィルタリングを施し、必要な箇所に絞ってアプリケーショントレースログ２４を出力できるものについて説明する。

【００３９】

図１３はこの発明の実施の形態４におけるコンピュータシステムのトレースプログラムの部分の詳細を示した図である。上記各実施の形態と比べて、トレースプログラム１２のトレース解析機能部１８が、フィルタリング設定データ２３を入力とすることで、アプリケーショントレースログ２４の情報をフィルタリングする。このとき、フィルタリング設定データ２３には、例えば、以下のような情報を設定することができる。

＊指定した関数の入り口／出口を、アプリケーショントレースログ２４として出力する／しない。また、その関数の引数／戻り値に応じて、アプリケーショントレースログ２４として出力する／しない。

＊指定した時間に呼び出された関数の入り口／出口を、アプリケーショントレースログ２４として出力する。

＊複数回呼び出される関数で最初に呼び出された関数の入り口／出口を、アプリケーショントレースログ２４として出力する。尚、これらのフィルタリングは、後から何度でも自由に条件を変えて行うことができる。

【００４０】

上記のように構成された実施の形態４のコンピュータシステムによれば、上記各実施の形態と同様の効果を奏するのはもちろんのこと、事前に、不必要なアプリケーショントレースログをフィルタリングすることでき、障害の解析が容易となる。

【００４１】

実施の形態５.

上記各実施の形態においては、出力されるアプリケーショントレースログ２４は、単なるログの並びであるため、上記実施の形態３のように、そのログのサイズが膨大になった場合、人間が解析を行うのが困難になる場合があった。そこで、本実施の形態５においては、上記各実施の形態に加えて、出力したアプリケーショントレースログ２４を、人間が理解しやすいグラフの形に変更し、可視化する場合について説明する。

【００４２】

図１４はこの発明の実施の形態５のコンピュータシステムにおけるトレースプログラムの部分の詳細を説明するための図である。上記各実施の形態と比べて、トレースプログラム１２に、新たに可視化機能部１９が追加されている。可視化機能部１９は、トレース解析機能部１８が出力したアプリケーショントレースログ２４を入力として、アプリケーショントレースグラフ２５を出力する。具体的には図１５および図１６に示すようなアプリケーショントレースグラフ２５の一例である。これらの図のように、出力されるグラフとしては、状態遷移図やシーケンス図が想定される。

【００４３】

上記のように構成された実施の形態５のコンピュータシステムによれば、上記各実施の形態と同様の効果を奏するのはもちろんのこと、単なるログの並びであったアプリケーショントレースログをグラフ化することができ、より容易に解析することが可能となる。

【００４４】

実施の形態６．

上記各実施の形態においては、スタンドアローンのシステム構成について示したが、近年の複雑化したコンピュータシステムにおいては、遠隔地にあるような端末に対してアプリケーショントレースログ２４を取得したい場合がある。そこで、本実施の形態６は、上記各実施の形態に加えて、コンピュータシステムを用いたネットワークシステムを構築し、遠隔地からネットワーク越しに、アプリケーショントレースログ記録の開始／終了や、アプリケーショントレースログ２４の取得ができるようにする場合について説明する。

【００４５】

図１７はこの発明の実施の形態６おけるコンピュータシステムを用いたネットワークシステムの構成を示す図である。上記各実施の形態と異なり、一の端末としてのサーバ端末２７と他の端末としてのクライアント端末２６とがネットワーク３０に接続された状態となっている。そして、クライアント端末２６上には、トレースクライアント機能部２１が備えられている。また、サーバ端末２７上には、上記各実施の形態と同様なコンピュータシステムに加えて、トレースプログラム部１２にはトレースサーバ機能部２０が追加される。尚、クライアント端末２６は、自身のアプリケーショントレースを行わないならば、サーバ端末２７と同様に、上記各実施の形態と同様なコンピュータシステムを備えていてもよい。但し、本実施の形態においては、クライアント端末２６は、自身でｆｔｒａｃｅを直接使用しないため、Ｌｉｎｕｘ（登録商標）である必要はない。

