(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第5361358号
(P5361358)

(45)発行日　平成25年12月4日(2013.12.4)　(24)登録日　平成25年9月13日(2013.9.13)

(51)Int.Cl.　　F I
G06F 9/54 (2006.01)　　G06F 9/46 480A
G06F 3/12 (2006.01)　　G06F 9/06 640C
　　G06F 3/12 C

請求項の数 17　(全 20 頁)

(21)出願番号　特願2008-316274 (P2008-316274)
(22)出願日　平成20年12月11日 (2008.12.11)
(65)公開番号　特開2010-140281 (P2010-140281A)
(43)公開日　平成22年6月24日 (2010.6.24)
審査請求日　平成23年12月7日 (2011.12.7)

(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】情報処理装置およびその制御方法、並びにプログラム

(57)【特許請求の範囲】
【請求項１】
第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリを作成する作成手段と、
前記作成手段が作成する前記名前付き共有メモリに情報を格納する格納手段と、
前記作成手段が作成する前記名前付き共有メモリから前記格納手段が格納する前記情報を取得する取得手段と、
を有し、
前記格納手段は、前記プリンタドライバによって実行され、
前記取得手段は、前記アプリケーションによって実行されることを特徴とする情報処理装置。
【請求項２】
前記格納手段は、前記アプリケーションが関数を呼び出した場合に、前記作成手段が作成する前記名前付き共有メモリに情報を格納し、
前記取得手段は、前記アプリケーションが前記関数の返却を受け取った場合に、前記作成手段が作成する前記名前付き共有メモリから前記格納手段が格納する前記情報を取得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３】
前記名前付き共有メモリを特定する特定情報を前記アプリケーションから取得する第２

の取得手段を更に有し、
前記第２の取得手段は、前記プリンタドライバによって実行され、
前記格納手段は、前記第２の取得手段が取得する前記特定情報に基づいて前記名前付き共有メモリを特定して、前記作成手段が作成する前記名前付き共有メモリに情報を格納することを特徴とする請求項１または２に記載の情報処理装置。
【請求項４】
前記格納手段は、オペレーティングシステムが用意するアプリケーションプログラミングインタフェースでは取得できない情報を、前記作成手段が作成する前記名前付き共有メモリに格納することを特徴とする請求項１乃至３のいずれか１項に記載の情報処理装置。
【請求項５】
前記格納手段は、前記作成手段が作成する前記名前付き共有メモリに前記情報と前記情報の生成に成功したか又は失敗したかを示す生成成否情報とを格納し、
前記取得手段は、前記格納手段が格納する前記生成成否情報が前記情報の生成に成功したことを示す場合に、前記作成手段が作成する前記名前付き共有メモリから前記格納手段が格納する前記情報を取得することを特徴とする請求項１乃至４のいずれか１項に記載の情報処理装置。
【請求項６】
前記オペレーティングシステムが用意する前記アプリケーションプログラミングインタフェースでは取得できない前記情報とは、縁無し印刷のはみ出し量であることを特徴とする請求項４に記載の情報処理装置。
【請求項７】
第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリを作成する作成手段と、
前記作成手段が作成する前記名前付き共有メモリから情報を取得する取得手段と、
を有し、
前記情報は、前記作成手段が作成する前記名前付き共有メモリに前記プリンタドライバによって格納され、
前記取得手段は、前記アプリケーションによって実行されることを特徴とする情報処理装置。
【請求項８】
第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリに情報を格納する格納手段を有し、
前記名前付き共有メモリは、前記アプリケーションによって作成され、
前記格納手段は、前記プリンタドライバによって実行され、
前記アプリケーションは、前記名前付き共有メモリから前記格納手段が格納する前記情報を取得することを特徴とする情報処理装置。
【請求項９】
コンピューターを、
第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリを作成する作成手段、
前記作成手段が作成する前記名前付き共有メモリに情報を格納する格納手段、
前記作成手段が作成する前記名前付き共有メモリから前記格納手段が格納する前記情報を取得する取得手段、
として機能させ、
前記格納手段は、前記プリンタドライバによって実行され、
前記取得手段は、前記アプリケーションによって実行されることを特徴とするプログラム。

【請求項１０】
前記格納手段は、前記アプリケーションが関数を呼び出した場合に、前記作成手段が作成する前記名前付き共有メモリに情報を格納し、
前記取得手段は、前記アプリケーションが前記関数の返却を受け取った場合に、前記作成手段が作成する前記名前付き共有メモリから前記格納手段が格納する前記情報を取得することを特徴とする請求項９に記載のプログラム。
【請求項１１】
コンピューターを、
前記名前付き共有メモリを特定する特定情報を前記アプリケーションから取得する第２の取得手段として更に機能させ、
前記第２の取得手段は、前記プリンタドライバによって実行され、
前記格納手段は、前記第２の取得手段が取得する前記特定情報に基づいて前記名前付き共有メモリを特定して、前記作成手段が作成する前記名前付き共有メモリに情報を格納することを特徴とする請求項９または１０に記載のプログラム。
【請求項１２】
前記格納手段は、オペレーティングシステムが用意するアプリケーションプログラミングインタフェースでは取得できない情報を、前記作成手段が作成する前記名前付き共有メモリに格納することを特徴とする請求項９乃至１１のいずれか１項に記載のプログラム。
【請求項１３】
前記格納手段は、前記作成手段が作成する前記名前付き共有メモリに前記情報と前記情報の生成に成功したか又は失敗したかを示す生成成否情報とを格納し、
前記取得手段は、前記格納手段が格納する前記生成成否情報が前記情報の生成に成功したことを示す場合に、前記作成手段が作成する前記名前付き共有メモリから前記格納手段が格納する前記情報を取得することを特徴とする請求項９乃至１２のいずれか１項に記載のプログラム。
【請求項１４】
前記オペレーティングシステムが用意する前記アプリケーションプログラミングインタフェースでは取得できない前記情報とは、縁無し印刷のはみ出し量であることを特徴とする請求項１２に記載のプログラム。
【請求項１５】
コンピューターを、
第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリを作成する作成手段、
前記作成手段が作成する前記名前付き共有メモリから情報を取得する取得手段、
として機能させ、
前記情報は、前記作成手段が作成する前記名前付き共有メモリに前記プリンタドライバによって格納され、
前記取得手段は、前記アプリケーションによって実行されることを特徴とするプログラム。
【請求項１６】
コンピューターを、
第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリに情報を格納する格納手段として機能させ、
前記名前付き共有メモリは、前記アプリケーションによって作成され、
前記格納手段は、前記プリンタドライバによって実行され、
前記アプリケーションは、前記名前付き共有メモリから前記格納手段が格納する前記情報を取得することを特徴とするプログラム。
【請求項１７】

第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリを作成する作成工程と、
前記作成工程にて作成する前記名前付き共有メモリに情報を格納する格納工程と、
前記作成工程にて作成する前記名前付き共有メモリから前記格納工程にて格納する前記情報を取得する取得工程と、
を有し、
前記格納工程は、前記プリンタドライバによって実行され、
前記取得工程は、前記アプリケーションによって実行されることを特徴とする情報処理装置の制御方法。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、オペレーティングシステムが提供するＡＰＩ（Ａｐｐｌｉｃａｔｉｏｎ Ｐｒｏｇｒａｍｍｉｎｇ Ｉｎｔｅｒｆａｃｅ）を呼び出すアプリケーションプログラムと、オペレーティングシステムにより呼び出されるプリンタドライバとの間のデータ通信方法、及びプログラムに関するものである。
【背景技術】
【０００２】
印刷を行うアプリケーションプログラムは、一般的にオペレーティングシステム（以下ＯＳと略す）が提供するＡＰＩを介して、印刷にかかわる様々な情報を取得し、また、印刷設定を通知する。
【０００３】
アプリケーションプログラムはプリンタの性能を取得するために、ＯＳが提供するＡＰＩの関数を呼び出す。ＯＳはプリンタの性能をプリンタドライバに問い合わせ、アプリケーションプログラムに返却する。アプリケーションプログラムは返却されたプリンタの性能情報を利用し、印刷時に使用する印刷設定を決定する。そして、アプリケーションプログラムはＯＳが提供するＡＰＩを呼び出すことで印刷設定をＯＳやプリンタドライバに通知する。
【０００４】
例えば取り扱うことのできる用紙の種類はプリンタの性能に依存している。そこで、アプリケーションプログラムは、印刷を行うプリンタで取り扱うことができる用紙の種類の一覧を取得する。印刷に使用する用紙の種類はユーザあるいはアプリケーションプログラムが指定する。ユーザに指定させる場合は、アプリケーションプログラムは取得した用紙の種類の一覧を画面に表示し、印刷に使用する用紙の種類をユーザに選択させる。印刷時に用いるためにユーザから選択された設定項目の選択値の集合は印刷設定と呼ばれ、アプリケーションプログラムはＯＳが提供するＡＰＩを呼び出すことで、ＯＳやプリンタドライバに通知することができる。一般的な設定項目には用紙の種類の他に、用紙のサイズや給紙方法、両面印刷、印刷の向き、部数等が挙げられる。
【０００５】
マイクロソフト社のＯＳであるＷｉｎｄｏｗｓ（登録商標）の場合、プリンタが取り扱うことができる用紙の種類等の一般的な設定項目に関する情報は、ＤｅｖｉｃｅＣａｐａｂｉｌｉｔｉｅｓ関数というＡＰＩを介して取得できる。用紙の種類ＩＤの一覧等における設定項目の情報種別ごとに本関数を呼び出すことで、アプリケーションプログラムは種別ごとの設定値の一覧を取得することが出来る。また、印刷で使用する印刷設定の情報は、ＣｒｅａｔｅＤＣ関数やＲｅｓｅｔＤＣ関数というＡＰＩを呼び出すことで、アプリケーションプログラムからＯＳやプリンタドライバに通知出来る。
【０００６】
一方、プリンタ固有の機能に関する情報は、ＯＳが提供する既定のＡＰＩ関数では取得出来ないことが多い。例えば、写真印刷で頻繁に用いられる縁無し印刷では、画像を用紙

