(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4681868号
(P4681868)
(45)発行日　平成23年5月11日(2011.5.11)　(24)登録日　平成23年2月10日(2011.2.10)

(51)Int.Cl.　F I
G06F 11/28 (2006.01)　G06F 11/28 310A

請求項の数 10　(全 14 頁)

(21)出願番号　特願2004-364783 (P2004-364783)
(22)出願日　平成16年12月16日 (2004.12.16)
(65)公開番号　特開2006-172206 (P2006-172206A)
(43)公開日　平成18年6月29日 (2006.6.29)
審査請求日　平成19年12月17日 (2007.12.17)

(73)特許権者　000001007
キヤノン株式会社
東京都大田区下丸子３丁目３０番２号
(74)代理人　100076428
弁理士　大塚　康徳
(74)代理人　100112508
弁理士　高柳　司郎
(74)代理人　100115071
弁理士　大塚　康弘
(74)代理人　100116894
弁理士　木村　秀二
(74)代理人　100130409
弁理士　下山　治
(74)代理人　100134175
弁理士　永川　行光

最終頁に続く

(54)【発明の名称】情報処理装置及びその制御方法、コンピュータプログラム及び記憶媒体

(57)【特許請求の範囲】
【請求項１】
第１のモジュールと、第２のモジュールと、前記第１のモジュールから前記第２のモジュール内の関数への呼び出しを仲介し、前記呼び出しに応じた前記第２のモジュールにおける処理のログを取得するための第３のモジュールと、を実行する情報処理装置であって、
前記第３のモジュールが、
前記ログを取得するログ取得手段と、
取得された前記ログから、前記第２のモジュール内の関数の引数として含まれている、バイナリデータの格納されている領域の先頭を表す書込開始アドレスと、データサイズとの情報を取得する情報取得手段と、
前記書込開始アドレスと前記データサイズとから、前記バイナリデータの書き込まれている領域を決定する領域決定手段と、
決定された前記領域へのアクセスが許可されているか否かをＯＳに問い合わせる問い合わせ手段と、
前記問い合わせの結果により前記決定された領域のうち一部の領域のみのアクセスが許可されている場合に、前記一部の領域に対応する前記バイナリデータの一部のメモリへの書込を行う書込制御手段と
を備えることを特徴とする情報処理装置。
【請求項２】

前記書込制御手段は、前記問い合わせの結果により前記決定された全領域のアクセスが許可されている場合に、前記バイナリデータを前記決定された領域からメモリへ書き込むことを特徴とする請求項１に記載の情報処理装置。
【請求項３】
前記書込制御手段は、前記問い合わせの結果により前記決定されたいずれの領域へのアクセスも許可されない場合に、前記バイナリデータの前記決定された領域からメモリへの書き込みを抑制することを特徴とする請求項１に記載の情報処理装置。
【請求項４】
前記一部の領域は、１バイト単位に決定されることを特徴とする請求項１に記載の情報処理装置。
【請求項５】
第１のモジュールと、第２のモジュールと、前記第１のモジュールから前記第２のモジュール内の関数への呼び出しを仲介し、前記呼び出しに応じた前記第２のモジュールにおける処理のログを取得するための第３のモジュールと、を実行する情報処理装置の制御方法であって、
前記第３のモジュールが、
前記ログを取得するログ取得工程と、
取得された前記ログから、前記第２のモジュール内の関数の引数として含まれている、バイナリデータの格納されている領域の先頭を表す書込開始アドレスと、データサイズとの情報を取得する情報取得工程と、
前記書込開始アドレスと前記データサイズとから、前記バイナリデータの書き込まれている領域を決定する領域決定工程と、
決定された前記領域へのアクセスが許可されているか否かをＯＳに問い合わせる問い合わせ工程と、
前記問い合わせの結果により前記決定された領域のうち一部の領域のみのアクセスが許可されている場合に、前記一部の領域に対応する前記バイナリデータの一部のメモリへの書込を行う書込制御工程と
を実行することを特徴とする情報処理装置の制御方法。
【請求項６】
前記書込制御工程では、前記問い合わせの結果により前記決定された全領域へのアクセスが許可されている場合に、前記バイナリデータが前記決定された領域からメモリに書き込まれることを特徴とする請求項５に記載の情報処理装置の制御方法。
【請求項７】
前記書込制御工程では、前記問い合わせの結果により前記決定されたいずれの領域へのアクセスも許可されない場合に、前記バイナリデータの前記決定された領域からメモリへの書き込みが抑制されることを特徴とする請求項５に記載の情報処理装置の制御方法。
【請求項８】
前記一部の領域は、１バイト単位に決定されることを特徴とする請求項５に記載の情報処理装置の制御方法。
【請求項９】
請求項５乃至８のいずれか１項に記載の情報処理装置の制御方法をコンピュータに実行させるためのコンピュータプログラム。
【請求項１０】
請求項９に記載のコンピュータプログラムを記憶したコンピュータで読み取り可能な記憶媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、情報処理装置及びその制御方法、コンピュータプログラム及び記憶媒体に関する。