【００４６】

次に、上記のように構成された実施の形態６のコンピュータシステムを用いたネットワークシステムの動作について説明する。まず、クライアント端末２６のトレースクライアント機能部２１は、ネットワーク３０でつながった遠隔地のサーバ端末２７のトレースサーバ機能部２０と接続を行い、アプリケーショントレースログの記録の開始および終了を指示する。次に、指示を受けたトレースサーバ機能部２０は、その指示にしたがって、上記各実施の形態と同様に、クライアント端末２６における、アプリケーショントレースログ記録の開始／終了を行う。そして、アプリケーショントレースログ記録を終了し、最終的なアプリケーショントレースログ２４やアプリケーショントレースグラフ２５が出力されたタイミングで、そのデータをクライアント端末２６上のトレースクライアント機能部２１に送信する。

【００４７】

上記のように構成された実施の形態６によれば、上記各実施の形態と同様の効果を奏するのはもちろんのこと、遠隔地に設置されたＬｉｎｕｘ（登録商標）コンピュータシステムでも、リモートでアプリケーショントレースログを取得することができ、一層の解析の容易化を図ることができる。

【００４８】

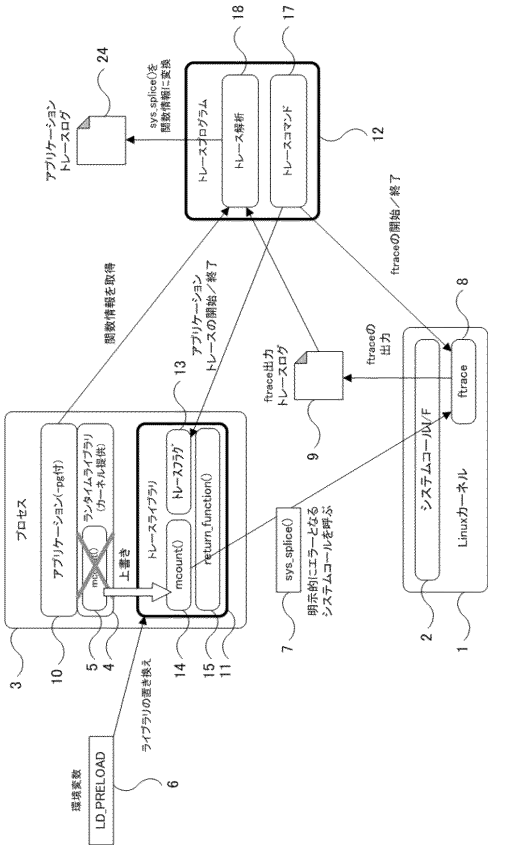
尚、本発明は、その発明の範囲内において、各実施の形態を自由に組み合わせたり、各実施の形態を適宜、変形、省略することが可能である。