のサイズより大きい領域に印刷する必要が有る為、用紙のサイズからのはみ出し量を指定出来ることが一般的である。これは、ユーザが用紙を斜めにプリンタにセットしたり、あるいは用紙が正しくセットされてもプリンタが用紙を搬送する間に斜めになったりすること想定しているためである。従って、縁無し印刷のはみ出し量の設定値はプリンタにより異なり、はみ出し量が大・中・小の三段階に設定できるプリンタもあれば、はみ出し量が小のみの設定しかできないプリンタも存在する。以下、縁無し印刷のはみ出し量と同様にＯＳが提供する既定のＡＰＩ関数では取得出来ない情報を、ベンダー独自情報と呼ぶ。

【０００７】

Ｗｉｎｄｏｗｓにおいてアプリケーションプログラムがベンダー独自情報を取得するには、プリンタドライバのモジュールが提供するベンダー独自のＡＰＩを利用する手法がある。また、アプリケーションプログラムはＯＳが提供するＥｘｔＥｓｃａｐｅ関数というＡＰＩを利用し、プリンタドライバからベンダー独自情報を取得する手法が有る。

【０００８】

まず、ベンダー独自のＡＰＩを利用する手法について説明する。ベンダー独自のＡＰＩは、プリンタドライバを構成するモジュールにより提供される。このモジュールはプリンタドライバＳＤＫモジュールと呼ばれることがある。アプリケーションプログラムは、まず、プリンタドライバＳＤＫモジュールを特定する。次にＬｏａｄＬｉｂｒａｒｙ関数を呼び出すことで、プリンタドライバＳＤＫモジュールを、アプリケーションプログラムのプロセスのアドレス空間にマップする。そして、プリンタドライバＳＤＫモジュールが提供するベンダー独自のＡＰＩの関数を呼び出すことで、ベンダー独自情報を取得することができる。（例えば、特許文献１参照）。

【０００９】

次にＥｘｔＥｓｃａｐｅ関数を用いる手法について説明する。ＥｘｔＥｓｃａｐｅ関数はＯＳが提供するＡＰＩであり、アプリケーションプログラムとプリンタドライバとの間で独自のデータを受け渡すために使用される。例えば、プリンタドライバが独自の画像処理を行う為のインタフェースを備えているかどうかをアプリケーションプログラムが確認する際にＥｘｔＥｓｃａｐｅ関数が使用されている（例えば、特許文献２参照）。

【００１０】

プリンタには数多くの種類があり、その種類ごとに能力が異なるため、ＯＳの種類ごとにプリンタドライバが必要となる。Ｗｉｎｄｏｗｓの場合、多数のプリンタドライバがＯＳの製品に同梱されている。また、ＯＳの製品に同梱されていない場合でもネットワーク経由で適切なプリンタドライバがマイクロソフト社から自動的にダウンロードされる仕組みがＯＳに備わっている。このように、Ｗｉｎｄｏｗｓでは多種多様のプリンタに対して印刷するための環境が提供されている。Ｗｉｎｄｏｗｓには、３２ｂｉｔ版ＯＳと６４ｂｉｔ版ＯＳが存在する。プリンタドライバはＯＳと密接に動作する為、３２ｂｉｔ版Ｗｉｎｄｏｗｓ製品に同梱されるプリンタドライバは３２ｂｉｔであり、６４ｂｉｔ版Ｗｉｎｄｏｗｓ製品に同梱されるプリンタドライバは６４ｂｉｔとなっている。自動的にダウンロードされるプリンタドライバも同様であり、６４ｂｉｔ版ＯＳであれば６４ｂｉｔプリンタドライバが、３２ｂｉｔ版ＯＳであれば３２ｂｉｔプリンタドライバがダウンロードされる。

【００１１】

また、Ｗｉｎｄｏｗｓで動作するプリンタドライバにはＧＤＩプリンタドライバとＸＰＳプリンタドライバの二種がある。ＧＤＩとはＧｒａｐｈｉｃｓ Ｄｅｖｉｃｅ Ｉｎｔｅｒｆａｃｅの略で、アプリケーションプログラムに対し、出力装置に依存しない画一的な描画指示を行わせるためのプログラムで、Ｗｉｎｄｏｗｓにより提供されている。ＧＤＩを介して行われたアプリケーションプログラムの描画指示をプリンタドライバのレンダリングモジュールが解釈し、印刷処理を行うのがＧＤＩプリンタドライバの特徴である。

【００１２】

一方、ＸＰＳプリンタドライバの特徴の一つに、スプールファイルに格納されるデバイス非依存の印刷データの形式がＸＰＳであることが挙げられる。ＸＰＳとはＸＭＬ Ｐａ

ｐｅｒ Ｓｐｅｃｉｆｉｃａｔｉｏｎの略で、マイクロソフト社が提案しているオープン規格の電子文書フォーマットの一つである。ＧＤＩを介して行われたアプリケーションプログラムの描画指示は、マイクロソフト社が提供するＭＸＤＣ（Ｍｉｃｒｏｓｏｆｔ ＸＰＳ Ｄｏｃｕｍｅｎｔ Ｃｏｎｖｅｒｔｅｒ）によりＸＰＳに変換され、スプールファイルに格納される。ＸＰＳプリンタドライバのレンダリングフィルタがＸＰＳを解釈して、印刷処理が行われる。ここで、ＸＰＳプリンタドライバのＭＸＤＣは、ＧＤＩプリンタドライバのレンダリングモジュールに相当する。

【００１３】

図２はベンダー独自のＡＰＩを利用する手法における、アプリケーションプログラムとプリンタドライバの関係を示す模式図の一例である。アプリケーションプログラム２０１がベンダー独自ＡＰＩ関数２１３を利用する場合、前述した様にベンダー独自ＡＰＩ提供モジュール２１１を、アプリケーションプログラムのプロセスのアドレス空間２０３にマップする。このためアプリケーションプログラム２０１は最初に、ベンダー独自ＡＰＩ提供モジュール２１１のモジュール名を特定する必要がある。例えば、アプリケーションプログラム２０１はＯＳが提供するＡＰＩであるＧｅｔＰｒｉｎｔｅｒＤｒｉｖｅｒ関数を呼び出すことで、プリンタドライバを構成するモジュールのうち、ユーザインタフェースモジュールと、レンダリングモジュールのファイル名を取得することができる。従って、ユーザインタフェースモジュールあるいはレンダリングモジュールがベンダー独自ＡＰＩ関数２１３を内包することで、アプリケーションプログラム２０１は、ベンダー独自ＡＰＩ提供モジュール２１１のモジュール名を特定することができる。また、ユーザインタフェースモジュールとレンダリングモジュール以外のモジュールでベンダー独自ＡＰＩを提供する場合は、後述するＥｘｔＥｓｃａｐｅ関数を用いてベンダー独自ＡＰＩ提供モジュール２１１のモジュール名を特定するようにしてもよい。アプリケーションプログラムのプロセスのアドレス空間２０３にベンダー独自ＡＰＩ提供モジュール２１１をマップした後、アプリケーションプログラム２０１はベンダー独自ＡＰＩ関数２１３のアドレスを取得し、呼び出しを行う。

【００１４】

図３（Ａ）はＯＳが提供するＡＰＩであるＥｘｔＥｓｃａｐｅ関数の動作を示す模式図の一例である。ＥｘｔＥｓｃａｐｅ関数を利用することにより、アプリケーションプログラム２０１とプリンタドライバはベンダー独自情報の受け渡しを行うことができる。ＯＳの印刷サポート機能３０１はＯＳにより提供される機能であり、アプリケーションプログラム２０１が呼び出したＡＰＩ関数の処理を行い、必要に応じてプリンタドライバを呼び出す機能を有する。ユーザインタフェースモジュール３１１とレンダリングモジュール３２１はプリンタドライバを構成するモジュール群に含まれる。ユーザインタフェースモジュール３１１は、印刷に関する各種項目の設定をユーザに行わせ、ユーザの指示に基づいて印刷設定を作成する機能を有する。レンダリングモジュール３２１は、ＧＤＩを介して受け取ったアプリケーションプログラム２０１の描画指示を、画像に変換したり、あるいは、ページ記述言語に変換したりする機能を有する。