【背景技術】
【０００２】
再現率の低いソフトウェアの障害に対しては、ソフトウェアの処理ログを取得することによって対処している場合が多いが、従来、処理ログの取得はアプリケーションソフトウェアのモジュール自体に手を加えて、処理ログ取得ルーチンを追加することによって行なわれている。しかし、ログ取得コードを埋め込む等のアプリケーションソフトウェアの修正を必要とする上記の方式では、修正処理が煩雑化する。
【０００３】
そこで、複数にモジュール分けされているソフトウェアにおいて、アプリケーションソフトウェアに相当するモジュールから別のモジュール内に存在する関数への呼び出しを仲介し、当該呼び出しに応じた別モジュールにおける処理のログを取得するログ取得用モジュールを提供することにより、アプリケーションソフトウェア自体の煩雑な修正を行わずとも、処理ログを取得可能とする方法が提案されている。
【０００４】
このようなログ取得用モジュールによりログをバイナリデータとして取得する場合、関数のパラメータに定義されているメモリポインタに基づいてデータへアクセスし保存することとなる。この時、アプリケーションソフトウェアの障害等により、バイナリデータの一部もしくは全てがアクセス禁止となる不正な領域に置かれてしまった場合、ログ取得ソフトウェアがその領域にアクセスすることで例外が発生してしまうので、対応する例外処理を行なわなければならない（特許文献１参照）。
【特許文献１】特開２００４－３８３１１号公報
【発明の開示】
【発明が解決しようとする課題】
【０００５】
これに対して、また、アプリケーションソフトウェア自体を変更しメモリの不正アクセスを検知する方法は提案されているが、アプリケーションソフトウェア自体の変更が煩雑であるため効果的ではない。
【０００６】
そこで、本発明では、アプリケーションソフトウェア自体の変更を行わずに、ログの不正領域への書き込み防止を可能とすることを目的とする。
【課題を解決するための手段】
【０００７】
上記課題を解決するために、第１のモジュールと、第２のモジュールと、前記第１のモジュールから前記第２のモジュール内の関数への呼び出しを仲介し、前記呼び出しに応じた前記第２のモジュールにおける処理のログを取得するための第３のモジュールと、を実行する情報処理装置であって、
前記第３のモジュールが、
前記ログを取得するログ取得手段と、
取得された前記ログから、前記第２のモジュール内の関数の引数として含まれている、バイナリデータの格納されている領域の先頭を表す書込開始アドレスと、データサイズとの情報を取得する情報取得手段と、
前記書込開始アドレスと前記データサイズとから、前記バイナリデータの書き込まれている領域を決定する領域決定手段と、
決定された前記領域へのアクセスが許可されているか否かをＯＳに問い合わせる問い合わせ手段と、
前記問い合わせの結果により前記決定された領域のうち一部の領域のみのアクセスが許可されている場合に、前記一部の領域に対応する前記バイナリデータの一部をメモリへ書込を行う書込制御手段と
を備える。
【発明の効果】

【０００８】
本発明によれば、アプリケーションソフトウェア自体の変更を行わずに、ログの不正領域への書き込みを防止することができる。
【発明を実施するための最良の形態】
【０００９】
以下、添付する図面を参照して本発明の実施形態を説明する。
【００１０】
図１は、本実施形態に対応する情報処理装置の構成の一例を示す図である。説明を簡略化するために、本実施形態では、情報処理システムが１台のPC内部に構築されるものとするが、本発明の特徴は１台のPC内部に構築されるか、あるいは複数のPCにネットワークシステムとして構築されるかによらず有効である。
【００１１】
本情報処理装置には、CPU１、チップセット２、RAM３、ハードディスクコントローラ４、ディスプレイコントローラ５、ハードディスクドライブ６、CD-ROMドライブ７、ディスプレイ８が搭載されている。また、CPU１とチップセット２とを繋ぐ信号線１１、チップセット２とRAM３とを繋ぐ信号線１２、チップセット２と各種周辺機器４、５とを繋ぐ周辺機器バス１３、ハードディスクコントローラ４とハードディスクドライブ６とを繋ぐ信号線１４、ハードディスクコントローラ４とCD-ROMドライブ７とを繋ぐ信号線１５、ディスプレイコントローラ５とディスプレイ８とを繋ぐ信号線１６が搭載されている。
【００１２】
本実施形態に対応する情報処理装置を説明するために、まず図２を参照して、複数のモジュールに分かれたソフトウェアが、通常の状態でどのようにメモリにロードされるかを説明する。図２は、RAMの内部構成の一例を示す図である。
【００１３】
通常、複数のモジュールに分割されたソフトウェアは、全体の制御を行なう実行ファイルEXE(２３)と、モジュールとして存在しEXEの補完的な役割を担うダイナミックリンクライブラリDLL(２７)とに分かれて存在する。そして、RAM３にはEXEとDLLの両方がロードされる。EXEはコードセグメント(２８)とデータセグメント(２９)、及び、インポート関数アドレステーブル(２２)で構成される。このうち、インポート関数アドレステーブルは、関数の所属するDLLによって更に分割されており(２１及び２４)、DLLごとにそれぞれの関数がロードされたアドレスが書かれている(３０～３５)。
【００１４】
DLLの関数の実体は、DLLごとに分けて(２５,２６)ロードされ、それぞれの関数は該当するDLLの一部としてロードされる(３６～４１)。この図では、１本のEXEがA.DLL及びB.DLLの２つのダイナミックリンクライブラリ内の関数を使用している例を示しており、実際に使用される関数はFunc AA, Func AB, Func AC, Func BA, Func BB, Func BCの６個となっている。
【００１５】
EXEのコードセグメント２８内にあるコードが関数Func AAを呼び出す場合には、まずインポート関数アドレステーブル内に書かれたFunc AAのアドレス(３０)が読み込まれる。ここには実際にはA.DLLの一部として読み込まれたFunc AAコード(３６)のアドレスが書かれており、そのアドレスをコールすることによって、EXEのコードはA.DLLのFunc AAを呼び出すことができる。
【００１６】
次に、ログ取得用のコードとしてIAT Patch(Import Address Table Patch)という手法を用いて関数呼び出しを仲介する場合の、情報処理装置のメモリ構成の一例を図３を参照して説明する。
【００１７】
ログ取得が開始されると、メモリ内にはIAT Patch用のDLLであるC.DLL(５８)がロードされる。C.DLLはインポート関数アドレステーブル(５２)内に書かれた関数のアドレスを

、C.DLL内のログ取得コードであるFunc CAA, Func CAB, Func CAC, Func CBA, Func CBB, Func CBCのアドレスに書き換える(６１～６６)。C.DLL内のFunc CAA, Func CAB, Func CAC, Func CBA, Func CBB, Func CBCのコード(７３～７８)は、ログを記録すると共に、元々関数呼び出しを受けるべくメモリにロードされている、該当する関数であるFunc AA, Func AB, Func AC, Func BA, Func BB, Func BC(６７～７２)を呼び出す。