【符号の説明】

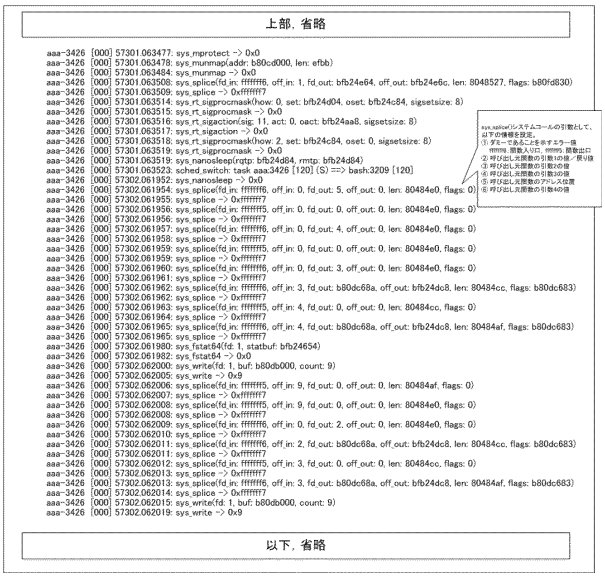
【００４９】

１　Ｌｉｎｕｘ（登録商標）カーネル、８　ｆｔｒａｃｅ、
９　ｆｔｒａｃｅ出力トレースログ、１１　トレースライブラリ部、
１２　トレースプログラム部、１７　トレースコマンド機能部、
１８　トレース解析機能部、１９　可視化機能部、２０　トレースサーバ機能部、
２１　トレースクライアント機能部、２２　トレース条件データ、
２３　フィルタリング設定データ、２４　アプリケーショントレースログ、
２６　クライアント端末、２７　サーバ端末、３０　ネットワーク。

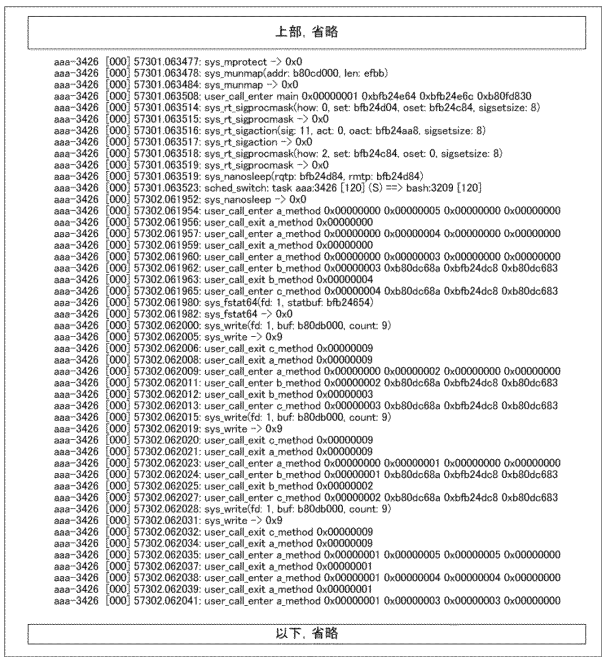
【図１】


アプリケーション
トレースログ
sys_spliceの
関数情報に変換
トレースプログラム
トレース解析
トレースコマンド
ftraceの開始／終了
関数情報を取得
アプリケーション
トレースの開始／終了
ftrace出力
トレースログ
ftraceの
出力
ftrace
システムコールI/F
Linuxカーネル
プロセス
アプリケーション(-pg付)
ランタイムライブラリ
(カーネル提供)
上書き
トレースライブラリ
トレースフラグ
mcount()
return_function()
sys_splice()
明示的にエラーとなる
システムコールを呼ぶ
環境変数
LD_PRELOAD
ライブラリの置き換え

