(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2007-242052
(P2007-242052A)
(43)公開日　平成19年9月20日(2007.9.20)

(51)Int.Cl.			F I			テーマコード（参考）
G06F	3/12	(2006.01)	G06F	3/12	C	5B021
H04N	1/00	(2006.01)	H04N	1/00	C	5C062
			H04N	1/00	Z	

審査請求 有　請求項の数 8　ＯＬ　（全 16 頁）

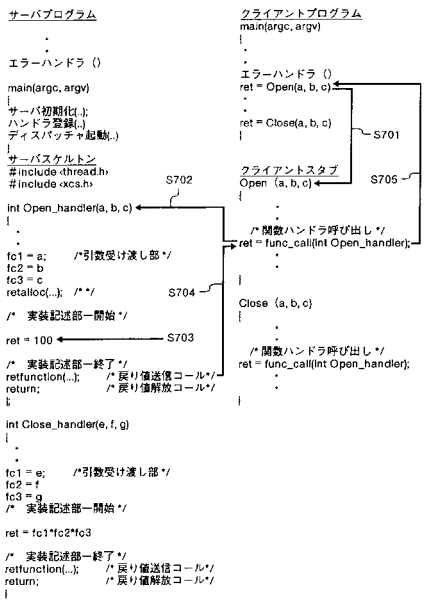
(21)出願番号　特願2007-140166 (P2007-140166)
(22)出願日　平成19年5月28日 (2007.5.28)
(62)分割の表示　特願2001-257042 (P2001-257042)の分割
原出願日　平成13年8月27日 (2001.8.27)

(71)出願人　000006747
株式会社リコー
東京都大田区中馬込１丁目３番６号
(74)代理人　100089118
弁理士　酒井　宏明
(72)発明者　千田　滋也
東京都大田区中馬込１丁目３番６号　株式会社リコー内

Ｆターム(参考)　5B021 AA05 AA19 CC05
5C062 AA02 AA05 AA13 AA35 AB38
AB41 AB42 AC34 AE15

(54)【発明の名称】プロセス間通信プログラムおよび画像情報処理装置

(57)【要約】
【課題】画像情報処理装置で提供されるアプリケーションやコントロールサービスのプログラムに可変性および多様性を持たせることができ、かつアプリケーションとコントロールサービスとを別個独立に開発すること。
【解決手段】印刷部または撮像部を有するハードウェア資源を利用した画像情報処理にかかるアプリケーションと、オペレーティングシステムと、オペレーティングシステム上で動作し、複数のアプリケーションからアクセスされてハードウェア資源の制御を行うプログラムとを備えた画像情報処理装置に実行させ、プロセス間通信を担うプロセス間通信プログラムであって、クライアントプロセスとなるアプリケーションによって、サーバプロセスになるプログラムに対する画像情報処理にかかる処理要求または設定要求の関数の呼び出しを行う関数呼び出しステップ、を画像情報処理装置に実行させる。
【選択図】　図７



【特許請求の範囲】
【請求項１】
印刷部または撮像部を有するハードウェア資源を利用した画像情報処理にかかるアプリケーションと、オペレーティングシステムと、前記オペレーティングシステム上で動作し、複数の前記アプリケーションからアクセスされて前記ハードウェア資源の制御を行うプログラムとを備えた画像情報処理装置に実行させ、プロセス間通信を担うプロセス間通信プログラムであって、
クライアントプロセスとなる前記アプリケーションによって、サーバプロセスになる前記プログラムに対する前記画像情報処理にかかる処理要求または設定要求の関数の呼び出しを行う関数呼び出しステップ、
を前記画像情報処理装置に実行させるプロセス間通信プログラム。
【請求項２】
前記関数呼び出しステップは、前記サーバプロセスになるとともに前記ハードウェア資源としてのエンジンを制御するエンジンコントロールサービスに対する処理要求または設定要求を行うジョブ制御関数の呼び出しを行うことを特徴とする請求項１に記載のプロセス間通信プログラム。
【請求項３】
前記関数呼び出しステップは、前記サーバプロセスになるとともに前記ハードウェア資源としてのメモリおよびハードディスクを制御するメモリコントロールサービスに対する処理要求または設定要求を行うメモリ制御関数の呼び出しを行うことを特徴とする請求項１または２に記載のプロセス間通信プログラム。
【請求項４】
前記関数呼び出しステップは、前記サーバプロセスになるとともに前記ハードウェア資源としてのファックスコントロールユニットを利用するファックス通信を制御するファックスコントロールサービスに対する処理要求または設定要求を行うファックス通信制御関数の呼び出しを行うことを特徴とする請求項１～３のいずれか一つに記載のプロセス間通信プログラム。
【請求項５】
印刷部または撮像部を有するハードウェア資源を利用した画像情報処理にかかるアプリケーションと、
オペレーティングシステムと、
前記オペレーティングシステム上で動作し、複数の前記アプリケーションからアクセスされて前記ハードウェア資源の制御を行うプログラムと、を備え、
前記アプリケーションは、サーバプロセスになる前記プログラムに対する前記画像情報処理にかかる処理要求または設定要求の関数の呼び出しを行う関数呼び出し手段、
を備えたことを特徴とする画像情報処理装置。
【請求項６】
前記関数呼び出し手段は、前記サーバプロセスになるとともに前記ハードウェア資源としてのエンジンを制御するエンジンコントロールサービスに対する処理要求または設定要求を行うジョブ制御関数の呼び出しを行うことを特徴とする請求項５に記載の画像情報処理装置。
【請求項７】
前記関数呼び出し手段は、前記サーバプロセスになるとともに前記ハードウェア資源としてのメモリおよびハードディスクを制御するメモリコントロールサービスに対する処理要求または設定要求を行うメモリ制御関数の呼び出しを行うことを特徴とする請求項５または６に記載の画像情報処理装置。
【請求項８】
前記関数呼び出し手段は、前記サーバプロセスになるとともに前記ハードウェア資源としてのファックスコントロールユニットを利用するファックス通信を制御するファックスコントロールサービスに対する処理要求または設定要求を行うファックス通信制御関数の

呼び出しを行うことを特徴とする請求項５～７のいずれか一つに記載の画像情報処理装置。

【発明の詳細な説明】

【技術分野】

【０００１】

この発明は、プリンタ、コピーまたはファクシミリなどの複合サービスを行う画像情報処理装置で動作するサーバプロセスとクライアントプロセスの間のプロセス間通信を担うプロセス間通信プログラムおよび画像情報処理装置に関する。