【００１８】

図４は、図３におけるIAT Patchの処理をあらわすタイミングチャートである。説明を簡略化するために、この図ではEXEがA.DLL内のFunc AAを呼び出す際に、IAT Patchによるログ取得コードがどのように動作するかの例をあらわしている。他の関数の場合についても同様の処理が行われることはいうまでもない。

【００１９】

EXE(９１)がFunc AAをコールすると(９４)、C.DLL内にあるログ取得コードがDLL名/関数名をメモリに保存し、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、メモリに保存する(９５)。その後C.DLLは本来呼び出されるはずであった、A.DLL(９３)内のFunc AAをコールする(９６)。A.DLLのFunc AA処理(９７)が終了し、C.DLLに制御がリターンすると(９８)、C.DLLはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、メモリに保存する(９９)。その後、C.DLLは保存したログ情報をファイルに書き込み(１００)、あたかもA.DLLのFunc AAが通常通りに終了したかのように、EXEにリターンする(１０１)。

【００２０】

図５は、本実施形態に対応する情報処理装置において実行形式ファイルＥＸＥが実行される場合の動作の一例を示す図である。通常は実行形式のEXE(１１３)が、DLL-１(１１６)やDLL-２(１１７)内の関数を呼び出すが、ここではAPIトレーサと呼ばれるログ取得コードを埋め込み(１１４)、処理ログを生成している(１１５)。APIトレーサは、DLL-１やDLL-２の関数定義を記述したファイル(１１１)と、どのDLLのどの関数のインポート関数テーブルを書き換えてログを取得するかの設定シナリオ(１１２)を元に動作する。

【００２１】

図６は、本実施形態に対応する情報処理装置において、実行ファイルEXE(１１８)がCOM(Component Object Model：コンポーネント・オブジェクト・モデル)サーバでエクスポートされているインターフェースのインスタンスを作成する場合の、メモリ構成の一例を示す図である。

【００２２】

通常、インターフェースのインスタンス作成を行うと、COMサーバ内で、要求されたインターフェース(１２１, １２２)と、そのメソッド(１３０～１３５)が作成され、それらは共にメモリ上にロードされる。ここで、virtual address tableは作成された各インターフェース毎に作られ(１１８, １２０)、作成要求を行ったEXEに渡される。このvirtual address tableには各メソッドについて作成されたアドレスが書かれている(１２４～１２９)。EXEはこれら情報を利用し、各インターフェースに対して呼び出しを行う。図６では、１本のEXEがInterface A及びInterface Bの２つのインターフェースのインスタンスを作成しており、そのインターフェース内部のメソッドを使用している例を示しており、実際に使用されているメソッドは、Method AA, Method AB, Method AC, Method BA, Method BB, Method BCとなっている。

【００２３】

EXEのコードが関数Method AAを呼び出す場合には、まずvirtual address table内に書かれたMethod AAのアドレス(１２４)が読み込まれる。このアドレス（１２４）にはCOMサーバのInterface Aの一部として作成されたMethod AAコード(１３０)のアドレスが書かれており、そのアドレスをコールすることによって、EXEのコードはInterface AのMethod AAを呼び出すことができる。

【００２４】

図７は、本実施形態に対応する情報処理装置のメモリ構成をあらわす図であり、図６とは、ログ取得用のコードとしてVTable Patch(virtual address table Patch)という手法を用いて、メソッド呼び出しを仲介しているという点で異なっている。

【００２５】

ログ取得が開始されると、メモリ内にはVTable Patch用のDLL(１４３)がロードされる。このDLLはvirtual address table(１３６，１３８)内に書かれたメソッドのアドレスを、DLL内のログ取得コードであるMethod A'A, Method A'B, Method A'C, Method B'A, Method B'B, Method B'Cのアドレスに書き換える(１４５～１５０)。DLL内のMethod A'A, Method A'B, Method A'C, Method B'A, Method B'B, Method B'Cのコード(１５７～１６２)は、ログを記録すると共に、元々のメソッド呼び出しを受けるべくメモリにロードされていたMethod AA, Method AB, Method AC, Method BA, Method BB, Method BC(１５７～１６２)を呼び出す。

【００２６】

図８は、図７におけるVTable Patchの処理をあらわすタイミングチャートである。説明を簡略化するために、この図ではEXEがCOMサーバ内のInterface AのMethod AAを呼び出す際に、VTable Patchによるログ取得コードがどのように動作するかの例をあらわしている。他のメソッドの場合についても同様の処理が行われることはいうまでもない。

【００２７】

EXE(１６３)がMethod AAをコールすると(１６６)、DLL内にあるログ取得コードがモジュール名/インターフェース名/メソッド名をメモリに保存し、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、メモリに保存する(１６７)。その後DLLは本来呼び出されるはずであった、COMサーバ(１６５)内のMethod AAをコールする(１６８)。COMサーバのMethod AA処理(１６９)が終了し、DLLに制御がリターンすると(１７０)、DLLはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、メモリに保存する(１７１)。その後、DLLは保存したログ情報をファイルに書き込み(１７２)、あたかもCOMサーバのMethod AAが通常通りに終了したかのように、EXEにリターンする(１７３)。

【００２８】

図９は、本実施形態に対応する情報処理装置において実行形式ファイルＥＸＥが実行される場合の動作の一例を示す図である。通常は実行形式のEXE(１７６)が、COMサーバ-１(１７９)やCOMサーバ-２(１８０)内のメソッドを呼び出すが、ここではAPIトレーサと呼ばれるログ取得コードを埋め込み(１７７)、処理ログを生成している(１７８)。APIトレーサは、COMサーバ-１(１７９)やCOMサーバ-２の関数定義を記述したファイル(１７４)と、どのCOMサーバのどのインターフェースのどのメソッドのvirtual address tableを書き換えてログを取得するかの設定シナリオ(１７５)を元に動作する。