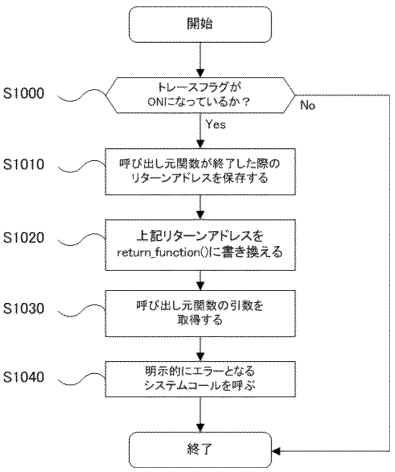
【図２】


上部，省略
以下，省略

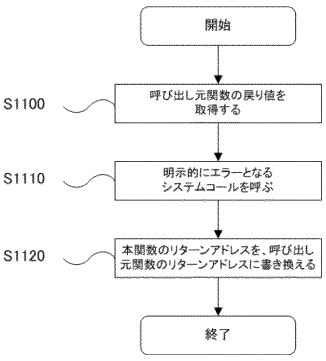
【図３】


上部，省略
以下，省略

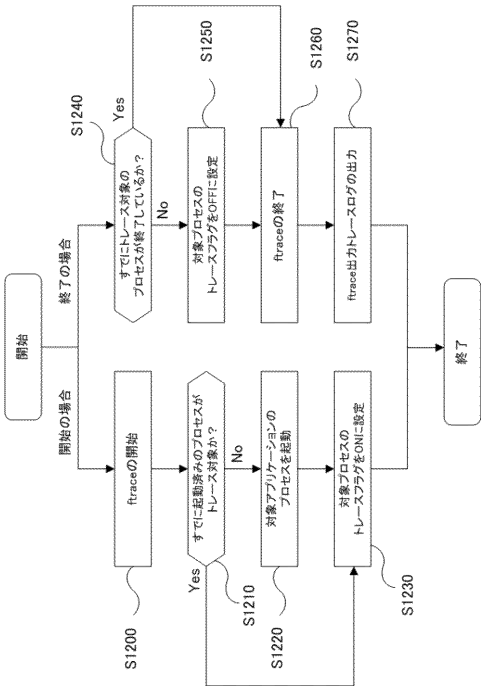
【図４】


開始
S1000
トレースフラグが
ONになっているか？
No
Yes
S1010
呼び出し元関数が終了した際の
リターンアドレスを保存する
S1020
上記リターンアドレスを
return_function()に書き換える
S1030
呼び出し元関数の引数を
取得する
S1040
明示的にエラーとなる
システムコールを呼ぶ
終了

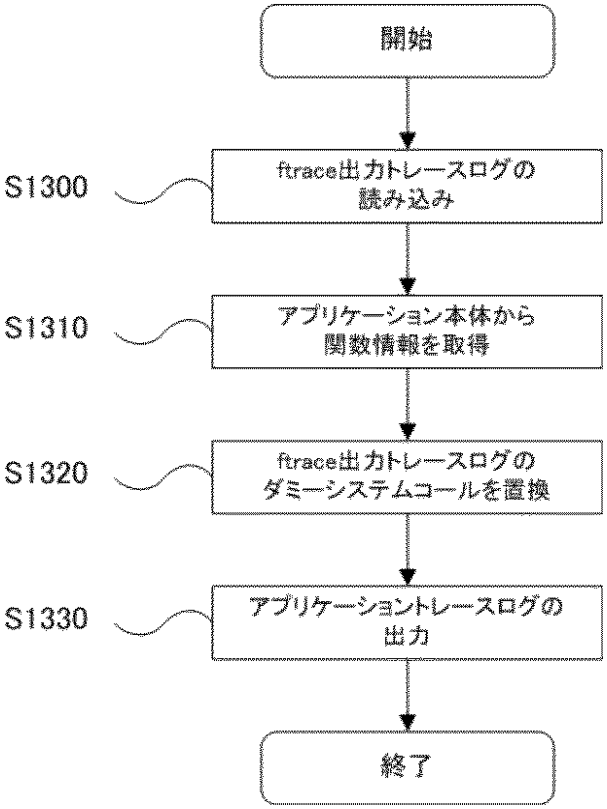
【図５】


開始
S1100
呼び出し元関数の戻り値を
取得する
S1110
明示的にエラーとなる
システムコールを呼ぶ
S1120
本関数のリターンアドレスを、呼び出し
元関数のリターンアドレスに書き換える
終了