【背景技術】

【０００２】

近年、プリンタ、コピー、ファクシミリ、スキャナなどの各装置の機能を１つの筐体内に収納した画像形成装置（以下、「複合機」という。）が一般的に知られている。この複合機は、１つの筐体内に表示部、印刷部および撮像部などを設けるとともに、プリンタ、コピーおよびファクシミリ装置にそれぞれ対応する３種類のソフトウェアを設け、ソフトウェアの切り替えによって、当該装置をプリンタ、コピー、スキャナまたはファクシミリ装置として動作させるものである。

【０００３】

従来の複合機では、内部にプリンタ、コピー、スキャナおよびファクシミリ装置に対応するソフトウェア（汎用ＯＳを含む）をそれぞれ別個に設ける構成となっており、各ソフトウェアの開発に多大の時間を要する。このため、出願人は、表示部、印刷部および撮像部などの画像形成処理で使用されるハードウェア資源を有し、プリンタ、コピーまたはファクシミリなどの各ユーザサービスにそれぞれ固有の処理をおこなうアプリケーションを複数搭載し、これらのアプリケーションとハードウェア資源との間に介在して、ユーザサービスを提供する際に、アプリケーションの少なくとも２つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行う各種コントロールサービスからなるプラットホームを備えた画像形成装置を発明した。この画像形成装置によれば、アプリケーションの少なくとも２つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行うプラットホームを備えた構成とすることによって、ソフトウェア開発の効率化を図るとともに、装置全体としての生産性を向上させることが可能となる。

【発明の開示】

【発明が解決しようとする課題】

【０００４】

このような複合機では、アプリケーションの少なくとも２つが共通的に必要とするサービスを提供するコントロールサービスを有する構成となっているため、機能変更、機能追加、または将来的なハードウェア資源の追加、変更などに柔軟に対応すべく、アプリケーションごとあるいはコントロールサービスごとの追加、変更を容易に行えることが望まれる。また、複合サービスを提供する際に、各アプリケーションと各コントロールサービス間、あるいは各コントロールサービス間でサービスやデータの授受を円滑に行うことが必要となってくる。

【０００５】

すなわち、複合機は、プリンタ、コピー、ファクシミリ、スキャナなど多種の機能を備えたものであるが、その一部の機能に故障あるいは仕様変更などが生じた場合に、全てのアプリケーションまたは全てのコントロールサービスのモジュールを修正しなければならないと、作業の労力が過大となる。また、将来的に、複合機に新たな機能を追加する場合でも一部の機能の追加のために全てのコントロールサービスや全てのアプリケーションのモジュール変更が必要となると、プログラム開発の労力が過大となってしまう。このため、複合機上で動作する各アプリケーションやコントロールサービスのモジュールは互いにサービスやデータの送受信を実現としながらも、モジュール間の独立性を維持していることが必要となってくる。

【０００６】

すなわち、アプリケーションや各コントロールサービスごとに独立性がないと、必要なアプリケーションや必要なコントロールサービスのみを提供することが困難となり、機能変更や機能追加などに柔軟に対応できるような複合機の構成に可変性を持たせることができないばかりか、多種多様な機能の実現が困難となるという問題がある。

【０００７】

また、各アプリケーションや各コントロールサービスごとに独立性がないと、アプリケーションやコントロールサービスごとの設計および構築が困難になるという問題がある。すなわち、アプリケーションの少なくとも２つが共通的に必要とするサービスを提供するコントロールサービスを有する構成の複合機では、その特徴的な構成を利用して、アプリケーションのプログラム開発を画像形成装置の製造元以外のサードベンダに委ねることも考えられるが、各モジュールに独立性がないと、ハードウェア資源との直接的なやりとりを行うコントロールサービスの内部動作をサードベンダが熟知していないと、アプリケーションの開発が困難となる。一方、画像形成装置の開発元としては、コントロールサービスの内部動作をサードベンダに開示することは営業秘密上好ましくない。

【０００８】

この発明は上記に鑑みてなされたもので、画像情報処理装置で提供されるアプリケーションやコントロールサービスのプログラムに可変性および多様性を持たせることができ、かつアプリケーションとコントロールサービスとを別個独立に開発することが容易に行えるプロセス間通信プログラムおよび画像情報処理装置を得ることを目的とする。

【課題を解決するための手段】

【０００９】

上述した課題を解決し、目的を達成するために、請求項１にかかる発明は、印刷部または撮像部を有するハードウェア資源を利用した画像情報処理にかかるアプリケーションと、オペレーティングシステムと、前記オペレーティングシステム上で動作し、複数の前記アプリケーションからアクセスされて前記ハードウェア資源の制御を行うプログラムとを備えた画像情報処理装置に実行させ、プロセス間通信を担うプロセス間通信プログラムであって、クライアントプロセスとなる前記アプリケーションによって、サーバプロセスになる前記プログラムに対する前記画像情報処理にかかる処理要求または設定要求の関数の呼び出しを行う関数呼び出しステップ、を前記画像情報処理装置に実行させる。

【００１０】

また、請求項２にかかる発明は、前記関数呼び出しステップは、前記サーバプロセスになるとともに前記ハードウェア資源としてのエンジンを制御するエンジンコントロールサービスに対する処理要求または設定要求を行うジョブ制御関数の呼び出しを行うことを特徴とする。

【００１１】

また、請求項３にかかる発明は、前記関数呼び出しステップは、前記サーバプロセスになるとともに前記ハードウェア資源としてのメモリおよびハードディスクを制御するメモリコントロールサービスに対する処理要求または設定要求を行うメモリ制御関数の呼び出しを行うことを特徴とする。

【００１２】

また、請求項４にかかる発明は、前記関数呼び出しステップは、前記サーバプロセスになるとともに前記ハードウェア資源としてのファックスコントロールユニットを利用するファックス通信を制御するファックスコントロールサービスに対する処理要求または設定要求を行うファックス通信制御関数の呼び出しを行うことを特徴とする。

【００１３】

また、請求項５にかかる発明は、印刷部または撮像部を有するハードウェア資源を利用した画像情報処理にかかるアプリケーションと、オペレーティングシステムと、前記オペレーティングシステム上で動作し、複数の前記アプリケーションからアクセスされて前記ハードウェア資源の制御を行うプログラムと、を備え、前記アプリケーションは、サーバプロセスになる前記プログラムに対する前記画像情報処理にかかる処理要求または設定要

求の関数の呼び出しを行う関数呼び出し手段を備えたことを特徴とする。

【００１４】

また、請求項６にかかる発明は、前記関数呼び出し手段は、前記サーバプロセスになるとともに前記ハードウェア資源としてのエンジンを制御するエンジンコントロールサービスに対する処理要求または設定要求を行うジョブ制御関数の呼び出しを行うことを特徴とする。