【００２９】

図１０は、本実施形態に対応する情報処理装置において利用される、関数及びメソッドのパラメータの形式や、戻り値の形式を指示する関数定義ファイルの一例を示す図である。この関数定義ファイルは、図５の関数定義１１１や図９の関数定義１７４に対応する。

【００３０】

関数定義ファイルは、DLL名及び関数/メソッド名を記述し、その関数/メソッドに対する、パラメータ及び戻り値の型が示されている。本実施形態は、この関数定義ファイルによって指示された内容に基づいて、それぞれの関数/メソッドがどのようなパラメータ/戻り値を有しているかが判定され、その内容がログとして取得される。

【００３１】

図１０において示されたA.DLL内の関数FuncABに対する関数定義は、本発明の実施形態を最もよくあらわすものの一つである。引数には、データの格納先（書込開始アドレス）として示されるポインタpBufと、そのデータのサイズとして示されるdwBufSizeが定義されており、pBufとdwBufSizeとの相関、すなわちpBufに格納されたデータのサイズがdwBuf

Sizeによって指示されているという相関が定義されている。

【００３２】

図１１は、図１０に示した関数定義ファイルを用いて、本発明の情報処理装置において取得されたログの一例を示す図である。それぞれの呼び出しに対して、関数/メソッドが呼び出された時刻、及びその際のパラメータ/戻り値が、ログとして生成される。

【００３３】

図１１において示されたA.DLL内の関数FuncABに対する呼び出しのログは、図１０の関数定義ファイルと共に、本発明の実施形態の特徴を最もよく表すものの一つである。図１０に示した関数定義ファイルに基づいて、FuncABに対する呼び出しの際には、引数としてのpBufの値（0x5034206D）だけでなく、その値が指し示すアドレス上にある、dwBufSize分すなわち２４byte分のデータが、通常のログとは別のバイナリログとして保存され、そのバイナリログファイル内に保存された該当するFuncABの呼び出しのID、すなわちDataIDが、追加情報としてログに保存される。

【００３４】

図１２は、図１１に示したログと共に保存されるバイナリログファイルの実体を一例として示す図である。ここでは、各バイナリログファイルを識別するためのタグとしてDataIDが保存され、そのDataID毎にバイナリデータの本体が保存されている。

【００３５】

図１０乃至図１２に示された関数定義ファイル、ログ、バイナリログは、指示するデータが入力データか出力データか、あるいは戻り値かに依存せずに有効である。例えば、関数からの出力パラメータとしてデータアドレスが戻され、更に関数からの戻り値としてデータサイズが出力された場合にも、同様にデータ本体のログを取得することが可能である。

【００３６】

［第１の実施形態］

以上の構成に基づいて、本発明の情報処理装置において実行される処理について説明する。図１３は、本実施形態におけるバイナリログの保存にかかわる処理の一例に対応するフローチャートである。

【００３７】

図１３（a）において、設定された関数/メソッドが呼び出されることによって、処理が開始されると(Ｓ１３０１)、本ソフトウェアはDLL名/関数名/呼び出し時の時刻をHDDに保存し(Ｓ１３０２)、その呼び出しに対してのパラメータをHDDに保存する(Ｓ１３０３)。次に本ソフトウェアは、パラメータがメモリアドレスとして定義されているかどうかを、第１１図に示した関数定義ファイルに基づいて判断し(Ｓ１３０４)、定義されている場合はバイナリ保存処理(Ｓ１３０５)を行なう。このバイナリ保存処理の詳細は、図１３（ｂ）に示すとおりであり、ここでは関数定義のサイズをあらわす引数からデータサイズを取得し(Ｓ１３１４)、ポインタpBufで示されるメモリアドレス（書込開始アドレス）からデータサイズ分に相当するメモリ領域を算出し、算出されたメモリ領域が正しい領域であるかをＯＳへ問い合わせる(Ｓ１３１５)。正しい領域であると判定された場合のみ、そのサイズ分のデータをメモリから読み込んで(Ｓ１３１６)、DataIDをつけてデータをHDDのバイナリログファイルに保存する(Ｓ１３１７)。一方、正しい領域でないと判定された場合には、バイナリ保存処理をそのまま終了し、対応するデータのバイナリログファイルとしての保存を行わない。パラメータがメモリアドレスとして定義されていない場合には、データ保存処理そのものを行わない。

【００３８】

図１３（a）に戻ると、関数内部の処理が終了すると(Ｓ１３０６)、本ソフトウェアはDLL名/関数名/終了時の時刻をHDDに保存し(Ｓ１３０７)、その呼び出しに対してのパラメータ及び戻り値をHDDに保存する(Ｓ１３０８)。次にパラメータがメモリアドレスとして定義されているかどうかを、図１１に示した関数定義ファイルに基づいて判定し(Ｓ１３０９)、定義されている場合はバイナリ保存処理(Ｓ１３１０)を行なう。バイナリ保存処理は上

述した図１３（ｂ）に示す処理と同様である。異常の処理は、評価対象となるプログラムが終了するまで(Ｓ１３１１)続行される。

【００３９】

以上のように、本実施形態に対応する本発明によれば、データを書き込むべきメモリアドレスとデータサイズに基づいてデータの書込領域が正しい領域であるかどうかを事前に問い合わせ、正しい領域との判定の上で書込を行うので、誤って不正領域にデータが書き込まれることがなくなり、不正領域にデータが書き込まれた場合に生ずるはずの例外処理を未然に防止することが可能となる。

【００４０】

［第２の実施形態］

上記の第１の実施形態では、保存するバイナリ領域全体として正しい領域でなければバイナリデータの保存処理を行わせないようにしていたが、本実施形態では、バイナリ領域に不正な領域が一部存在していた場合であっても、不正なメモリアドレスが存在する直前までのバイナリデータを保存可能とする。