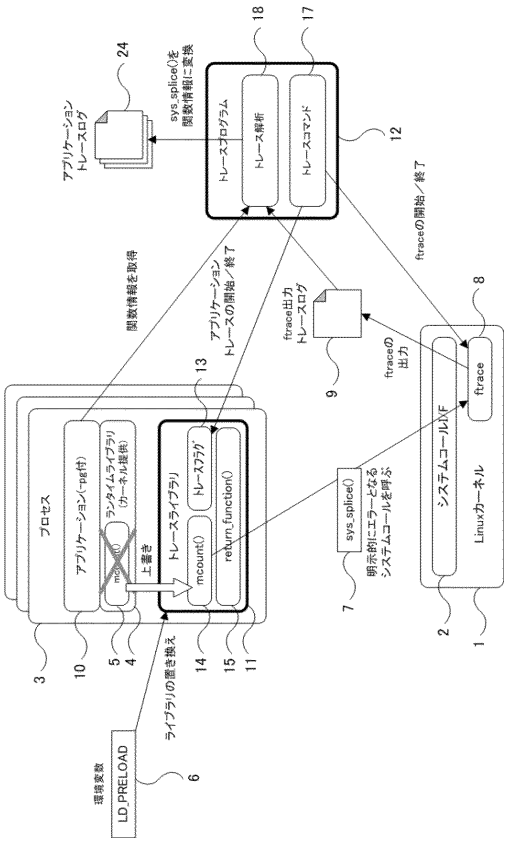
【図６】


開始
終了の場合
開始の場合
すでにトレース対象のプロセスが終了しているか？
Yes
No
対象プロセスのトレースフラグをOFFに設定
ftraceの終了
ftrace出力トレースログの出力
ftraceの開始
すでに起動済みのプロセスがトレース対象か？
対象アプリケーションのプロセスを起動
対象プロセスのトレースフラグをONに設定
終了
S1200
S1210
S1220
S1230
S1240
S1250
S1260
S1270

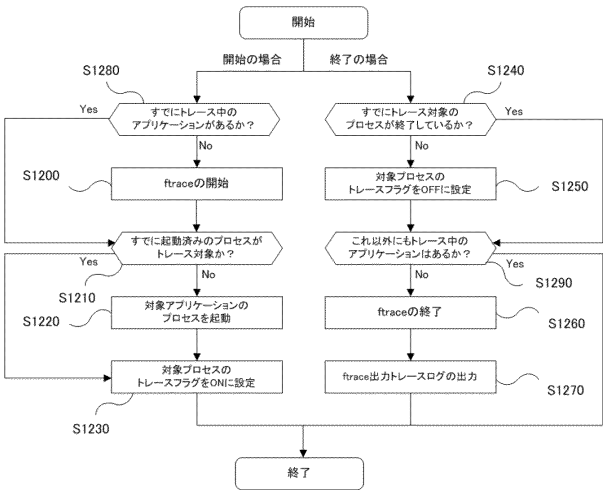
【図７】


開始
S1300
ftrace出力トレースログの読み込み
S1310
アプリケーション本体から関数情報を取得
S1320
ftrace出力トレースログのダミーシステムコールを置換
S1330
アプリケーショントレースログの出力
終了