【００１５】

また、請求項７にかかる発明は、前記関数呼び出し手段は、前記サーバプロセスになるとともに前記ハードウェア資源としてのメモリおよびハードディスクを制御するメモリコントロールサービスに対する処理要求または設定要求を行うメモリ制御関数の呼び出しを行うことを特徴とする。

【００１６】

また、請求項８にかかる発明は、前記関数呼び出し手段は、前記サーバプロセスになるとともに前記ハードウェア資源としてのファックスコントロールユニットを利用するファックス通信を制御するファックスコントロールサービスに対する処理要求または設定要求を行うファックス通信制御関数の呼び出しを行うことを特徴とする。

【発明の効果】

【００１７】

本発明によれば、印刷部または撮像部を有するハードウェア資源を利用した画像情報処理にかかるアプリケーションと、オペレーティングシステムと、前記オペレーティングシステム上で動作し、複数の前記アプリケーションからアクセスされて前記ハードウェア資源の制御を行うプログラムとを備えた画像情報処理装置に実行させ、プロセス間通信を担うプロセス間通信プログラムであって、クライアントプロセスとなる前記アプリケーションによって、サーバプロセスになる前記プログラムに対する前記画像情報処理にかかる処理要求または設定要求の関数の呼び出しを行う関数呼び出しステップを前記画像情報処理装置に実行させることで、通信プロトコルの隠蔽性を図りつつ画像情報処理装置で動作するクライアントオブジェクトプログラムを汎用的に開発することができるという効果を奏する。

【００１８】

また、本発明によれば、印刷部または撮像部を有するハードウェア資源を利用した画像情報処理にかかるアプリケーションと、オペレーティングシステムと、前記オペレーティングシステム上で動作し、複数の前記アプリケーションからアクセスされて前記ハードウェア資源の制御を行うプログラムと、を備え、前記アプリケーションは、サーバプロセスになる前記プログラムに対する前記画像情報処理にかかる処理要求または設定要求の関数の呼び出しを行う関数呼び出し手段、を備えたことで、クライアントプロセスが動作する画像情報処理装置の構成に可変性を持たせることができ、画像情報処理装置に多種多様な機能を実現させることができるという効果を奏する。

【発明を実施するための最良の形態】

【００１９】

以下に添付図面を参照して、この発明にかかる、プロセス間通信プログラムおよび画像情報処理装置の好適な実施の形態を詳細に説明する。

【００２０】

（実施の形態１）

図１は、この発明の実施の形態１である画像情報処理装置用通信プログラム生成装置（以下、「スタブジェネレータ」という。）の機能的構成を示すブロック図である。実施の形態１のスタブジェネレータは、メッセージファイルからヘッダと、クライアントスタブと、サーバスケルトンを自動生成するものである。

【００２１】

図１に示すように、メッセージファイル１０４を入力する入力部１０１と、メッセージファイル１０４に記述された内容の構文チェックを行う構文解析部１０２と、メッセージ

ファイル１０４の記述内容から、ヘッダ１０６とクライアントスタブ１０７と、サーバスケルトン１０５とを生成してハードディスク等の記憶媒体に格納するコード生成部１０３とを備えている。

【００２２】

まず、実施の形態１のスタブジェネレータに入力されるメッセージファイル１０４と、スタブジェネレータで生成されるヘッダ１０６、クライアントスタブ１０７およびサーバスケルトン１０５について説明する。メッセージファイル１０４は、プロセス間通信の通信内容をＣ言語などのソースコードで記述したソースファイルである。

【００２３】

図３は、メッセージファイル１０４の記述例を示す説明図である。図３に示すように、メッセージファイル１０４には、別のメッセージファイルや、Ｃ言語記述のインクルードヘッダを宣言するインクルード宣言と、サーバプロセスとクライアントプロセスの間で送受信するメッセージを記述したメッセージ記述と、クライアントプロセスからサーバプロセスに対して発行する関数の宣言などが記述されている。

【００２４】

メッセージは、主としてサーバプロセスとクライアントプロセスとの間でイベントや通知を行う際に発行するものであり、メッセージファイル１０４のメッセージ記述としては、メッセージ名と、メッセージＩＤと、メッセージ方向、メッセージ内容を記述するようになっている。

【００２５】

ここで、メッセージ方向には、「ＩＮ」、「ＯＵＴ」、「ＳＥＬＦ」があり、「ＩＮ」はクライアントプロセスからサーバプロセスに対して送信するメッセージを意味し、「ＯＵＴ」はサーバプロセスからクライアントプロセスに対して送信するメッセージを意味する。また、「ＳＥＬＦ」は、自分自身のプロセスに対して送信するメッセージを意味する。

【００２６】

関数は、クライアントプロセスからサーバプロセスに処理要求や設定要求を行う際に発行するものである。メッセージファイル１０４の関数宣言には、関数名と、関数の型と、引数のみを記述し、関数の処理内容は記述しないようになっている。

【００２７】

クライアントスタブ１０７は、クライアントプログラムから呼び出される関数のサーバプロセスに対する発行を記述したソースファイルである。クライアントスタブ１０７をコンパイルしてライブラリ化し、クライアントプログラムとリンクすることにより、クライアントプログラムからの関数コールがクライアントスタブ１０７を介してサーバプロセスへ発行されるようになっている。

【００２８】

図４はスタブジェネレータにより生成されるクライアントスタブ１０７の一例を示す説明図である。図４に示すように、クライアントスタブ１０７には、後述するサーバスケルトン１０５に登録された関数ハンドラを呼び出すことにより、関数コールが行われるようになっている。

【００２９】

サーバスケルトン１０５は、クライアントスタブ１０７から呼び出された関数ハンドラとメッセージハンドラを登録したソースファイルである。

【００３０】

図５はスタブジェネレータにより生成されるサーバスケルトン１０５の一例を示す説明図である。図５に示すように、サーバスケルトン１０５には、クライアントスタブ１０７に記述された関数に対応した関数ハンドラが登録されているが、関数ハンドラでは、引数の受け渡し部のみが記述され、処理内容を記述する実装記述部は空欄の状態で生成されている。この実装記述部には、クライアントプログラムからクライアントスタブ１０７を介して発行される関数の処理内容を、サーバプログラムの開発者が自在に記述できるように

なっている。

【００３１】

ヘッダ１０６は、サーバスケルトン１０５とクライアントスタブ１０７で共通の定義、宣言などを記述したソースファイルである。図６はスタブジェネレータにより生成されるヘッダ１０６の一例を示す説明図である。図６に示すように、ヘッダ１０６には、メッセージファイル１０４に記述されたインクルード宣言、メッセージＩＤ、メッセージの構造体などのメッセージ宣言、関数宣言および関数ハンドラ宣言などが記述された状態で生成されるようになっている。