【００４１】

以下、本発明の情報処理装置において実行される処理について説明する。図１４は、本実施形態におけるバイナリログの保存にかかわる処理の一例に対応するフローチャートである。

【００４２】

図１４（ａ）において、設定された関数/メソッドが呼び出されることによって、処理が開始されると(Ｓ１４０１)、本ソフトウェアはDLL名/関数名/呼び出し時の時刻をHDDに保存し(Ｓ１４０２)、その呼び出しに対してのパラメータをHDDに保存する(Ｓ１４０３)。次に本ソフトウェアは、パラメータがメモリアドレスとして定義されているかどうかを、図１１に示した関数定義ファイルに基づいて判断し(Ｓ１４０４)、定義されている場合はバイナリ保存処理(Ｓ１４０５)を行なう。

【００４３】

ここでバイナリ保存処理の詳細は、図１４（ｂ）に示す通りであり、ここにおいて関数定義のサイズをあらわす引数からデータサイズを取得し(Ｓ１４１４)、データサイズ分のループ処理(Ｓ１４１５乃至Ｓ１４１８)でデータ領域を、１バイト単位に正しい領域に書き込まれるか否かの判定を行なう。取得を行なうメモリアドレスの先頭からnバイト目の１バイト分のメモリ領域が正しい領域であるか否かをＯＳへ問い合わせる(Ｓ１４１７)。正しい領域であると判定された場合のみ、次のデータ領域の判断を行なうため、nをインクリメントする(Ｓ１４１７))。不正な領域であると判定された場合には、そこ時点でデータ領域の判定を終了する。

【００４４】

Ｓ１４１５～Ｓ１４１８までのループ処理が終了すると、正しい領域のバイト数はｎとなるので、ｎが０であるか否かにより正しい領域があるか否かを判定し(Ｓ１４１９)、正しい領域があったと判定された場合には、そのサイズ分である、ｎバイトのデータをメモリ領域の先頭から読み込んで(Ｓ１４２０)、DataIDをつけてデータをHDDのバイナリログファイルに保存する(Ｓ１４２１)。一方、ｎが０であり、全領域が誤りであると判定された場合には、データの保存を行わずにバイナリ保存処理を終了する。また、パラメータがメモリアドレスとして定義されていない場合には、データ保存処理そのものを行わない。

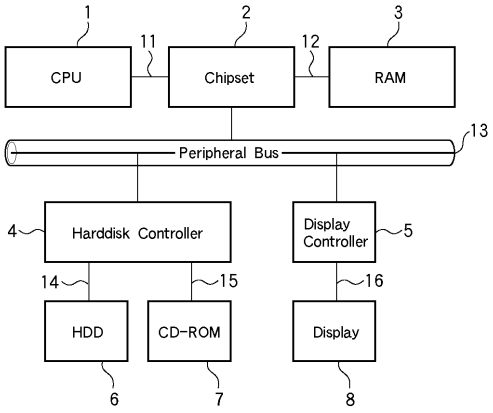
【００４５】

関数内部の処理が終了すると(Ｓ１４０６)、本ソフトウェアはDLL名/関数名/終了時の時刻をHDDに保存し(Ｓ１４０７)、その呼び出しに対してのパラメータ及び戻り値をHDDに保存する(Ｓ１４０８)。次に、パラメータがメモリアドレスとして定義されているかどうかを、図１１に示した関数定義ファイルに基づいて判断し(Ｓ１４０９)、定義されている場合はバイナリ保存処理を行なう。バイナリ保存処理は、上述の図１４（ｂ）における処理と同様である。以上の処理は、評価対象となるプログラムが終了するまで(Ｓ１４１１)続行される。

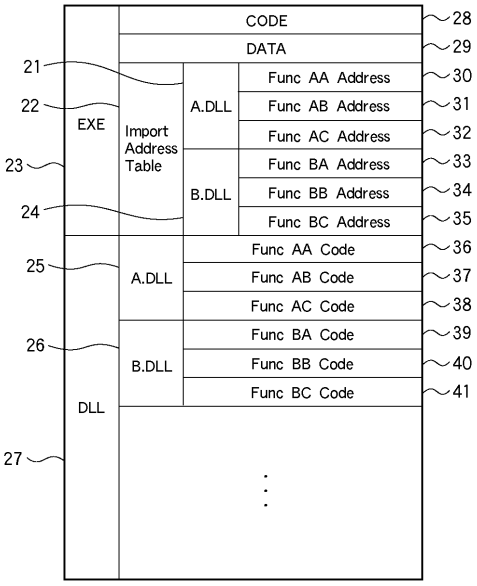
【００４６】
以上によれば、保存するバイナリ領域に不正な領域が一部存在していた場合であっても、不正なメモリアドレスが存在する直前までのバイナリデータを保存することができる。
【００４７】
［その他の実施形態］
なお、本発明は、複数の機器（例えばホストコンピュータ、インタフェイス機器、リーダ、プリンタなど）から構成されるシステムに適用しても、一つの機器からなる装置（例えば、複写機、ファクシミリ装置など）に適用してもよい。
【００４８】
また、本発明の目的は、前述した実施形態の機能を実現するソフトウェアのプログラムコードを記録した記憶媒体（または記録媒体）を、システムあるいは装置に供給し、そのシステムあるいは装置のコンピュータ（またはＣＰＵやＭＰＵ）が記憶媒体に格納されたプログラムコードを読み出し実行することによっても、達成されることは言うまでもない。この場合、記憶媒体から読み出されたプログラムコード自体が前述した実施形態の機能を実現することになり、そのプログラムコードを記憶した記憶媒体は本発明を構成することになる。また、コンピュータが読み出したプログラムコードを実行することにより、前述した実施形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼働しているオペレーティングシステム（ＯＳ）などが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。
【００４９】
さらに、記憶媒体から読み出されたプログラムコードが、コンピュータに挿入された機能拡張カードやコンピュータに接続された機能拡張ユニットに備わるメモリに書込まれた後、そのプログラムコードの指示に基づき、その機能拡張カードや機能拡張ユニットに備わるＣＰＵなどが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。
【図面の簡単な説明】
【００５０】
【図１】本発明の実施形態に対応する情報処理装置の構成の一例を示す図である。
【図２】本発明の実施形態に対応する情報処理装置のメモリに、複数のモジュールに分割されたソフトウェアがロードされる場合を説明する図である。
【図３】本発明の実施形態に対応する、ログ取得用のコードとしてIAT Patchを用いて関数呼び出しを仲介する場合の、情報処理装置のメモリ構成の一例を示す図である。
【図４】本発明の実施形態に対応する情報処理装置において、IAT Patchの処理を実行した場合のタイミングチャートの一例を示す図である。
【図５】本発明の実施形態に対応する情報処理装置において実行形式ファイルＥＸＥが実行される場合の動作の一例を示す図である。
【図６】本発明の実施形態に対応する情報処理装置において、実行ファイルEXEがCOMサーバでエクスポートされているインターフェースのインスタンスを作成する場合の、メモリ構成の一例を示す図である。
【図７】本発明の実施形態に対応する情報処理装置のメモリ構成をあらわす図である。
【図８】本発明の実施形態に対応する情報処理装置において、VTable Patchの処理を実行した場合のタイミングチャートの一例を示す図である。
【図９】本発明の実施形態に対応する情報処理装置において実行形式ファイルＥＸＥが実行される場合の動作の一例を示す図である。
【図１０】本発明の実施形態に対応する情報処理装置において利用される、関数及びメソッドのパラメータの形式や、戻り値の形式を指示する関数定義ファイルの一例を示す図である。
【図１１】図１０に示した関数定義ファイルを用いて、本発明の実施形態に対応する情報処理装置において取得されたログの一例を示す図である。