【００１５】

アプリケーションプログラム２０１は、例えば縁無し印刷におけるはみ出し量の一覧等にて所望するベンダー独自情報を得る為に、ＥｘｔＥｓｃａｐｅ関数の引数に、ベンダー独自のＥｓｃａｐｅコードを指定する（Ｓ３１１）。また、同時に、後述するＥｓｃａｐｅ処理に必要なデータをＥｘｔＥｓｃａｐｅ関数の引数で渡す。アプリケーションプログラム２０１がＥｘｔＥｓｃａｐｅ関数を呼び出すと、ＯＳの印刷サポート機能３０１は、プリンタドライバのユーザインタフェースモジュール３１１のＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数を呼び出す（Ｓ３１２）。ＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数は、アプリケーションプログラム２０１がＥｘｔＥｓｃａｐｅ関数を呼び出した時の引数を参照することができる。Ｅｓｃａｐｅイベントが通知され、ＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数の処理が終了すると（Ｓ３１３）、ＯＳの印刷サポート機能３０１はプリンタドライバのレンダリングモジュール３２１のＤｒｖＥｓｃａｐｅ関数を呼び出す（Ｓ３１４）。

ＤｒｖＥｓｃａｐｅ関数は、アプリケーションプログラム２０１がＥｘｔＥｓｃａｐｅ関数を呼び出した時の引数を参照することができる。ＤｒｖＥｓｃａｐｅ関数は引数に応じた処理を行い、ＥｘｔＥｓｃａｐｅ関数の返却データを生成することができる。ここで返却データはベンダー独自情報を含む様にする。ＤｒｖＥｓｃａｐｅ関数により生成された返却データは、ＯＳの印刷サポート機能３０１を介してアプリケーションプログラム２０１に返却される（Ｓ３１５及びＳ３１６）。

【００１６】

ここで、３２ｂｉｔ版Ｗｉｎｄｏｗｓ上の３２ｂｉｔアプリケーションプログラム、或いは６４ｂｉｔ版Ｗｉｎｄｏｗｓ上の６４ｂｉｔアプリケーションプログラムがＥｘｔＥｓｃａｐｅ関数を呼び出した場合、プリンタドライバは一般的にアプリケーションプログラムと同一プロセスで動作する。一方、６４ｂｉｔ版Ｗｉｎｄｏｗｓ上の３２ｂｉｔアプリケーションプログラムがＥｘｔＥｓｃａｐｅ関数を呼び出した場合、６４ｂｉｔプリンタドライバはアプリケーションプログラムの３２ｂｉｔプロセスとは異なる６４ｂｉｔプロセスで動作する。

【特許文献１】特開２００４－２８８０１３

【特許文献２】特開平１０－２４８０１４

【発明の開示】

【発明が解決しようとする課題】

【００１７】

しかしながら、従来の手法により、ベンダー独自情報をプリンタドライバからアプリケーションプログラムに返却する場合、下記のような問題があった。

【００１８】

まず、ＸＰＳプリンタドライバではＥｘｔＥｓｃａｐｅ関数の返却データを生成することが出来ない問題が有る。図３（Ｂ）はＸＰＳプリンタドライバにおけるＥｘｔＥｓｃａｐｅ関数の動作を示す模式図の一例である。アプリケーションプログラム２０１はベンダー独自情報を取得する為にベンダー独自のＥｓｃａｐｅコードを指定してＥｘｔＥｓｃａｐｅ関数を呼び出す（Ｓ３２１）。ＯＳの印刷サポート機能３０１は、最初にＸＰＳプリンタドライバのユーザインタフェースモジュール３３１のＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数を呼び出す（Ｓ３２２）。Ｅｓｃａｐｅイベントを通知し、ＸＰＳプリンタドライバのユーザインタフェースモジュール３３１の処理が終了した後（Ｓ３２３）、ＯＳの印刷サポート機能３０１はＭＸＤＣ３４１のＤｒｖＥｓｃａｐｅ関数を呼び出す（Ｓ３２４）。前述した様にＭＸＤＣ３４１はマイクロソフト社により提供されているため、ベンダー独自のＥｓｃａｐｅコードは未知のＥｓｃａｐｅコードとして処理される。従って、ＥｘｔＥｓｃａｐｅ関数はベンダー独自のコードに対し、返却データを生成できないため、アプリケーションプログラム２０１はベンダー独自情報を取得出来ない問題が発生している（Ｓ３２５及びＳ３２６）。

【００１９】

次に、６４ｂｉｔ版Ｗｉｎｄｏｗｓ製品に同梱されるプリンタドライバは、ベンダー独自のＡＰＩを３２ｂｉｔアプリケーションプログラムに対して提供出来ない問題が有る。Ｗｉｎｄｏｗｓの仕様により６４ｂｉｔモジュールを３２ｂｉｔアプリケーションプログラムのプロセスのアドレス空間にマップできない。６４ｂｉｔ版Ｗｉｎｄｏｗｓでは６４ｂｉｔプリンタドライバが必要とされるが、３２ｂｉｔアプリケーションプログラムも６４ｂｉｔアプリケーションプログラムも動作する。従って６４ｂｉｔ版Ｗｉｎｄｏｗｓで動作する３２ｂｉｔアプリケーションプログラムに対して、６４ｂｉｔプリンタドライバがベンダー独自のＡＰＩを提供する為には、６４ｂｉｔプリンタドライバを構成するモジュールに３２ｂｉｔモジュールを少なくとも一つは含める必要が有る。しかしながら、前述した様に６４ｂｉｔ版Ｗｉｎｄｏｗｓ製品に同梱されるプリンタドライバは６４ｂｉｔであるため、３２ｂｉｔモジュールを含まない。従って、６４ｂｉｔ版Ｗｉｎｄｏｗｓ製品に同梱される６４ｂｉｔプリンタドライバでは、ベンダー独自のＡＰＩを３２ｂｉｔアプリケーションプログラムに対して提供出来ない問題が有る。

【００２０】
加えて、Ｗｉｎｄｏｗｓ製品に同梱されるプリンタドライバの場合、ユーザインタフェースモジュールとレンダリングモジュールではベンダー独自のＡＰＩを提供できない問題がある。これはＧｅｔＰｒｉｎｔｅｒＤｒｉｖｅｒ関数で取得できるユーザインタフェースモジュールとレンダリングモジュールはマイクロソフト社製のモジュールとなっており、他社の独自ＡＰＩを提供できないためである。さらに、Ｗｉｎｄｏｗｓ製品に同梱されるＸＰＳプリンタドライバの場合、前述した様にＥｘｔＥｓｃａｐｅ関数の返却データを生成できないため、アプリケーションプログラムはベンダー独自ＡＰＩ提供モジュールを特定出来ない問題がある。

【００２１】
そこで、本発明は上記の問題に鑑みて成されたものであり、オペレーティングシステム上で動作するアプリケーションプログラムとプリンタドライバとの間のデータ通信方法であり、ベンダー独自ＡＰＩを用いず、かつ、レンダリングモジュールに依存しないデータ通信方法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】

【００２２】
上記目的を達成するために本発明は次のような構成からなる。すなわち、情報処理装置であって、第１のビット数で動作するアプリケーションと、前記第１のビット数とは異なる第２のビット数で動作するプリンタドライバの両方がアクセスできる名前付き共有メモリを作成する作成手段と、前記作成手段が作成する前記名前付き共有メモリに情報を格納する格納手段と、前記作成手段が作成する前記名前付き共有メモリから前記格納手段が格納する前記情報を取得する取得手段と、を有し、前記格納手段は、前記プリンタドライバによって実行され、前記取得手段は、前記アプリケーションによって実行されることを特徴とする。

【発明の効果】

【００２３】
本発明によれば、プリンタドライバはベンダー独自情報を名前付きオブジェクトに格納し、アプリケーションプログラムは名前付きオブジェクトに格納された情報を参照することにより、ＯＳが提供するＡＰＩの仕様に制限されず処理を実行できるという効果を奏する。

【００２４】
さらに、アプリケーションプログラムは名前付き共有メモリ及びＥｘｔＥｓｃａｐｅ関数を利用するとしたため、ユーザインタフェースモジュールのイベント処理部あるいはレンダリングモジュールのＥｓｃａｐｅ処理部のどちらでも、ベンダー独自情報を名前付き共有メモリに格納することが可能になる。このことにより、ユーザインタフェースモジュールでの処理が可能になる為、ＸＰＳプリンタドライバでも、ベンダー独自情報をアプリケーションプログラムに返却することが出来るという効果を奏する。

【００２５】
さらに、名前付きオブジェクトを用いることで、動作条件の異なるアプリケーションプログラムとプリンタドライバとが別個のプロセスとして動作する場合にも、ベンダー独自情報をアプリケーションプログラムに返却することが出来るという効果を奏する。