【００３２】

次に、実施の形態１のスタブジェネレータによるサーバスケルトン１０５、クライアントスタブ１０７およびヘッダ１０６の生成処理について説明する。図２は、サーバスケルトン１０５、クライアントスタブ１０７およびヘッダ１０６の生成処理のフローチャートである。

【００３３】

スタブジェネレータの構文解析部１０２は、メッセージファイル１０４に記述された関数宣言、メッセージ定義、インクルード宣言、Ｃ言語記述の文法チェックを行い（ステップＳ２０１）、さらにメッセージファイル１０４に定義されている関数名およびメッセージ定義の中のメッセージＩＤ、メッセージ名の一意性の判断を行う（ステップＳ２０２）。

【００３４】

そして、関数宣言、メッセージ定義、インクルード宣言の記述に文法エラーがあった場合、あるいは関数名、メッセージＩＤ、メッセージ名のいずれかが重複している場合には、ユーザに構文エラーである旨のメッセージを通知し処理を終了する（ステップＳ２０３、Ｓ２０４）。

【００３５】

構文解析後、コード生成部１０３は、メッセージファイル１０４の記述内容から、ヘッダ１０６と、サーバスケルトン１０５と、クライアントスタブ１０７を生成する。具体的には、メッセージファイル１０４のインクルード宣言とＣ言語記述とをヘッダ１０６にそのまま転記し、メッセージファイル１０４のメッセージ定義に記述されているメッセージ構造体をヘッダ１０６にコピーする（ステップＳ２０５）。

【００３６】

また、コード生成部１０３は、メッセージファイル１０４の関数宣言から関数ハンドラ名を自動生成し（たとえば、関数名ａｂｃの場合、関数ハンドラ名をａｂｃ＿ｈａｎｄｌｅｒとして生成する）、生成された関数ハンドラ名をサーバスケルトン内に登録する。そして、関数ハンドラの処理内に、引数受け渡しの記述を行うとともに、戻り値領域の生成システムコールの発行を記述する。そして、関数ハンドラの実際の処理内容を記述する実装記述部の欄を空欄とし、その実装記述部の後に、戻り値送信と戻り値領域解放の各システムコールの発行を記述する（ステップＳ２０６）。

【００３７】

また、コード生成部１０３は、メッセージファイル１０４に記述された関数宣言ごとに、関数名と引数の記述をクライアントスタブ１０７に登録し、その関数の処理内に、関数コールメッセージ生成コールと関数ハンドラの呼び出しと戻り値解放の各システムコールを記述する（ステップＳ２０７）。なお、関数ハンドラ呼び出しには、発行した関数の戻り値の待ち受ける関数戻り値待ちシステムコールが内部的に発行されるようになっている。

【００３８】

このようにタブジェネレータにより、クライアントスタブ１０７とサーバスケルトン１０５とヘッダ１０６を生成したら、次にサーバオブジェクトおよびクライアントオブジェクトを完成させる。サーバプログラムの開発者は、サーバスケルトン１０５の関数ハンドラの実装記述部（空欄）に、関数ハンドラで行うべき処理内容をエディタなどを利用して

記述し、サーバスケルトン１０５をヘッダ１０６とともにコンパイルしてサーバスタブオブジェクトを生成する。

【００３９】

また、サーバスケルトン１０５に登録した関数ハンドラを宣言するとともに、クライアントプロセスとの間でメッセージファイル１０４に記述されたメッセージの送受信を行うメッセージハンドラ、エラー処理を行うエラーハンドラ、サーバの初期化、サーバディスパッチャ起動等のシステムコールの発行を記述したサーバソースファイルを作成してコンパイルし、サーバスタブオブジェクトとリンクして、サーバプログラムを完成させる。

【００４０】

一方、クライアントスタブ１０７はヘッダ１０６とともにコンパイルしてクライアントスタブオブジェクトを生成する。サーバプロセスごとにクライアントスタブオブジェクトを生成し、生成した複数のクライアントスタブオブジェクトをライブラリとする。

【００４１】

また、関数の発行、関数ハンドラの宣言、エラー処理を行うエラーハンドラの宣言、クライアントプロセスとの間でメッセージファイル１０４に記述されたメッセージの送受信を行うメッセージハンドラ、およびクライアントプロセスの初期化やクライアントディスパッチャ起動等のシステムコールの発行を記述したクライアントプログラムを作成してコンパイルし、スタブのライブラリとリンクすることによりクライアントプログラムを完成させる。

【００４２】

図７は、クライアントプログラムから関数コールを行った場合におけるサーバプログラムとの呼び出しおよび応答のシーケンスを示す説明図である。なお、サーバプログラムとクライアントプログラムは実際は実行可能形式のオブジェクトファイルであるが、図７では、説明の都合上いずれもソースコードで表示している。

【００４３】

クライアントプログラムからサーバプログラムに対して、たとえばＯｐｅｎ関数をコールすると、クライアントスタブオブジェクトで定義されたＯｐｅｎが呼び出される（ステップＳ７０１）。そして、このスタブのＯｐｅｎの処理の中で、Ｏｐｅｎの関数ハンドラＯｐｅｎ＿ｈａｎｄｌｅｒがサーバプログラムに対してコールされる（ステップＳ７０２）。

【００４４】

サーバプログラムでは、クライアントスタブオブジェクトから関数ハンドラＯｐｅｎ＿ｈａｎｄｌｅｒの発行を受け取って、サーバスタブオブジェクトに登録されているＯｐｅｎ＿ｈａｎｄｌｅｒの実装記述部に記述されている処理を実行し（ステップＳ７０３）、その実行結果をクライアントスタブへ戻り値として返す（ステップＳ７０４）。クライアントプログラムでは、この戻り値をクライアントスタブから受け取って（ステップＳ７０５）、次の処理へ移行する。

【００４５】

このように実施の形態１のスタブジェネレータでは、メッセージファイル１０４に関数宣言を記述することによりクライアントスタブ１０７とサーバスケルトン１０５を容易に生成することができる。しかも生成されるサーバスケルトン１０５の実装記述部は空欄となっているので、プログラム開発者は必要に応じて関数の処理内容を容易に変更するができる。このため複合機に可変性をもたせることができるとともに、多種多様な機能を実現させることができる。さらに、クライアントプログラムの開発を、サードベンダーなどの他社が行う場合でも、クライアントプログラムには提供される関数コールを記述するだけでプロセス間通信が可能となるので、内部の通信プロトコルの隠蔽性を保つことが可能となる。