【図１２】図１１に示したログと共に保存されるバイナリログファイルの実体を一例として示す図である。
【図１３】本発明の第１の実施形態に対応する処理の一例のフローチャートである。
【図１４】本発明の第２の実施形態に対応する処理の一例のフローチャートである。

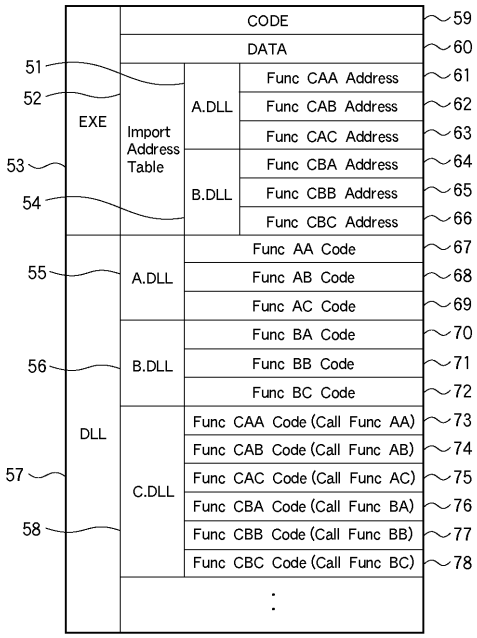
【図１】



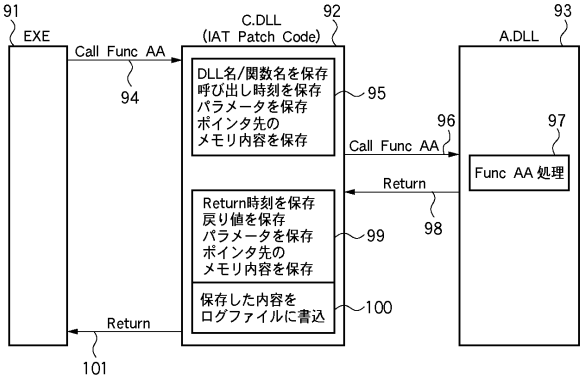
【図２】



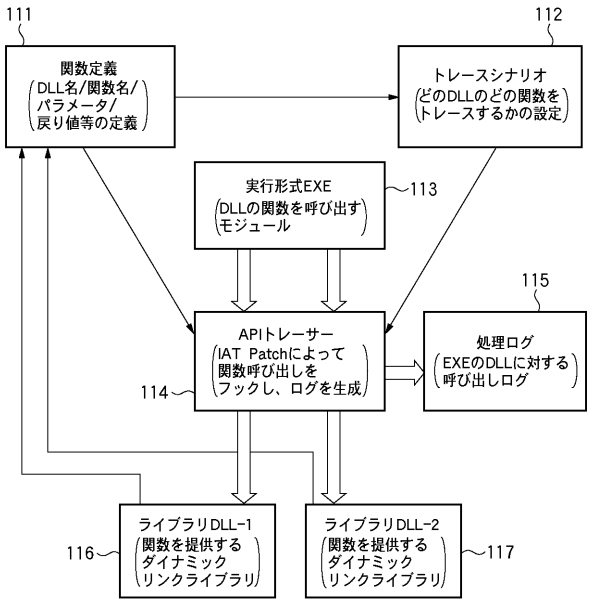
【図３】


CODE
DATA
EXE
Import Address Table
A.DLL
B.DLL
Func CAA Address
Func CAB Address
Func CAC Address
Func CBA Address
Func CBB Address
Func CBC Address
DLL
Func AA Code
Func AB Code
Func AC Code
Func BA Code
Func BB Code
Func BC Code
C.DLL
Func CAA Code (Call Func AA)
Func CAB Code (Call Func AB)
Func CAC Code (Call Func AC)
Func CBA Code (Call Func BA)
Func CBB Code (Call Func BB)
Func CBC Code (Call Func BC)
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78