【００２６】
さらに、６４ｂｉｔ版Ｗｉｎｄｏｗｓ製品に同梱されているプリンタドライバが、ベンダー独自情報を３２ｂｉｔ版アプリケーションプログラムに返却することができるという効果を奏する。

【００２７】
また、アプリケーションプログラムがＡＰＩを呼び出した時にのみプリンタドライバが動作する為、ベンダー独自情報を提供する為のプログラムをサービスや別プロセスとして待機させておく必要が無いという効果を奏する。

【発明を実施するための最良の形態】

【００２８】
以下、図面を参照して本発明の好適な実施の形態を詳しく説明する。尚、以下の実施の形態は特許請求の範囲に係る本発明を限定するものでなく、また本実施の形態で説明されている特徴の組み合わせの全てが本発明の解決手段に必須のものとは限らない。
【００２９】
＜実施形態１＞
まず、本実施形態におけるＸＰＳプリンタドライバの構成と、印刷動作について図５を用いて説明する。また、本実施形態を適用可能なハードウェア構成に関しては図１にて後述する。
【００３０】
＜印刷時のＸＰＳプリンタドライバの動作＞
図５はオペレーティングシステム上のアプリケーションプログラムが印刷処理をする際にＸＰＳプリンタドライバが動作した場合におけるデータの流れを説明する為のブロック図の一例である。図５において、ＸＰＳプリンタドライバを構成するモジュールはユーザインタフェースモジュール４１１、ＭＸＤＣ４２１、レイアウトフィルタ５２３、レンダーフィルタ５２５である。
【００３１】
アプリケーションプログラム２０１からＯＳの印刷サポート機能３０１を介してユーザインタフェースモジュール４１１を呼び出すことで、印刷に関する各種設定をユーザに行わせることができる。
【００３２】
アプリケーションプログラム２０１からユーザ操作により印刷指示がなされると、その指示内容はＯＳの印刷サポート機能３０１に送られる。具体的にはＧＤＩにより仮想化された印刷領域であるＤＣ（Ｄｅｖｉｃｅ　Ｃｏｎｔｅｘｔ）が作成され、アプリケーションプログラム２０１へ供給される。そして、アプリケーションプログラム２０１により文書データの描画処理がＤＣに対して行われる。描画内容はＭＸＤＣ４２１によりＸＰＳ形式に変換され、その結果、ＸＰＳ形式の印刷データが生成される。ＸＰＳ形式の印刷データは、ＯＳの印刷サポート機能３０１によって、印刷ジョブ毎にスプールファイル５０１として一時的に記憶装置にスプールされる。
【００３３】
ここでアプリケーションプログラム２０１はＧＤＩに描画を行わせることを前提に説明した。一方、ＸＰＳ形式の電子文書を作成することができるアプリケーションプログラムの場合は、ＧＤＩとＭＸＤＣ４２１に頼らずに、作成したＸＰＳ形式の印刷データをスプールすることも可能である。
【００３４】
スプールファイル５０１に格納されたデータはＯＳの印刷パイプラインサービス５１１により呼び出されたフィルタパイプライン５２１に供給される。フィルタパイプライン５２１は、図示しないＸＰＳプリンタドライバのパイプライン構成ファイルに記述されている任意の数のフィルタによって構成される。印刷装置１２０が解釈可能な印刷コマンドがＸＰＳ形式では無い場合、少なくとも、スプールされたＸＰＳ形式のデータを印刷装置が解釈可能な形式の印刷コマンドに変換する為のフィルタが必要となる。図５ではフィルタパイプライン５２１はレイアウトフィルタ５２３とレンダーフィルタ５２５で構成される。レイアウトフィルタ５２３はスプールファイルからＸＰＳ形式の印刷データを取得し、印刷設定に基づき必要に応じて印刷データを編集し出力する。印刷データを加工する必要が無い場合は、取得した印刷データをそのまま出力する。レイアウトフィルタ５２３の出力は、レンダーフィルタ５２５に供給される。レンダーフィルタ５２５は、供給されたＸＰＳ形式の印刷データにおける各ページのレンダリング処理を行い、必要な画像処理を行ったうえで、印刷コマンドに変換して出力する。レンダーフィルタ５２５が出力した印刷コマンドは印刷装置１２０に供給され、印刷装置１２０が印刷動作を実行する。
【００３５】

＜印刷設定ダイアログ＞

次に、ユーザインタフェースモジュール４１１がユーザに提供する印刷設定ダイアログについて説明する。図６は本実施形態における印刷設定をユーザが操作する際に表示される印刷設定ダイアログの一例を示す図である。印刷設定ダイアログは設定に関する内容を表示するとともに、ユーザによる上記設定内容の変更指示や入力を受け付ける。図６では、ＯＳが提供するＡＰＩでは取得出来ない、縁無し印刷のはみ出し量の情報に関する設定を例として示す。ここでは、前述したＤｅｖｉｃｅＣａｐａｂｉｌｉｔｉｅｓ関数で得られるプリンタが取り扱うことができる一般的な情報に関する設定項目は図示しない。

【００３６】

図６は縁無し印刷に関する印刷設定を行う為の縁無し設定ダイアログの表示例である。縁無し設定ダイアログ６０１は表示エリア６１１、６１３、６１５、６１７、６２１、６２３から構成される。縁無し設定部６２１ではチェックボックスをＯＮに設定することで縁無し印刷機能を有効にすることができる。縁無しはみ出し量設定部６２３では、縁無し印刷機能が有効な時に、用紙のサイズよりも印刷文書のページを大きく印刷する必要が有る為、印刷文書のページのはみ出し量をどの程度にするかを指定することが出来る。ここでは４段階の調整が可能として、右から、大、中、小、はみ出し量無し、となっている。

【００３７】

ユーザは各項目を選択した後にＯＫボタン６１１を押下することで縁無し設定ダイアログ６０１を閉じ、選択した印刷設定を印刷に反映させることができる。キャンセルボタン６１３を押下すると縁無し設定ダイアログ６０１が閉じ、選択した設定項目の内容は破棄され印刷に反映されることはない。標準に戻すボタン６１５が押下されると縁無し設定ダイアログ６０１上の各設定項目の設定値が標準状態に戻される。ヘルプボタン６１７は縁無し設定ダイアログ６０１の各設定項目に関する説明文を別ウィンドウで表示させることができる。

【００３８】

この様にしてユーザは、ユーザインタフェースモジュール４１１がユーザに提供する印刷設定ダイアログを介して、用紙のサイズなどの一般的な情報だけではなく、プリンタ固有の性能に関する印刷設定も行うことができる。

【００３９】

＜名前付き共有メモリとＥｘｔＥｓｃａｐｅ関数を用いた独自データの取得＞

次に、アプリケーションプログラム２０１がＥｘｔＥｓｃａｐｅ関数を用いて、ＸＰＳプリンタドライバからベンダー独自情報を取得する処理の流れについて説明する。まず名前付き共有メモリについて説明する。名前付き共有メモリは、ファイルマッピングオブジェクトの一種であり、ＯＳにより提供される名前付きオブジェクトの一つである。名前付き共有メモリの特徴として、複数のプロセスからアクセス可能である事が挙げられる。ここでの複数のプロセスというのは３２ｂｉｔプロセスや６４ｂｉｔプロセスを含み、３２ｂｉｔプロセスと６４ｂｉｔプロセスの両方からアクセスが可能であることを意味する。例えば、ＥｘｔＥｓｃａｐｅ関数を呼び出すアプリケーションプログラムのプロセスと、ＯＳの印刷サポート機能から呼び出されるプリンタドライバが動作するプロセスとがｂｉｔ数において一致しない場合でも、名前付き共有メモリを使用することにより各プロセスよりアクセスが可能になる。

【００４０】

図４は本実施形態におけるＸＰＳプリンタドライバにおけるＥｘｔＥｓｃａｐｅ関数の動作を示す模式図の一例である。アプリケーションプログラム２０１はＥｘｔＥｓｃａｐｅ関数を呼び出す前に名前付き共有メモリ４０１を作成し、取得したいベンダー独自情報の種類（例えば、設定項目）を格納する（Ｓ４０１）。続いてベンダー独自情報を取得する為にベンダー独自のＥｓｃａｐｅコードと、作成済みの名前付き共有メモリ４０１の名称とを指定してＥｘｔＥｓｃａｐｅ関数を呼び出す（Ｓ４０２）。ＯＳの印刷サポート機能３０１は、最初にＸＰＳプリンタドライバのユーザインタフェースモジュール３３１のＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数を呼び出す（Ｓ４０３）。ＸＰＳプリンタドライ