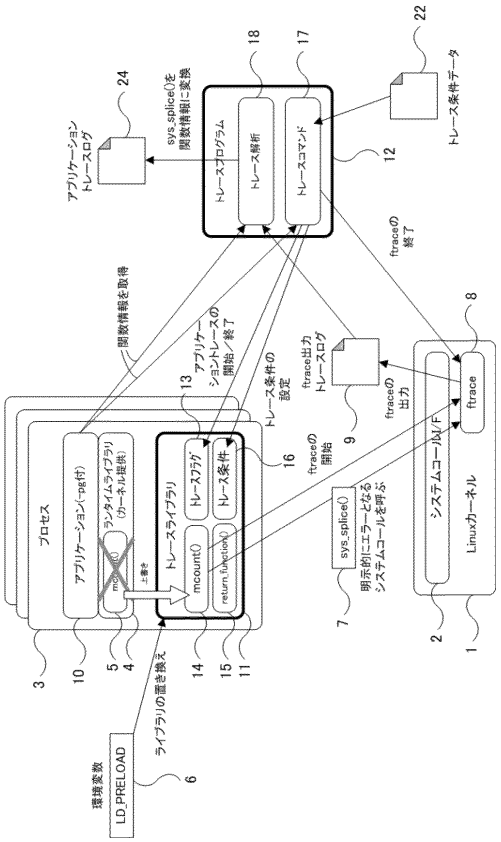
【図８】


アプリケーショントレースログ
sys_spliceを関数情報に変換
トレースプログラム
トレース解析
トレースコマンド
関数情報を取得
アプリケーショントレースの開始／終了
ftraceの開始／終了
ftrace出力トレースログ
ftraceの出力
プロセス
アプリケーション(-pg付)
ランタイムライブラリ(カーネル連携)
mcount()
上書き
トレースライブラリ
トレースフラグ
mcount()
return_function()
sys_splice()
明示的にエラーとなるシステムコールを呼ぶ
システムコールI/F
ftrace
Linuxカーネル
ライブラリの置き換え
環境変数
LD_PRELOAD

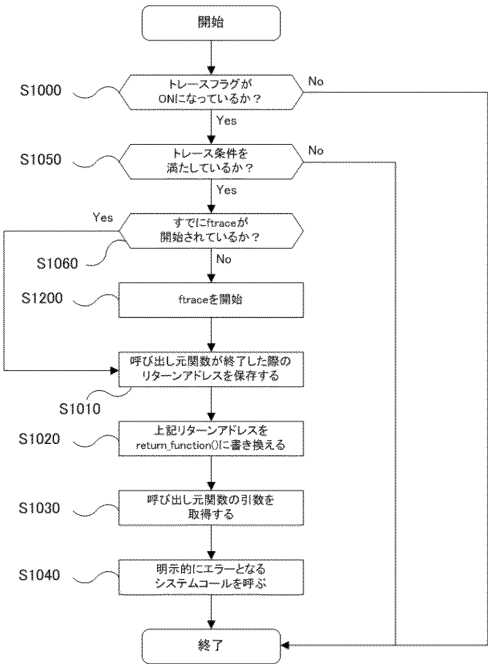
【図９】


開始
開始の場合
終了の場合
S1280
すでにトレース中のアプリケーションがあるか？
Yes
No
S1200
ftraceの開始
すでに起動済みのプロセスがトレース対象か？
S1210
S1220
対象アプリケーションのプロセスを起動
対象プロセスのトレースフラグをONに設定
S1230
S1240
すでにトレース対象のプロセスが終了しているか？
対象プロセスのトレースフラグをOFFに設定
S1250
これ以外にもトレース中のアプリケーションはあるか？
S1290
ftraceの終了
S1260
ftrace出力トレースログの出力
S1270
終了