【００４６】

また、実施の形態１のスタブジェネレータでは、プロセス間通信を関数コールで実現しているので、ジョブの並列実行をスレッドで実現した場合でも、高速なプロセス間通信が

可能となる。また、関数からの戻り値によってスレッド間の同期をとることができるので、同期のための処理プログラムを別途作成する必要がなくなり、プログラム開発の労力を低減することができる。

【００４７】

（実施の形態２）

実施の形態１のスタブジェネレータは、複合機で動作するプロセスやスレッドを特に限定しないものであったが、この実施の形態２のスタブジェネレータは、複合機で動作するコントロールサービスをサーバプロセスとした場合のサーバスケルトン、クライアントスタブおよびヘッダの生成を行うものである。

【００４８】

実施の形態２のスタブジェネレータの構成は、図１に示す実施の形態１のスタブジェネレータの構成と同様であり、またスタブジェネレータによるクライアントスタブ、サーバスケルトンおよびヘッダの生成処理も図２に示す実施の形態１のスタブジェネレータによる処理手順と同様であるので説明を省略する。

【００４９】

ここで、図８は、実施の形態２のスタブジェネレータで生成したサーバプロセス、クライアントプロセスが動作する複合機のソフトウェア構成を示すブロック図である。図８に示すように、複合機は、白黒ラインプリンタ（Ｂ＆Ｗ LP）８０１と、カラーラインプリンタ（Color LP）８０２と、その他ハードウェアリソース８０３などを有するとともに、ソフトウェア群８１０はプラットホーム８２０およびアプリケーション８３０からなる。

【００５０】

プラットホーム８２０は、アプリケーションからの処理要求を解釈してハードウェア資源の獲得要求を発生させるコントロールサービスと、一または複数のハードウェア資源の管理を行い、コントロールサービスからの獲得要求を調停するシステムリソースマネージャ（ＳＲＭ）８２３と、ＵＮＩＸ（登録商標）などの汎用ＯＳ８２１とを有する。

【００５１】

コントロールサービスは、複数のサービスモジュールから形成され、ＳＣＳ（システムコントロールサービス）８２２と、ＥＣＳ（エンジンコントロールサービス）８２４と、ＭＣＳ（メモリコントロールサービス）８２５と、ＯＣＳ（オペレーションパネルコントロールサービス）８２６と、ＦＣＳ（ファックスコントロールサービス）８２７と、ＮＣＳ（ネットワークコントロールサービス）８２８とから構成される。

【００５２】

ＳＣＳ８２２はアプリ管理、操作部制御、システム画面表示、ＬＥＤ表示、リソース管理、割り込みアプリ制御を行うものであり、ＥＣＳ８２４は、白黒ラインプリンタ（Ｂ＆Ｗ LP）８０１、カラーラインプリンタ（Color LP）８０２、その他ハードウェアリソース８０３などのエンジンを制御するものである。

【００５３】

ＭＣＳ８２５は、画像メモリの取得および解放、ハードディスク装置（ＨＤＤ）の利用、画像データの圧縮および伸張などを行うものである。ＯＣＳ８２６は、オペレータと本体制御間の情報伝達手段となる操作パネルを制御するモジュールである。

【００５４】

ＦＣＳ８２７は、システムコントローラの各アプリ層からＰＳＴＮ／ＩＳＤＮ網を利用したファクシミリ送受信、ＢＫＭ（バックアップＳＲＡＭ）で管理されている各種ファクシミリデータの登録／引用、ファクシミリ読みとり、ファクシミリ受信印刷、融合送受信を行うためのＡＰＩを提供するものである。

【００５５】

ＮＣＳ８２８は、ネットワークＩ／Ｏを必要とするアプリケーションに対して共通に利用できるサービスを提供するためのモジュール群であり、ネットワーク側から各プロトコルによって受信したデータを各アプリケーションに振り分けたり、アプリケーションからデータをネットワーク側に送信する際の仲介を行うものである。

【００５６】
アプリケーション８３０は、ページ記述言語（ＰＤＬ）、ＰＣＬおよびポストスクリプト（ＰＳ）を有するプリンタ用のアプリケーションであるプリンタアプリ８１１と、コピー用アプリケーションであるコピーアプリ８１２と、ファクシミリ用アプリケーションであるファックスアプリ８１３と、スキャナ用アプリケーションであるスキャナアプリ８１４と、ネットワークファイル用アプリケーションであるネットファイルアプリ８１５と、工程検査用アプリケーションである工程検査用アプリ８１６とを有している。
【００５７】
これらの各コントロールサービスと各アプリとは、ＲＰＣ(Remote ProcessorCall)を利用したプロセス間通信によって相互に各種データを送受信している。実施の形態２のスタブジェネレータは、このようなプロセス間通信を関数コールで実現するためのクライアントスタブおよびサーバスケルトンを生成するものである。
【００５８】
次に、ＥＣＳ８２４をサーバプロセスとして、ＥＣＳ８２４の機能を利用する場合のサーバスケルトンおよびクライアントスタブを生成する処理について説明する。
【００５９】
図９は、サーバプロセスとしてＥＣＳ８２４の機能を利用する場合のメッセージファイルの一例を示す説明図である。図９に示すように、メッセージファイルには、ジョブハンドル、ジョブ実行可否通知、ジョブエンド通知等のメッセージ定義を行うとともに、関数宣言として、ジョブオープン要求、ジョブ動作モード設定、ジョブエントリー要求、ジョブスタート要求、ジョブクローズ要求等の関数宣言を行う。
【００６０】
そして、実施の形態１と同様に、このメッセージファイルをスタブジェネレータに入力して実行すると、図１０に示すクライアントスタブとサーバスケルトンが生成され、図１１に示すヘッダが生成される。図１０に示すように、サーバスケルトンには、ジョブオープン要求、ジョブ動作モード設定、ジョブエントリー要求、ジョブスタート要求、ジョブクローズ要求に対する関数ハンドラが実装記述部を空欄の状態で記述され、クライアントスタブには、これらの関数ハンドラコールが記述される。また、図１１に示すように、ヘッダにはジョブハンドル、ジョブ実行可否通知、ジョブエンド通知等のメッセージが登録される。
【００６１】
そして、このように生成されたクライアントスタブとサーバスケルトンとヘッダから、実施の形態１と同様にクライアントプログラムとサーバプログラムを生成する。たとえば、クライアントプログラムとしてコピーアプリ８１２を利用する場合には、コピーアプリ８１２のソースファイルにジョブオープン要求、ジョブ動作モード設定、ジョブエントリー要求、ジョブスタート要求、ジョブクローズ要求等の関数コールを記述してコンパイルおよびクライアントスタブオブジェクトとのリンクを行ってコピーアプリ８１２を生成する。
【００６２】
これにより、コピーアプリ８１２からＥＣＳ８２４に対して、コピー動作のジョブに関するジョブオープン要求、ジョブ動作モード設定、ジョブエントリー要求、ジョブスタート要求、ジョブクローズ要求を関数コールとしたコピーアプリ８１２とＥＣＳ８２４との間のプロセス間通信を実現することができる。また、ＥＣＳ８２４からコピーアプリ８１２に対してジョブ実行可否通知、ジョブエンド通知等のメッセージによるプロセス間通信が可能となる。
【００６３】
また、コントロールサービス間のプロセス間通信としては、たとえばＦＣＳ８２７のソースファイルにジョブオープン要求、ジョブ動作モード設定、ジョブエントリー要求、ジョブスタート要求、ジョブクローズ要求等の関数コールを記述して、同様にコンパイルおよびクライアントスタブオブジェクトとのリンクを行ってＦＣＳ８２７を生成すると、フ