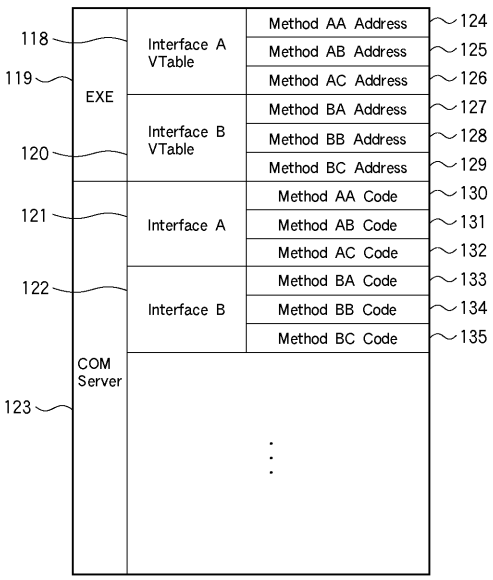
【図４】


91
EXE
Call Func AA
94
C.DLL
(IAT Patch Code)
92
DLL名/関数名を保存
呼び出し時刻を保存
パラメータを保存
ポインタ先の
メモリ内容を保存
95
Return時刻を保存
戻り値を保存
パラメータを保存
ポインタ先の
メモリ内容を保存
99
保存した内容を
ログファイルに書込
100
Return
101
96
Call Func AA
Return
98
93
A.DLL
97
Func AA 処理

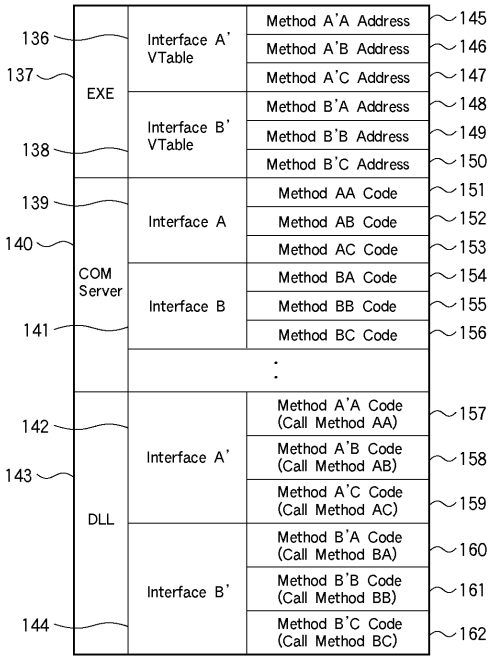
【図５】


111
関数定義
(DLL名/関数名/
パラメータ/
戻り値等の定義)
112
トレースシナリオ
(どのDLLのどの関数を
トレースするかの設定)
実行形式EXE
(DLLの関数を呼び出す
モジュール)
113
APIトレーサー
(IAT Patchによって
関数呼び出しを
フックし、ログを生成)
114
115
処理ログ
(EXEのDLLに対する
呼び出しログ)
ライブラリDLL-1
(関数を提供する
ダイナミック
リンクライブラリ)
116
ライブラリDLL-2
(関数を提供する
ダイナミック
リンクライブラリ)
117

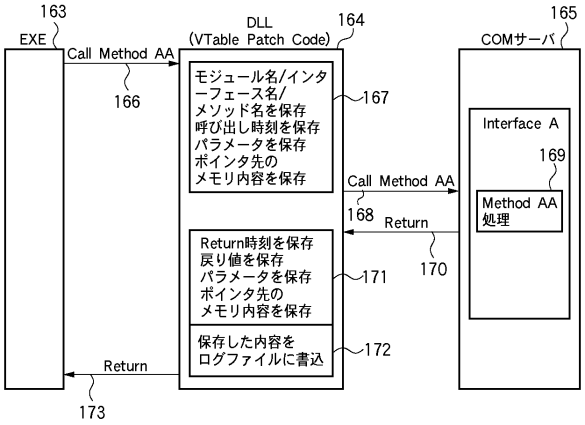
【図６】


EXE
COM Server
Interface A VTable
Interface B VTable
Interface A
Interface B
Method AA Address
Method AB Address
Method AC Address
Method BA Address
Method BB Address
Method BC Address
Method AA Code
Method AB Code
Method AC Code
Method BA Code
Method BB Code
Method BC Code
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135

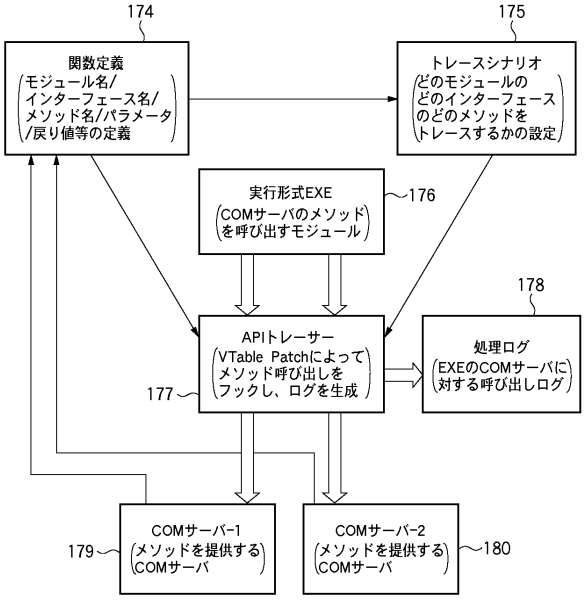
【図７】



【図８】



【図９】



【図１０】

```
DLL名：  A. DLL
関数名：  FuncAA
引数：    DWORD dwID
戻り値：  DWORD dwRet

DLL名：  A. DLL
関数名：  FuncAB
引数：    DWORD dwBufSize
          char ＊pBuf (Size=dwBufSize)
戻り値：  int nRet

DLL名：  B. DLL
関数名：  FuncBA
引数：    DWORD dwID
戻り値：  DWORD dwRet

DLL名：  B. DLL
関数名：  FuncBB
引数：    DWORD dwHandle
戻り値：  int nRet
```