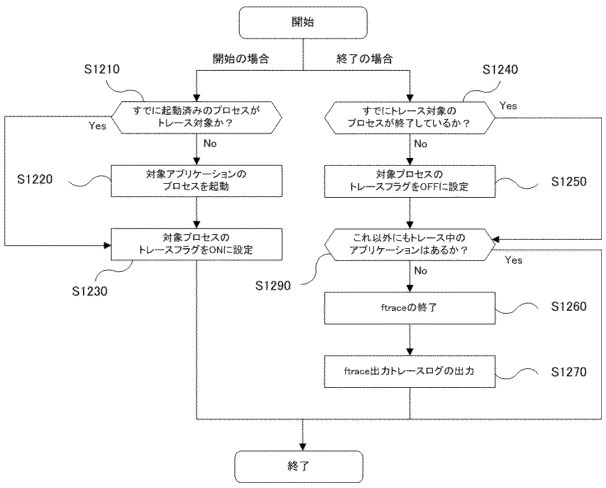
【図１０】



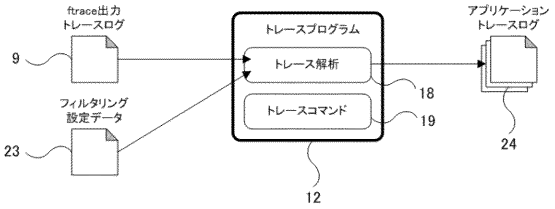
【図１１】



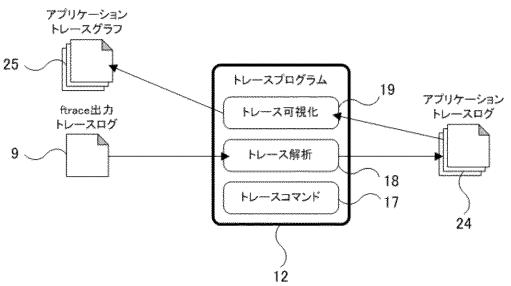
【図１２】



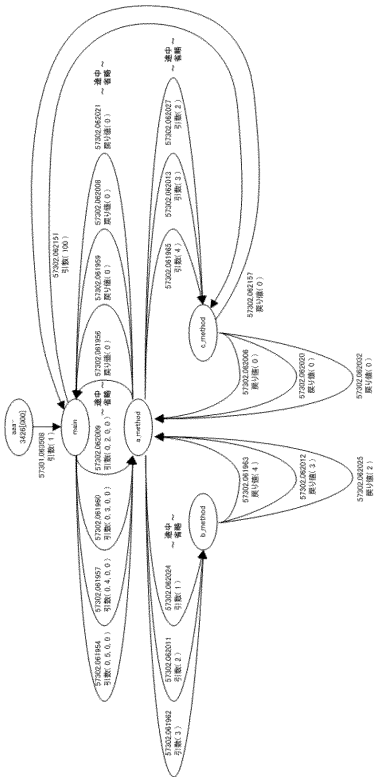
【図１３】



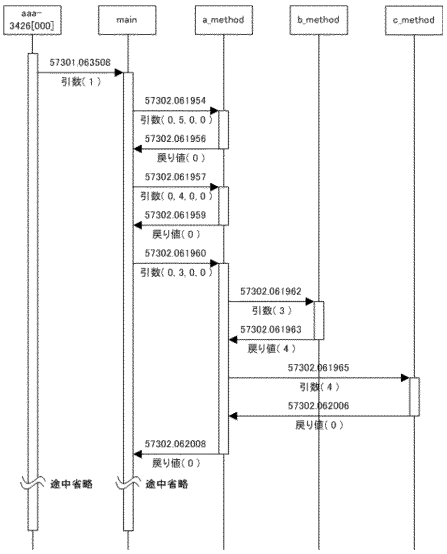
【図１４】



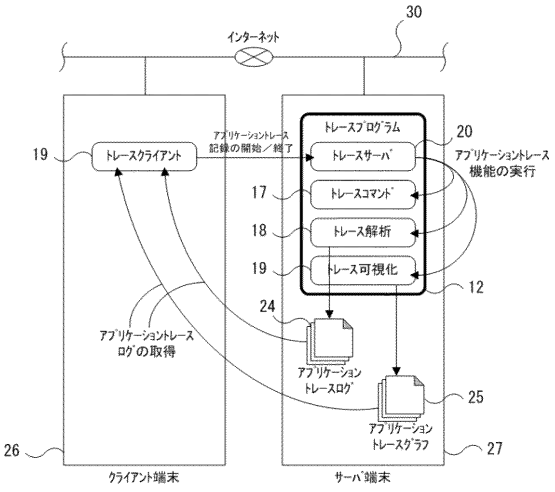
【図１５】



【図１６】



【図１７】



30：ネットワーク

フロントページの続き

審査官　多賀　実

(56)参考文献　特開２００９－２３７６１０（ＪＰ，Ａ）
丸山直也　外１名，大規模分散システムにおける故障の解析，電子情報通信学会技術研究報告，社団法人電子情報通信学会，２００６年　7月２５日，第１０６巻，第１９８号，ｐ．１９－２４
今井浩二，ＵＮＩＸ処方箋，ＵＮＩＸ　ＵＳＥＲ，ソフトバンクパブリッシング株式会社，２００５年　2月　1日，第１４巻，第2号，ｐ．１３３－１３７

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ１１／２８－１１／３６