ァクシミリ動作に関するジョブオープン要求、ジョブ動作モード設定、ジョブエントリー要求、ジョブスタート要求、ジョブクローズ要求を関数コールとしたＦＣＳ８２７とＥＣＳ８２４との間のプロセス間通信を実現することができる。

【００６４】

次に、ＦＣＳ８２７をサーバプロセスとして、ＦＣＳ８２７の機能を利用する場合のサーバスケルトンおよびクライアントスタブを生成する処理について説明する。この場合にも、ＥＣＳ８２４をサーバプロセスとする場合と同様にメッセージファイルに関数宣言とメッセージ定義を行う。ＦＣＳ８２４をサーバプロセスとする場合には、送信スタート、送信モード変更等の関数宣言を行うとともに、スキャンパラメータ通知等のメッセージ定義を行う。

【００６５】

そして、実施の形態１と同様に、このメッセージファイルをスタブジェネレータに入力して実行すると、クライアントスタブとサーバスケルトンとヘッダとが生成される。ＥＣＳ８２４をサーバプロセスとした場合と同様に、サーバスケルトンには、関数宣言を行った送信スタート、送信モード変更に対する関数ハンドラが実装記述部を空欄の状態で記述され、クライアントスタブには、これらの関数ハンドラコールが記述される。また、ヘッダにはメッセージ定義を行ったスキャンパラメータ通知等のメッセージが登録される。

【００６６】

そして、このように生成されたクライアントスタブとサーバスケルトンとヘッダから、実施の形態１と同様にクライアントプログラムとサーバプログラムを生成する。たとえば、クライアントプログラムをＥＣＳ８２４とした場合のプロセス間通信としては、ＥＣＳ８２４のソースファイルに送信スタート、送信モード変更等の関数コールを記述して、同様にコンパイルおよびクライアントスタブオブジェクトとのリンクを行ってＥＣＳ８２４を生成すると、ファクシミリ動作に関する送信スタート、送信モード変更を関数コールとしたＥＣＳ８２４とＦＣＳ８２７との間のプロセス間通信を実現することができる。また、ＦＣＳ８２７からＥＣＳ８２４に対してスキャンパラメータ通知等のメッセージによるプロセス間通信が可能となる。

【００６７】

次に、ＭＣＳ８２５をサーバプロセスとして、ＭＣＳ８２５の機能を利用する場合のサーバスケルトンおよびクライアントスタブを生成する処理について説明する。この場合にも、ＥＣＳ８２４をサーバプロセスとする場合と同様にメッセージファイルに関数宣言とメッセージ定義を行う。

【００６８】

ＭＣＳ８２５をサーバプロセスとする場合には、メッセージファイルにメモリ画像情報要求、ファイル生成要求、ページ生成要求、ページ情報登録、ページオープン要求、ページクローズ要求、ページ情報要求、ページ削除要求、ファイル情報登録、ファイルクローズ要求、ファイルオープン要求、ファイル削除要求、ワークエリア獲得要求、ワークエリア削除要求、分割読み出し要求等の関数宣言を行うとともに、ジョブエンド通知等のメッセージ定義を行う。

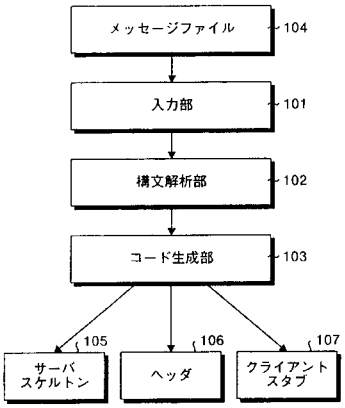
【００６９】

そして、実施の形態１と同様に、このメッセージファイルをスタブジェネレータに入力して実行すると、クライアントスタブとサーバスケルトンとヘッダとが生成される。ＥＣＳ８２４をサーバプロセスとした場合と同様に、サーバスケルトンには、関数宣言を行ったメモリ画像情報要求、ファイル生成要求、ページ生成要求、ページ情報登録、ページオープン要求、ページクローズ要求、ページ情報要求、ページ削除要求、ファイル情報登録、ファイルクローズ要求、ファイルオープン要求、ファイル削除要求、ワークエリア獲得要求、ワークエリア削除要求、分割読み出し要求に対する関数ハンドラが実装記述部を空欄の状態で記述され、クライアントスタブには、これらの関数ハンドラコールが記述される。また、ヘッダには、メッセージ定義を行ったジョブエンド通知等のメッセージが登録される。