バのユーザインタフェースモジュール３３１はベンダー独自のＥｓｃａｐｅコードを判定し、ＥｘｔＥｓｃａｐｅ関数のイベントであれば、ベンダー独自情報の生成を行う。ＸＰＳプリンタドライバのユーザインタフェースモジュール３３１は、引数で渡された名前付き共有メモリ４０１の名称情報を元に、生成したベンダー独自情報を名前付き共有メモリ４０１に格納する（Ｓ４０４）。ＸＰＳプリンタドライバのユーザインタフェースモジュール３３１の処理が終了した後（Ｓ４０５）、ＯＳの印刷サポート機能３０１はＭＸＤＣ３４１のＤｒｖＥｓｃａｐｅ関数を呼び出す（Ｓ４０６）。前述した様にＭＸＤＣ３４１はマイクロソフト社により提供されているため、ベンダー独自のＥｓｃａｐｅコードは未知のＥｓｃａｐｅコードとして処理される。従って、ベンダー独自のＥｘｔＥｓｃａｐｅ関数の返却データを生成できないため、アプリケーションプログラム２０１はベンダー独自情報を取得出来ない（Ｓ４０７及びＳ４０８）。以上のようにＥｘｔＥｓｃａｐｅ関数の処理によってベンダー独自情報を取得できないため、アプリケーションプログラム２０１はＥｘｔＥｓｃａｐｅ関数の処理結果を無視し、名前付き共有メモリ４０１からベンダー独自情報を取得する（Ｓ４０９）。そして最後にアプリケーションプログラム２０１は名前付き共有メモリ４０１を削除する。

【００４１】

図７は名前付き共有メモリとＥｘｔＥｓｃａｐｅ関数を用いて、アプリケーションプログラムがベンダー独自情報を取得する処理のタイミングチャートの一例である。最初に、アプリケーションプログラム２０１が、名前付き共有メモリの名称を決定し、決定した名称を付与した名前付き共有メモリを作成する（７０１）。アプリケーションプログラム２０１のプロセス内において複数のベンダー独自情報を同時に取得する可能性があるため、名前付き共有メモリの名称は、プロセスＩＤとスレッドＩＤを利用して決定するのが望ましい。続いて、アプリケーションプログラム２０１は、ＥｘｔＥｓｃａｐｅ関数を呼び出す為の処理を行う（７０２）。まず、アプリケーションプログラム２０１は、取得したいベンダー独自情報の種類を名前付き共有メモリに格納する。そして、名前付き共有メモリの名称情報を引数に指定し、ベンダー独自のＥｓｃａｐｅコードでＯＳの印刷サポート機能３０１のＥｘｔＥｓｃａｐｅ関数を呼び出す。ＯＳの印刷サポート機能３０１はＥｘｔＥｓｃａｐｅ関数呼び出し（７０３）を解釈し、ユーザインタフェースモジュール４１１のＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数の呼び出し（７０５）を行う。

【００４２】

ＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数を呼び出されたユーザインタフェースモジュール４１１は、名前付き共有メモリへのベンダー独自情報の格納処理を行う（７０７）。処理の条件として、ＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数に通知されたイベントがＥｓｃａｐｅイベントで、かつベンダー独自のＥｓｃａｐｅコードが指定されている場合、ＥｘｔＥｓｃａｐｅ関数の引数で渡された名前付き共有メモリの名称情報に基づいて、名前付き共有メモリにアクセスする。ユーザインタフェースモジュール４１１は名前付き共有メモリに格納された、ベンダー独自情報の種類を読み出し、この種類に応じたベンダー独自情報を生成し、名前付き共有メモリに格納する。またこの時、ベンダー独自情報の生成に成功したか、或いは、失敗したかを示す生成成否情報も併せて名前付き共有メモリに格納する。以上でユーザインタフェースモジュール４１１はＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数の処理を終了し、ＯＳの印刷サポート機能３０１に処理の終了を通知する（７０９）。

【００４３】

ＯＳの印刷サポート機能３０１はユーザインタフェースモジュール４１１のＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数の処理が終了した後に、レンダリングモジュールであるＭＸＤＣ４２１のＤｒｖＥｓｃａｐｅ関数を呼び出す（７１１）。ＤｒｖＥｓｃａｐｅ関数を呼び出されたＭＸＤＣ４２１は、ベンダー独自Ｅｓｃａｐｅコードが指定されている場合は、その内容を解釈することができない（７１３）。従って、ＤｒｖＥｓｃａｐｅ関数は処理を実行できず、失敗した結果をＯＳの印刷サポート機能３０１に返却する（７１５）。ＯＳの印刷サポート機能３０１はＭＸＤＣ４２１のＤｒｖＥｓｃａｐｅ関数が失敗した

処理結果から、ＥｘｔＥｓｃａｐｅ関数が処理に失敗した結果をアプリケーションプログラム２０１に返却値として返却する（７１７）。

【００４４】

ＥｘｔＥｓｃａｐｅ関数の処理が終了した後、アプリケーションプログラム２０１は次の手順にてベンダー独自情報を取得する（７１９）。まず、ＥｘｔＥｓｃａｐｅ関数の返却値は常に処理失敗を意味することとなるため、アプリケーションプログラム２０１はこの返却値を無視する。続いてアプリケーションプログラム２０１は名前付き共有メモリに格納された生成成否情報を参照する。ユーザインタフェースモジュール４１１がベンダー独自情報の生成に成功していれば、アプリケーションプログラム２０１は名前付き共有メモリに格納されたベンダー独自情報を取得する。最後に、アプリケーションプログラム２０１は名前付き共有メモリを削除する（７２１）。

【００４５】

以上の様に、名前付き共有メモリとＯＳが提供するＥｘｔＥｓｃａｐｅ関数を用いることで、アプリケーションプログラム２０１はＸＰＳプリンタドライバからベンダー独自情報を取得することができる。従って、６４ｂｉｔ版Ｗｉｎｄｏｗｓ上で３２ｂｉｔアプリケーションプログラムが動作する環境においても、本手法を用いることによりアプリケーションプログラムのプロセスとは異なる６４ｂｉｔプロセスで動作するプリンタドライバがベンダー独自情報を提供することが出来る。

【００４６】

なお、複数の種類のベンダー独自情報を取得する場合、アプリケーションプログラム２０１は処理７０１の後に、必要に応じて処理７０２と処理７１９を複数回繰り返し、最後に処理７２１を行えばよいことは言うまでもない。

【００４７】

なお、ここではＥｘｔＥｓｃａｐｅ関数の引数として、名前付き共有メモリの名称情報を渡しているが、名称情報を生成する為の情報であってもよい。例えば名前付き共有メモリの名称は、前述した様にプロセスＩＤとスレッドＩＤを用いて生成するのが望ましい。従って、プロセスＩＤとスレッドＩＤをＥｘｔＥｓｃａｐｅ関数の引数に渡してもよい。その場合、ユーザインタフェースモジュール４１１は、引数として渡されたプロセスＩＤとスレッドＩＤを用いて名前付き共有メモリの名称情報を生成し、その名称情報を用いて名前付き共有メモリにアクセスする。

【００４８】

＜名前付き共有メモリに格納されるデータの構造＞

次に名前付き共有メモリに格納されるデータの構造について説明する。図８は本実施形態における名前付き共有メモリに格納されるデータの構造の一例を示す図である。名前付き共有メモリ８０１はメンバ８１１から８５１までの領域で構成される。

【００４９】

メンバ８１１は名前付き共有メモリ８０１のサイズを示す。シグネチャ８１３は名前付き共有メモリに利用者が付加する任意の値であり、一般的に整数値や短い文字列が用いられる。このシグネチャ８１３はアプリケーションプログラムとプリンタドライバが共通で識別出来る値にしておくと良い。入力データオフセット８１５は入力データの位置を示すオフセットであり、名前付き共有メモリ８０１の先頭からのバイト数などで表される。入力データサイズ８１７は入力データのサイズであり、後述する入力データ８３１のサイズをバイト数等で表す。出力データオフセット８１９は出力データの位置を示すオフセットであり、名前付き共有メモリ８０１の先頭からのバイト数などで表される。出力データ８２１は出力データのサイズであり、後述する出力データ８４１のサイズをバイト数等で表す。

【００５０】

生成可否情報８２３には、ユーザインタフェースモジュール４１１がベンダー独自情報の生成に成功したか否かを示す情報が格納される。アプリケーションプログラム２０１は本情報を参照し、ベンダー独自情報の生成状況を判定する。入力データ８３１は、アプリ

ケーションプログラム２０１がプリンタドライバに渡すデータが格納される。ここで、入力データ８３１はベンダー独自情報における単一の項目のみを要求しても良いし、設定項目の内容を細分化し、複数の設定項目を要求するようなデータを入力データとしてもよい。例えば、複数の設定項目の場合には、印刷設定情報と共に、ベンダー独自情報の種類として縁無し印刷におけるはみ出し量の設定値のリストをまとめて要求するデータを入力データとしても良い。

【００５１】

出力データ８４１は、ユーザインタフェースモジュール４１１が生成した、ベンダー独自情報が格納される。例えば、印刷設定情報に応じた縁無し印刷におけるはみ出し量のリスト等が格納される。ここで、入力データ８３１の入力内容などに従って、メンバ８４１の内容を細分化し、複数の設定項目を出力データとしてもよい。例えば、機能に応じた縁無し印刷におけるはみ出し量の設定値のリストと共に、リストに含まれる要素の個数を格納しても良い。ただし、入力データ８３１及び出力データ８４１に含まれるデータはこれに限定するものではなく、アプリケーションプログラムとユーザインタフェースモジュールとの間で通信すべきデータであればどのようなものでもかまわない。メンバ８５１は名前付き共有メモリ８０１の未使用領域である。