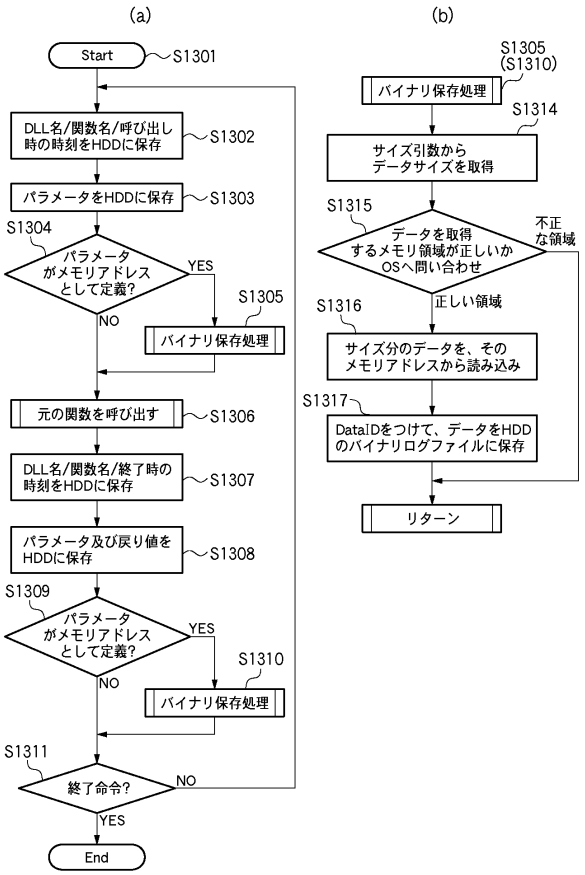
【図１１】

```
DLL名 :    A. DLL
関数名 :    FuncAA
引数 :      DWORD  dwID : 256
戻り値 :    DWORD  dwRet : 0
In時刻 :    2002/03/25  22 : 24 : 12. 025
Out時刻 :   2002/03/25  22 : 24 : 12. 035

DLL名 :    A. DLL
関数名 :    FuncAB
引数 :      DWORD  dwBufSize : 24
            char  ＊pBuf : 0x5034206D
                  Size=dwBufSize,  DataID=0x0001
戻り値 :    int  nRet : 0
In時刻 :    2002/03/25  22 : 24 : 12. 046
Out時刻 :   2002/03/25  22 : 24 : 12. 057

DLL名 :    A. DLL
関数名 :    FuncAB
引数 :      DWORD  dwBufSize : 24
            char  ＊pBuf : 0x503860C
                  Size=dwBufSize,  DataID=0x0002
戻り値 :    int  nRet : 0
In時刻 :    2002/03/25  22 : 24 : 12. 068
Out時刻 :   2002/03/25  22 : 24 : 12. 079

…
```

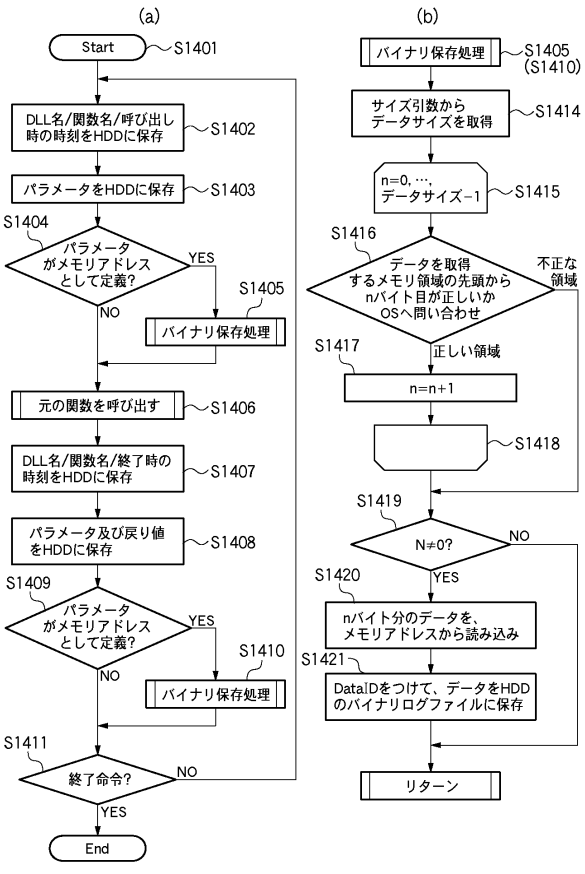
【図１２】

```
DataID :   0x0001

00000000 :   01  5D  33  B2  20  49  20  39
00000008 :   02  5D  33  B2  20  49  30  39
00000010 :   01  6D  33  B2  20  59  20  39

DataID :   0x0002

00000000 :   01  5D  44  B2  20  49  20  39
00000008 :   02  5D  44  B2  20  49  30  39
00000010 :   01  6D  44  B2  20  59  20  39

DataID :   0x0003

00000000 :   01  5D  55  B2  20  49  20  39
00000008 :   02  5D  55  B2  20  49  30  39
00000010 :   01  6D  55  B2  20  59  20  39

DataID :   0x0004

00000000 :   01  5D  66  B2  20  49  20  39
00000008 :   02  5D  66  B2  20  49  30  39
00000010 :   01  6D  66  B2  20  59  20  39

…
```

【図１３】



【図１４】



フロントページの続き

(72)発明者　三原　誠
東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内

審査官　多胡　滋

(56)参考文献　特開平０９－０８１４１１（ＪＰ，Ａ）
特開２００４－３１８５２４（ＪＰ，Ａ）
特開２００４－０３８３１４（ＪＰ，Ａ）
特開２００４－０２１４２９（ＪＰ，Ａ）
矢野越夫，Microsoft Visual C++ 6.0，Ｉｎｔｅｒｆａｃｅ，日本，ＣＱ出版株式会社，１９９８年１２月　１日，第24巻、第12号，pp.146-153

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１１／２８
Ｇ０６Ｆ　１１／３６