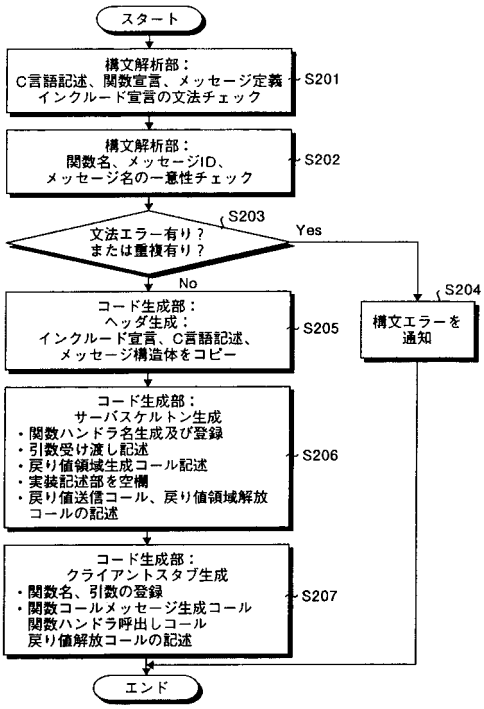
【００７０】
そして、このように生成されたクライアントスタブとサーバスケルトンとヘッダから、実施の形態１と同様にクライアントプログラムとサーバプログラムを生成する。たとえば、クライアントプログラムをＥＣＳ８２４とした場合のプロセス間通信としては、ＥＣＳ８２４のソースファイルにメモリ画像情報要求等の関数コールを記述して、ＥＣＳ８２４を生成すると、プリンタ動作におけるメモリ確保のためメモリ画像情報要求を関数コールとしたＥＣＳ８２４とＭＣＳ８２５との間のプロセス間通信を実現することができる。
【００７１】
また、クライアントプログラムをスキャナアプリ８１４とした場合のプロセス間通信としては、たとえばスキャナアプリ８１４のソースファイルにファイル生成要求、ページ情報登録、ページオープン要求、ページクローズ要求、ページ情報要求、ページ削除要求、ファイル情報登録、ファイルクローズ要求、ファイルオープン要求、ファイル削除要求、ワークエリア獲得要求、ワークエリア削除要求、分割読み出し要求等の関数コールを記述して、スキャナアプリ８１４を生成すると、スキャナ動作における各要求を関数コールとしたスキャナアプリ８１４とＭＣＳ８２５との間のプロセス間通信を実現することができる。
【００７２】
さらに、クライアントプログラムをＥＣＳ８２４とした場合のプロセス間通信としては、たとえばＥＣＳ８２４のソースファイルにページ生成要求、ページ情報登録、ページクローズ要求、ページ情報要求等の関数コールを記述して、ＥＣＳ８２４を生成すると、複数ページのスキャナ動作における各要求を関数コールとしたＥＣＳ８２４とＭＣＳ８２５との間のプロセス間通信を実現することができる。
【００７３】
このように実施の形態２のスタブジェネレータでは、メッセージファイルに関数宣言を記述することにより、アプリケーションなどのクライアントスタブ１０７とコントロールサービスなどのサーバスケルトン１０５を容易に生成することができる。また、スタブジェネレータで生成されるサーバスケルトンの実装記述部は空欄となっているので、プログラム開発者は必要に応じて関数の処理内容を容易に変更することができ、システムに可変性を持たせることができる。さらに、アプリケーションプログラムの開発を、サードベンダーなどの他社が行う場合でも、アプリケーションプログラムには提供される関数コールを記述するだけでプロセス間通信が可能となるので、内部の通信プロトコルの隠蔽性を保つことができる。
【００７４】
また、実施の形態２のスタブジェネレータでは、コントロールサービス間またはコントロールサービスとアプリケーション間のプロセス間通信を関数コールで実現することができ、ジョブの並列実行をプロセス内の複数のスレッドを起動させることにより実現した場合でも、高速なプロセス間通信が可能となる。
【００７５】
さらに、実施の形態２のスタブジェネレータでは、関数からの戻り値によってスレッド間の同期をとることができるので、同期処理を別途設ける必要がなくなり、プログラム開発の労力を低減することができる。
【図面の簡単な説明】
【００７６】
【図１】この発明の実施の形態１のスタブジェネレータの機能的構成を示すブロック図である。
【図２】実施の形態１のスタブジェネレータにおけるサーバスケルトン、クライアントスタブおよびヘッダの生成処理のフローチャートである。
【図３】実施の形態１のスタブジェネレータに入力するメッセージファイルの記述例を示す説明図である。
【図４】実施の形態１のスタブジェネレータにより生成されるクライアントスタブの一例

を示す説明図である。
【図５】実施の形態１のスタブジェネレータにより生成されるサーバスケルトンの一例を示す説明図である。
【図６】実施の形態１のスタブジェネレータにより生成されるヘッダの一例を示す説明図である。
【図７】実施の形態１において、クライアントプログラムから関数コールを行った場合におけるサーバプログラムとの呼び出しおよび応答のシーケンスを示す説明図である。
【図８】実施の形態２のスタブジェネレータで生成したサーバプロセス、クライアントプロセスが動作する複合機のソフトウェア構成を示すブロック図である。
【図９】実施の形態２において、サーバプロセスとしてＥＣＳの機能を利用する場合のメッセージファイルの一例を示す説明図である。
【図１０】実施の形態２のスタブジェネレータで生成されたクライアントスタブとサーバスケルトンとヘッダの一例を示す説明図である。
【図１１】実施の形態２のスタブジェネレータで生成されたヘッダの一例を示す説明図である。
【符号の説明】
【００７７】
１０１　入力部
１０２　構文解析部
１０３　コード生成部
１０４　メッセージファイル
１０５　サーバスケルトン
１０６　ヘッダ
１０７　クライアントスタブ
８００　複合機
８０１　白黒ラインプリンタ
８０２　カラーラインプリンタ
８０３　ハードウェアリソース
８１０　ソフトウェア群
８１１　プリンタアプリ
８１２　コピーアプリ
８１３　ファックスアプリ
８１４　スキャナアプリ
８１５　ネットファイルアプリ
８１６　工程検査用アプリ
８２０　プラットホーム
８２１　汎用ＯＳ
８２２　ＳＣＳ
８２３　ＳＲＭ
８２４　ＥＣＳ
８２５　ＭＣＳ
８２６　ＯＣＳ
８２７　ＦＣＳ
８２８　ＮＣＳ
８３０　アプリケーション
８２７　ＦＣＳ

【図１】



【図２】



【図３】

```
xcs.msg

 /* インクルード宣言  */
 #include <ecs_2.msg>     /* メッセージファイルインクルード  */
 #include "common.h"      /* C 言語記述  */
 #include "xcs_def"

/* メッセージ定義  */
message {
          name=END
          id = 0x01
          dir = OUT
          content [[[
                    ID id;
                    u_char Dev;
                    u_long EndStatus;
        ]]];
};
message {
          name=STOP
          id = 0x02
          dir = INOUT
          content [[[
                    ID Jobid;
                    u_char Dev;
                    u_long StopStatus;
          ]]];
};
            .
            .
            .
/* 関数宣言  */
function [3] int Open(a, b, c)
function [4] int Close(a, b, c)
          .
          .
```

【図４】

```
xcs_stub.c

#include <thread.h>
#include <xcs.h>    /* ヘッダのインクルード */

/* 関数コール  */
int Open(a, b, c)
{ .
  .
get_function(...);              /* 関数コールメッセージ生成コール */
ret = func_call(int Open_handler);   /* 関数ハンドラ呼び出し */
free_ret(...);                  /* 戻り値解放コール */
}
int Close(a, b, c)
{ .
  .
get_function(...);              /* 関数コールメッセージ生成コール */
ret = func_call(int Close_handler);  /* 関数ハンドラ呼び出し */
free_ret(...);                  /* 戻り値解放コール */
}
  .
  .
```