【００５２】

＜アプリケーションプログラムのベンダー独自情報の取得処理＞

次に、アプリケーションプログラムがベンダー独自情報を取得する処理について説明する。図９はアプリケーションプログラムがベンダー独自情報を取得する処理のフローチャートの一例である。まずＳ９０１で処理を開始する。続いて、名前付き共有メモリの名称を決定し、名前付き共有メモリを作成する（Ｓ９０３）。次にＳ９０５で取得するベンダー独自情報の種類を名前付き共有メモリに格納する。そして名前付き共有メモリの名称情報を引数に指定し、ベンダー独自ＥｓｃａｐｅコードでＥｘｔＥｓｃａｐｅ関数３０１を呼び出す。続いてＳ９０７において、アプリケーションプログラムはＥｘｔＥｓｃａｐｅ関数３０１の返却データを無視し、名前付き共有メモリに格納された生成成否情報８２３を参照する。続くＳ９０９で生成成否情報８０１からベンダー独自情報の生成に成功したかどうかを判定し、成功していればＳ９１１に移り、失敗していればＳ９１３に移る。Ｓ９１１では名前付き共有メモリ８０１に格納されたベンダー独自情報を取得する。Ｓ９１３では、アプリケーションプログラムが取得したいベンダー独自情報を全て取得したかどうかを判定し、取得済みであればＳ９１５に移り、取得済みでなければＳ９０５に移る。Ｓ９１５では、Ｓ９０３で作成した名前付き共有メモリを削除し、最後にＳ９１６で処理を終了する。

【００５３】

＜ＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数の処理＞

次に、ユーザインタフェースモジュールのＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数の処理について説明する。図１０はＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数における処理のフローチャートの一例である。

【００５４】

まず、Ｓ１００１で処理を開始する。続いてＳ１００３で、通知されたイベントの種類がＥｓｃａｐｅイベントかどうかを判定し、ＥｓｃａｐｅイベントならＳ１００５に移り、Ｅｓｃａｐｅイベントで無ければＳ１０１１に移る。Ｓ１００５ではＥｓｃａｐｅイベントで渡されたＥｓｃａｐｅコードがベンダー独自Ｅｓｃａｐｅコードかどうかを判定し、ベンダー独自ＥｓｃａｐｅコードであればＳ１００７に移り、ベンダー独自ＥｓｃａｐｅコードでなければＳ１００９に移る。Ｓ１００７ではＥｘｔＥｓｃａｐｅ関数から引数で渡された名前付き共有メモリの名称情報を取得し、名前付き共有メモリから、ベンダー独自情報の種類を参照する。そして、これに従いベンダー独自情報を生成し、ベンダー独自情報の生成に成功したか失敗したかを示す生成成否情報と共に、名前付き共有メモリに格納する。Ｓ１０１１では通知されたＥｓｃａｐｅイベント以外の各種イベントに応じた処理を行い、Ｓ１００９に移る。Ｓ１０１１の各種イベントと、イベントに応じた処理の

内容は本発明に関連が無いため詳細は説明しない。最後にＳ１００９で処理を終了する。

【００５５】

＜印刷システムの構成例＞

次に図１を用いて本実施形態を適用可能な印刷システムについて説明する。図１は本実施形態における印刷システムの構成を示すブロック図の一例である。本実施形態における印刷システムは大きく印刷装置１２０とデータ処理装置１００により構成され、データ処理装置１００はブロック１０１～１１４により構成される。データ処理装置はパーソナルコンピュータなどが想定されるが、本実施形態が実現できれば、どのようなものでも構わない。アプリケーションプログラム及びプリンタドライバが動作可能なＯＳの環境を有していればよい。

【００５６】

１０１は、ＣＲＴ表示装置であり、プリンタドライバのユーザインタフェースモジュールが提供する印刷設定ダイアログ等を表示する。１０２はＣＲＴＣで表示装置用のコントローラである。１０３は、キーボードなどのデータ入力装置であり、１０４は、キーボードコントローラである。１０５は、ポインティングデバイス等の座標入力装置であり、１０６は、ポインティングデバイスコントローラである。１０７は、装置全体の制御を司るＣＰＵである。１０８は、ブートプログラムなどを記憶しているＲＯＭである。１０９は、ＯＳ、各アプリケーションプログラム、本実施形態のフローチャートに関るプリンタドライバプログラムの格納場所としてや、さらにはワークエリアとしても利用されるＲＡＭである。１１０は、ＯＳ、各アプリケーションプログラム、本実施形態のフローチャートに関るプログラムを含むプリンタドライバプログラム、フォントデータ、さらにはデータファイル等を記憶しているハードディスク装置であり、１１１はハードディスクコントローラである。ハードディスク装置１１０には、スプールファイルも一時的に格納される。１１２は、可搬性記憶媒体の駆動装置であるフロッピー（登録商標）ディスク装置であり、１１３はフロッピー（登録商標）ディスクコントローラである。

【００５７】

１３０はインタフェースであり、１１４はインタフェースコントローラである。データ処理装置１００はインタフェースケーブルを介してインクジェットプリンタなどの印刷装置１２０に接続される。１１６は、各デバイスを接続するシステムバスである。本装置に電源が投入されると、ＣＰＵ１０７はＲＯＭ１０８に格納されているブートプログラムに従って起動し、ハードディスク装置１１０からＯＳをロードし、操作者の操作待ち状態になる。そして、操作者からＫＢ１０３またはＰＤ１０５からアプリケーションプログラムを介して印刷指示やプリンタドライバの印刷設定変更指示を受けた場合、もしくは自動的に起動するように設定されている場合は、ハードディスク装置１１０に格納されているプリンタドライバプログラムがＲＡＭ１０９にロードされＣＰＵ１０７により実行される。

【００５８】

＜その他の実施形態に関して＞

なお、本実施形態では、単一の名前付き共有メモリで入力データと出力データの両方を格納する例を示したが、複数の名前付き共有メモリに分割して格納するようにしても良いことは言うまでもない。

【００５９】

なお、本実施形態では、入力データと出力データを格納するために名前付き共有メモリを使用した例を示したが、他の名前付きオブジェクトでも良いことは言うまでもない。例えば、名前付き共有メモリの代わりにファイルを利用して、入力データと出力データを格納し、ファイルの名称をＯＳが提供するＡＰＩを介してアプリケーションプログラムからプリンタドライバに通知するといった方法が挙げられる。

【００６０】

また、本実施形態では、名前付き共有メモリの名称を、ＯＳが提供するＡＰＩを介してアプリケーションプログラムからプリンタドライバに通知する例を示したが、名称情報を渡すのではなく、名前付き共有メモリにアクセスするための情報を渡せばよいことは言う

までもない。例えば、名前付き共有メモリにアクセスするために利用する名称情報を生成するための情報を通知する様にしても良い。具体的にはプロセスＩＤとスレッドＩＤから名称情報を生成するのであれば、名称情報の生成規則をアプリケーションプログラムとプリンタドライバとの間で合致させた上で、プロセスＩＤとスレッドＩＤを通知する様な方法が挙げられる。

【００６１】

また、名前付き共有メモリではなく、他の名前付きオブジェクトを用いる場合、複数のプロセス間で名前付きオブジェクトのハンドルを共用できる場合は、名前付きオブジェクトにアクセスするための情報として、ハンドルを渡せばよいことは言うまでもない。

【００６２】

なお、本実施形態では、ＯＳが提供するＡＰＩとしてＥｘｔＥｓｃａｐｅ関数を使用する例を示したが、これに限定されるものではなく、ＯＳの印刷サポート機能を介してプリンタドライバを構成するモジュールが呼び出されるＡＰＩであれば良いことは言うまでもない。例えば、印刷設定情報を格納するＤＥＶＭＯＤＥ構造体のプライベート領域等に、名前付き共有メモリの名称情報とベンダー独自情報を取得するための領域を付加し、ＤｏｃｕｍｅｎｔＰｒｏｐｅｒｔｉｅｓ関数や、ＤｅｖｉｃｅＣａｐａｂｉｌｉｔｉｅｓ関数を用いても良いことは言うまでもない。

【００６３】

＜本実施形態の効果＞

以上の様にすることで、プリンタドライバはベンダー独自情報を名前付きオブジェクトに格納し、アプリケーションプログラムはプリンタドライバによって名前付きオブジェクトに格納された情報を参照することにより、ＯＳが提供するＡＰＩの仕様に制限されず処理を実行できる。また、アプリケーションプログラムがＡＰＩを呼び出した時にのみプリンタドライバが動作する為、ベンダー独自情報を提供する為の別プログラムをサービスや別プロセスとして待機させておく必要が無い。

【００６４】

さらに、アプリケーションプログラムは名前付き共有メモリとＥｘｔＥｓｃａｐｅ関数としたため、プリンタドライバにおけるユーザインタフェースモジュールのイベント処理部あるいはレンダリングモジュールのＥｓｃａｐｅ処理部のどちらでも、ベンダー独自情報を名前付き共有メモリに格納することが可能になる。ユーザインタフェースモジュールでの処理が可能になる為、ＸＰＳプリンタドライバでも、ベンダー独自情報をアプリケーションプログラムに返却することが出来る。

【００６５】

さらに、名前付きオブジェクトを用いることで、動作条件の異なるアプリケーションプログラムとプリンタドライバが別個のプロセスとして動作する場合にも、ベンダー独自情報をアプリケーションプログラムに返却することが出来る。もちろん、アプリケーションプログラムとプリンタドライバが同一プロセスとして動作する場合でも適用可能である。

【００６６】

さらに、６４ｂｉｔ版Ｗｉｎｄｏｗｓ製品に同梱されているプリンタドライバが、ベンダー独自情報を３２ｂｉｔ版アプリケーションプログラムに返却することができる。

【００６７】

以上により、ベンダー独自ＡＰＩを用いず、かつ、レンダリングモジュールに依存しないデータ通信方法が可能となる。

【００６８】

＜備考＞

なお本発明は、複数の機器（例えばホストコンピュータ、インタフェース機器、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。

【００６９】

また本発明の目的は、前述の実施形態の機能を実現するプログラムを記録した記録媒体

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が記憶媒体に格納されたプログラムを読み出し実行することによっても達成される。この場合、記憶媒体から読み出された実行可能なプログラム自体が前述した実施形態の機能を実現することになり、そのプログラム自体およびプログラムを記憶した記憶媒体は本発明を構成することになる。

【００７０】

また、本発明には、プログラムの指示に基づき、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。さらに、記憶媒体から読み出されたプログラムが、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き込まれた場合についても、本発明は適用される。その場合、書き込まれたプログラムの指示に基づき、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。

【００７１】

また、発明の実施の形態は、本発明を中核として構成される装置又は方法を説明している。このため本実施形態には本発明の本質的部分を加えて付加的な構成要件も記載されている。すなわち発明の実施の形態において説明した装置又は方法の構成要件を備えることは、本発明を成立させるための十分条件ではあるものの、必要条件ではない。

【図面の簡単な説明】

【００７２】

【図１】印刷システムの構成を示すブロック図である。

【図２】ベンダー独自のＡＰＩを利用する手法における、アプリケーションプログラムとプリンタドライバの関係を示す模式図である。

【図３】アプリケーションプログラムがＥｘｔＥｓｃａｐｅ関数を呼び出した時の従来の処理の流れを説明する為の模式図である。

【図４】アプリケーションプログラムがＥｘｔＥｓｃａｐｅ関数を呼び出した時の本発明における処理の流れを説明する為の模式図である。

【図５】ＸＰＳプリンタドライバの構成と印刷動作を説明するための模式図である。

【図６】縁無し印刷に関る印刷設定を行う為のダイアログの表示例である。

【図７】名前付き共有メモリとＥｘｔＥｓｃａｐｅ関数を用いたベンダー独自情報の取得及び関数呼び出しのタイミングチャートである。

【図８】名前付き共有メモリに格納されるデータの構造の模式図である。

【図９】アプリケーションプログラムがベンダー独自情報を取得する処理のフローチャートである。

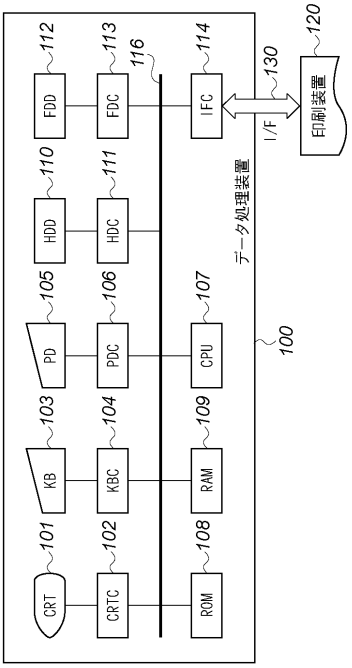
【図１０】ＤｒｖＤｏｃｕｍｅｎｔＥｖｅｎｔ関数の処理のフローチャートである。

【符号の説明】

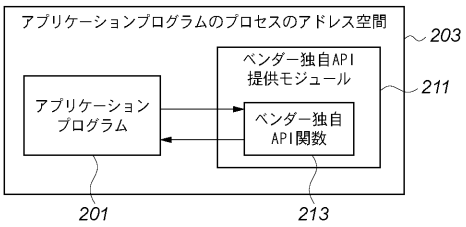
【００７３】

１２０　印刷装置
２０１　アプリケーションプログラム
３０１　ＯＳの印刷サポート機能
４１１　ユーザインタフェースモジュール
４２１　ＭＸＤＣ
５０１　スプールファイル
５１１　ＯＳの印刷パイプラインサービス
５２１　フィルタパイプライン
５２３　レイアウトフィルタ
５２５　レンダーフィルタ

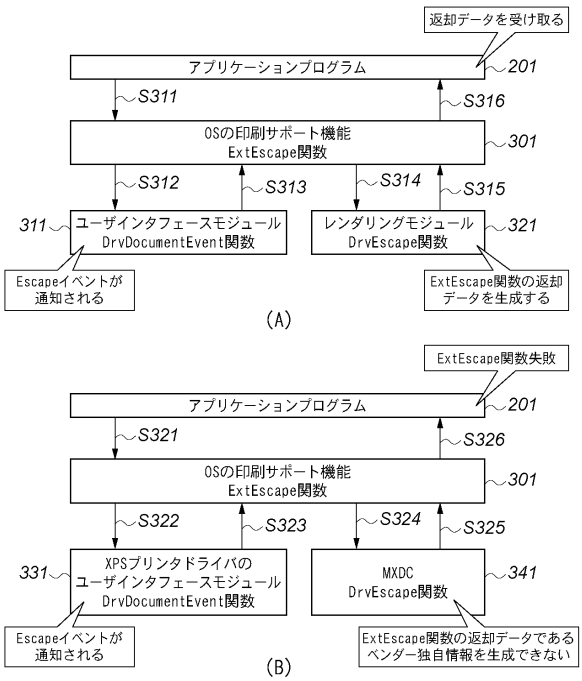
【図１】


FDD
FDC
IFC
印刷装置
I/F
データ処理装置
HDD
HDC
PD
PDC
CPU
KB
KBC
RAM
CRT
CRTC
ROM
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
116
120
130

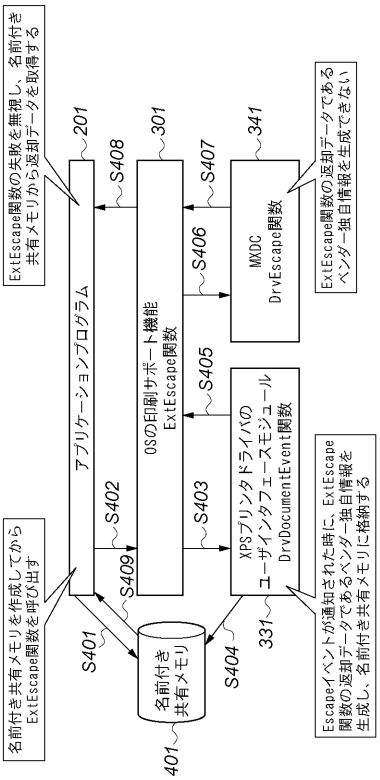
【図２】


アプリケーションプログラムのプロセスのアドレス空間
203
ベンダー独自API
提供モジュール
211
アプリケーション
プログラム
ベンダー独自
API関数
201
213

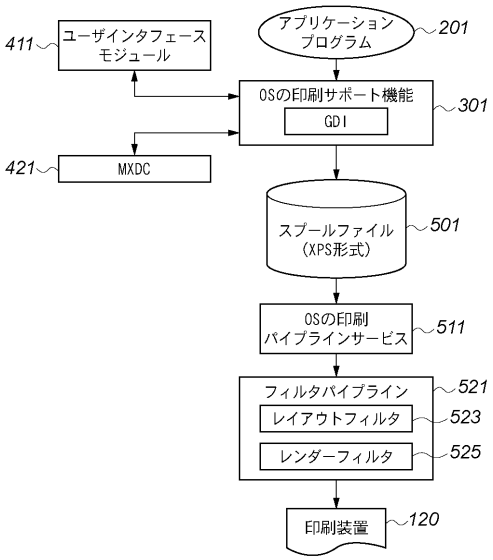
【図３】


返却データを受け取る
アプリケーションプログラム
201
S311
S316
OSの印刷サポート機能
ExtEscape関数
301
S312
S313
S314
S315
311
ユーザインタフェースモジュール
DrvDocumentEvent関数
レンダリングモジュール
DrvEscape関数
321
Escapeイベントが
通知される
ExtEscape関数の返却
データを生成する
(A)
ExtEscape関数失敗
アプリケーションプログラム
201
S321
S326
OSの印刷サポート機能
ExtEscape関数
301
S322
S323
S324
S325
331
XPSプリンタドライバの
ユーザインタフェースモジュール
DrvDocumentEvent関数
MXDC
DrvEscape関数
341
Escapeイベントが
通知される
ExtEscape関数の返却データである
ベンダー独自情報を生成できない
(B)