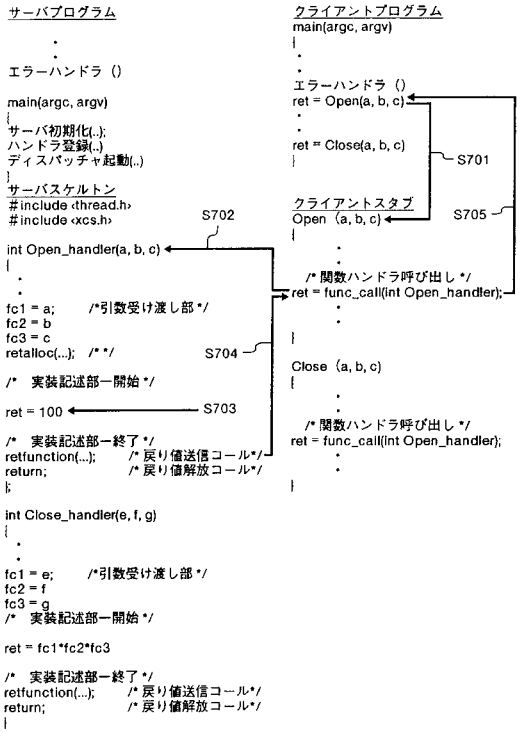
【図５】

```
xcs_skel.c

#include <thread.h>
#include <xcs.h>    /* ヘッダのインクルード */

int Open_handler(a, b, c)
{
  .
  .
fc1 = a;       /*引数受け渡し部 */
fc2 = b
fc3 = c
retalloc(...);   /* */
/*   実装記述部一開始 */


/*   実装記述部一終了 */
retfunction(...);       /* 戻り値送信コール*/
return;                 /* 戻り値解放コール*/
};

int Close_handler(e, f, g)
{
  .
  .
fc1 = e;       /*引数受け渡し部 */
fc2 = f
fc3 = g
/*   実装記述部一開始 */


/*   実装記述部一終了 */
retfunction(...);       /* 戻り値送信コール*/
return;                 /* 戻り値解放コール*/
}
```

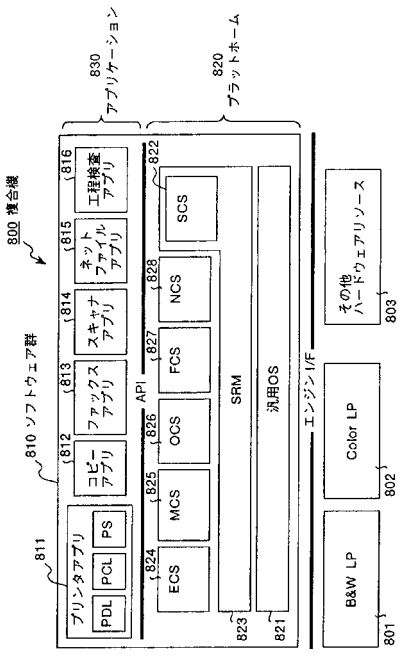
【図６】

```
xcs.h

/* インクルード宣言   */
#include "common.h"      /* C言語記述   */
#include "xcs_def"

/* メッセージＩＤ   */
#define END       1
#define STOP      2
        .
        .
/* メッセージ構造体   */
typedef struct msg_END {
                  ID id;
                  u_char Dev;
                  u_long EndStatus;
};
typedef struct msg_STOP {
                  ID id;
                  u_char Dev;
                  u_long StopStatus;
};
        .
        .

/* 関数宣言   */
int Open(a, b, c)
int Close(a, b, c)
        .
        .
/* 関数ハンドラ宣言   */
int Open_handler(a, b, c)
int Close_handler(a, b, c)
        .
        .
```

【図７】



【図８】



【図９】

ecs.msg

```
/* メッセージ定義　*/
message [/* ジョブハンドル*/
        name=JHDL
        id = 0x01
        dir = OUT
        content [ [ [
          ・
];
message [/* ジョブ実行可否通知*/
        name=JEXE
        id = 0x02
        dir = OUT
        content [ [ [
          ・
];
message [/* ジョブエンド通知*/
        name=JEND
        id = 0x03
        dir = OUT
        content [ [ [
          ・
];
        ・
        ・
        ・
/* 関数宣言　*/
function [3] int JOpen(...)  /* ジョブオープン要求　*/
function [4] int JMode(...)  /* ジョブ動作モード設定　*/
function [5] int JEntry(...)  /* ジョブエントリー要求　*/
function [6] int JStart(...)  /* ジョブスタート要求　*/
function [7] int JClose(...)  /* ジョブクローズ要求　*/
      ・
      ・
```

【図１０】

ecs_stub.c　（クライアントスタブ）

```
#include <ecs.h>　/* ヘッダのインクルード */

/* 関数コール　*/
int JOpen(...)
{ ・
ret = func_call(int JOpen_handler);　/* 関数ハンドラ呼び出し */
  ・}
int JMode(...)
{ ・
ret = func_call(int JMode_handler);　/* 関数ハンドラ呼び出し */
  ・}
int JEntry(...)
{ ・
ret = func_call(int JEntry_handler);　/* 関数ハンドラ呼び出し */
  ・}
int JStart(...)
{ ・
ret = func_call(int JStart_handler);　/* 関数ハンドラ呼び出し */
  ・}
```

xcs_skel.c　（サーバスケルトン）

```
#include <xcs.h>　/* ヘッダのインクルード */

int JOpen_handler(...)
{ ・
/*　実装記述部ー開始 */

/*　実装記述部ー終了 */
  ・};

int JMode_handler(...)
{ ・
/*　実装記述部ー開始 */

/*　実装記述部ー終了 */
  ・};
int JEntry_handler(...)
{ ・
/*　実装記述部ー開始 */

/*　実装記述部ー終了 */
  ・};
int JStart_handler(...)
{ ・
/*　実装記述部ー開始 */

/*　実装記述部ー終了 */
  ・};
```

【図１１】

ecs.h (ヘッダ)

```
/* メッセージＩＤ　*/
#define JEND      1
#define JEXE      2
#define JHDL      3
      ・
      ・

/* 関数宣言　*/
int JOpen(...)
int JMode(...)
int JEntry(...)
int JStart(...)
      ・
      ・
/* 関数ハンドラ宣言　*/
int JOpen_handler(...)
int JMode_handler(...)
int JEntry_handler(...)
int JStart_handler(...)
```