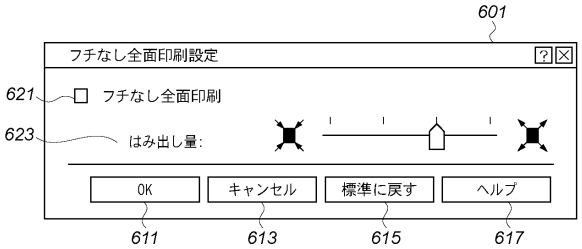
【図４】


ExtEscape関数の失敗を無視し、名前付き
共有メモリから返却データを取得する
201
S408
301
S407
341
S406
MXDC
DrvEscape関数
ExtEscape関数の返却データである
ベンダー独自情報を生成できない
アプリケーションプログラム
OSの印刷サポート機能
ExtEscape関数
S405
S402
S403
XPSプリンタドライバの
ユーザインタフェースモジュール
DrvDocumentEvent関数
名前付き共有メモリを作成してから
ExtEscape関数を呼び出す
S409
S401
名前付き
共有メモリ
401
S404
331
Escapeイベントが通知された時に、ExtEscape
関数の返却データであるベンダー独自情報を
生成し、名前付き共有メモリに格納する

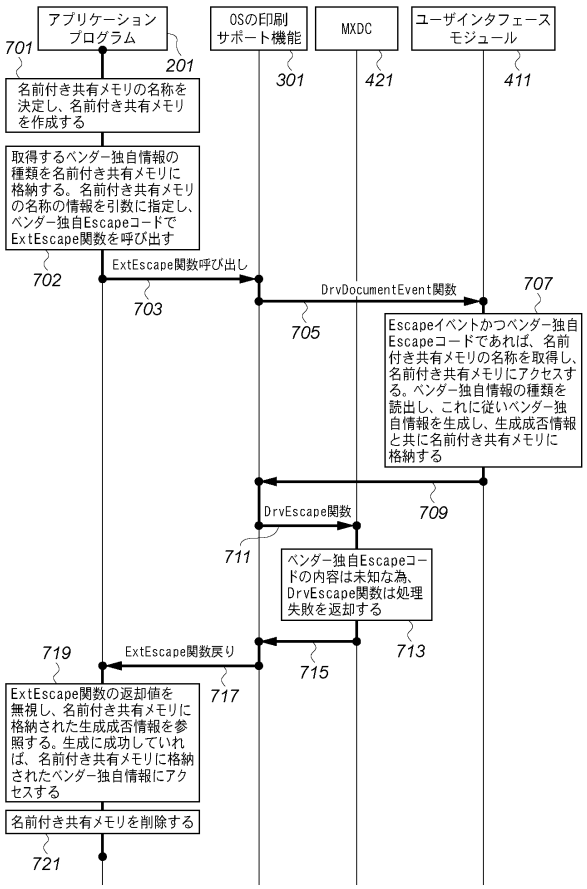
【図５】


411
ユーザインタフェース
モジュール
アプリケーション
プログラム
201
OSの印刷サポート機能
GDI
301
421
MXDC
スプールファイル
(XPS形式)
501
OSの印刷
パイプラインサービス
511
フィルタパイプライン
521
レイアウトフィルタ
523
レンダーフィルタ
525
印刷装置
120

【図６】


601
フチなし全面印刷設定
621
フチなし全面印刷
623
はみ出し量:
OK
キャンセル
標準に戻す
ヘルプ
611
613
615
617

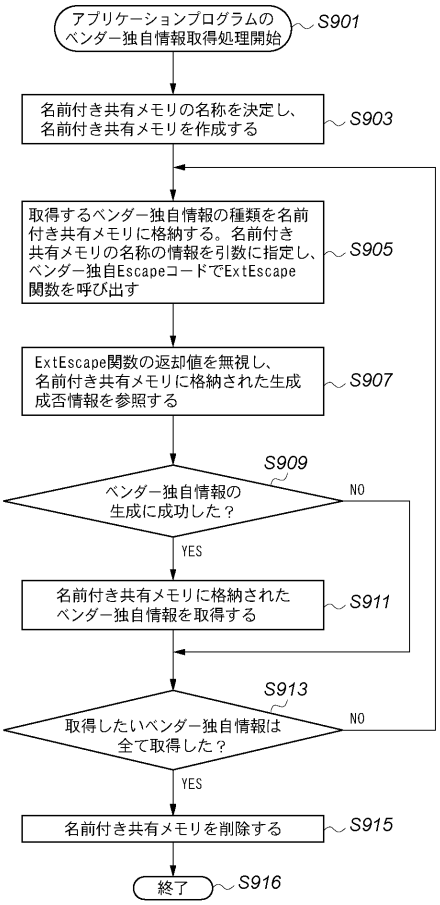
【図７】


アプリケーション
プログラム
701
201
OSの印刷
サポート機能
301
MXDC
421
ユーザインタフェース
モジュール
411
名前付き共有メモリの名称を
決定し、名前付き共有メモリ
を作成する
取得するベンダー独自情報の
種類を名前付き共有メモリに
格納する。名前付き共有メモ
リの名称の情報を引数に指定し、
ベンダー独自Escapeコードで
ExtEscape関数を呼び出す
702
ExtEscape関数呼び出し
703
DrvDocumentEvent関数
705
707
Escapeイベントかつベンダー独自
Escapeコードであれば、名前
付き共有メモリの名称を取得し、
名前付き共有メモリにアクセスす
る。ベンダー独自情報の種類を
読出し、これに従いベンダー独
自情報を生成し、生成成否情報
と共に名前付き共有メモリに
格納する
709
DrvEscape関数
711
ベンダー独自Escapeコー
ドの内容は未知な為、
DrvEscape関数は処理
失敗を返却する
713
715
719
ExtEscape関数戻り
717
ExtEscape関数の返却値を
無視し、名前付き共有メモリに
格納された生成成否情報を参
照する。生成に成功していれ
ば、名前付き共有メモリに格納
されたベンダー独自情報にアク
セスする
名前付き共有メモリを削除する
721

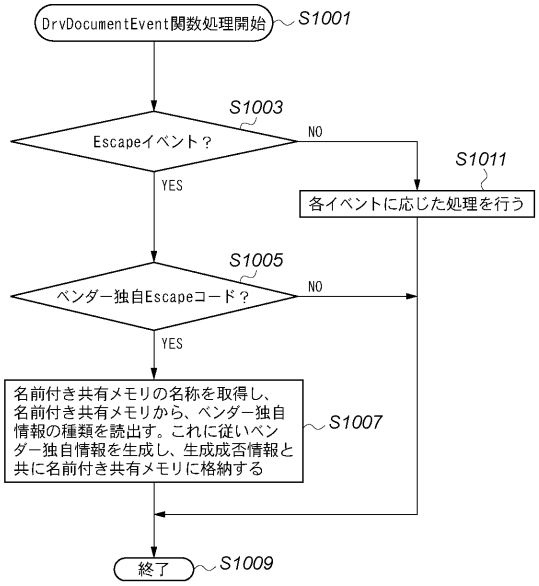
【図８】


801
サイズ
811
シグネチャ
813
入力データオフセット
815
入力データサイズ
817
出力データオフセット
819
出力データサイズ
821
生成成否情報
823
入力データ
831
出力データ
841
未使用領域
851

【図９】


アプリケーションプログラムの
ベンダー独自情報取得処理開始
S901
名前付き共有メモリの名称を決定し、
名前付き共有メモリを作成する
S903
取得するベンダー独自情報の種類を名前
付き共有メモリに格納する。名前付き
共有メモリの名称の情報を引数に指定し、
ベンダー独自EscapeコードでExtEscape
関数を呼び出す
S905
ExtEscape関数の返却値を無視し、
名前付き共有メモリに格納された生成
成否情報を参照する
S907
S909
ベンダー独自情報の
生成に成功した？
NO
YES
名前付き共有メモリに格納された
ベンダー独自情報を取得する
S911
S913
取得したいベンダー独自情報は
全て取得した？
NO
YES
名前付き共有メモリを削除する
S915
終了
S916

【図１０】


DrvDocumentEvent関数処理開始
S1001
S1003
Escapeイベント？
NO
YES
S1011
各イベントに応じた処理を行う
S1005
ベンダー独自Escapeコード？
NO
YES
名前付き共有メモリの名称を取得し、
名前付き共有メモリから、ベンダー独自
情報の種類を読出す。これに従いベン
ダー独自情報を生成し、生成成否情報と
共に名前付き共有メモリに格納する
S1007
終了
S1009

フロントページの続き

(72)発明者　名取　英夫
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　篠塚　隆

(56)参考文献　特開２００４－２８８０１３（ＪＰ，Ａ）
特開２００５－２５８７１２（ＪＰ，Ａ）
特開２００５－５０３０７（ＪＰ，Ａ）
特開平１０－２４８０１４（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ３／１２
９／４６
９／４８
９／５０－９／５２
９／５４
１３／１０