5P-4　CTRON/UNIXインタフェーサにおけるプロセス制御

竹内　宏典　　工藤　明彦

（ＮＴＴ情報通信処理研究所）

1　はじめに

ＣＴＲＯＮ*1仕様に準拠したリアルタイムＯＳ（以下、ＣＴＲＯＮと略す）上で既存プログラムの実行やオンライン処理ＡＰの開発を可能とする方法として、既存ＯＳの機能を持つインタフェーサをＣＴＲＯＮ上に作成するアプローチがある[1]。筆者らは、汎用小型計算機上に試作されたＣＴＲＯＮ上に、ＵＮＩＸ*2インタフェーサを開発した。ＵＮＩＸインタフェーサは、図１に示すように、基本ＯＳや拡張ＯＳの機能を利用して、ＣＴＲＯＮ上にＵＮＩＸ環境を疑似するものである。

本稿では、ＵＮＩＸのプロセス管理をＣＴＲＯＮ上に実現する方法について述べる。

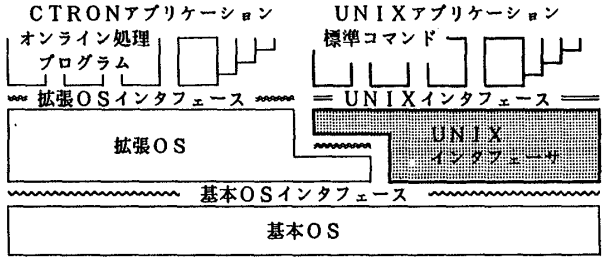

図1　ＵＮＩＸインタフェーサ

2　プロセスの設計

2．1　処理単位

ＵＮＩＸにおける処理の単位はプロセスである。ＵＮＩＸインタフェーサでＵＮＩＸ環境を疑似するためには、プログラムの実行制御をＵＮＩＸのプロセスと等価な処理単位で行う必要がある。また、ＵＮＩＸでは、利用者がプログラムの実行、強制終了、状態参照等を行う場合に、プロセスを意識することから、ＵＮＩＸインタフェーサでの処理単位をＵＮＩＸのプロセスと同等とする必要がある。本稿では、ＵＮＩＸインタフェーサにおける処理の単位を＜プロセス＞と呼ぶ。

ＣＴＲＯＮにおける処理の単位には、「タスク」と「プロセス」がある。「タスク」とは資源を伴わない並列処理の最小単位であり、タスクを実行するための資源の割り付けや解放は利用者が行う必要がある。「タスク」は、ＣＴＲＯＮ基本ＯＳのタスク管理機能により制御される処理単位である。一方、「プロセス」とは資源を伴った並列処理の単位であり、１つまたは複数の「タスク」とメモリやファイル等の資源で構成される。「プロセス」は、ＣＴＲＯＮ拡張ＯＳのプロセス管理機能により制御される処理単位である。

2．2　プロセスの構成

ＵＮＩＸインタフェーサの＜プロセス＞を構成する方法として、次の２案が考えられる。

Ａ．ＣＴＲＯＮの「プロセス」により構成する

Ｂ．ＣＴＲＯＮの「タスク」および資源により構成する

案Ａでは、スケジュール、資源の割り付け等は、ＣＴＲＯＮ拡張ＯＳのプロセス管理に委ねるため、ＵＮＩＸインタフェーサは利用者インタフェース機能を実装すればよい。ただし、プロセス管理機能について、ＣＴＲＯＮとＵＮＩＸの相違を整合させる必要はある。

案Ｂでは、スケジュールはＣＴＲＯＮ基本ＯＳのタスク管理に委ねるが、資源の割り付け等はＵＮＩＸインタフェーサで行う必要がある。すなわち、ＵＮＩＸインタフェーサ内に、ＣＴＲＯＮ拡張ＯＳのプロセス管理とは独立したプロセス管理機能を構築する必要がある。

2．3　プロセス管理機能の構築

まず、案Ａについて検討するために、プロセス管理機能に関してＣＴＲＯＮとＵＮＩＸの比較を行った[2][3]。これを表１に示す。表１より、以下の点が問題となる。

①ＵＮＩＸのプロセス生成（fork）機能を実現するためには、親プロセスのコンテクストと資源をそのまま子プロセスに引き継ぐ必要があるが、ＣＴＲＯＮの「プロセス」生成機能では、「プロセス」に対して新たにプログラムと資源の割り付けが行われる。プログラムと資源の割り付けは、ＵＮＩＸではプロセス起動（exec）時に行われるものであり、プロセス生成（fork）とは明確に分離されなければならない。

②ＵＮＩＸにおけるプロセスの親子関係を実現するためには、あるプロセスＰの親プロセスが終了した場合に、プロセスＰの親プロセスをinitプロセスに変更する機能が必要であるが、ＣＴＲＯＮでは「プロセス」Ｐの親「プロセス」を他の「プロセス」に変更できない。

③ＵＮＩＸのプロセスのユーザＩＤは、プロセスの親子関係と密接に関連する。すなわち、ログイン時を除き、あるプロセスから生成される全ての子孫プロセスに対してユーザＩＤが引き継がれる必要がある。しかし、ＣＴＲＯＮでは、「プロセス」の親子関係とユーザＩＤは、何の関係も持たない。

④ＵＮＩＸのプロセスのデータ域拡張（sbrk）機能を実現するためには、プロセス実行中に動的に論理空間の割り付けを行い、かつ、割り付け後の論理空間は連続していなければならない。ＣＴＲＯＮの「プロセス」のデータ域への論理空間の割り付けは、「プロセス」生成時に行われ、「プロセス」実行中に拡張することはできない。

上記の問題点のうち、②、③について解決するためには、ＵＮＩＸインタフェーサで親子関係、ユーザＩＤの引き継ぎ等を処理するように、ＵＮＩＸインタフェーサ内にプロセス管理機能の一部を構築すればよい。しかし、①、④については、ＵＮＩＸインタフェーサでは対処不可能である。

以上より、案Ｂを採用する。

3　プロセスの制御

3．1　プロセスの生成（fork）

ＵＮＩＸインタフェーサ上の＜プロセス＞の生成は、Ｃ

Implementation of UNIX Process Control on CTRON Architecture

Hironori TAKEUCHI and Akihiko KUDOH
NTT Communications and Information Processing Laboratories

*1 ＣＴＲＯＮは、Central and Communication TRON (the realtime operating system nucleous) の略称である。

*2 ＵＮＩＸはＡＴ＆Ｔが開発し、ライセンスしているオペレーティングシステムである。

表1　UNIXとCTRONのプロセス管理機能の比較

UNIX	CTRON
fork（プロセス生成） 自プロセスのコンテクストをコピーしてプロセスを生成する。 自プロセスの資源を生成したプロセスに引き継ぐ。 自プロセスが親となり、生成したプロセスは子となる。 生成したプロセスはそのまま実行される。	CRE_PRC（プロセス生成） プロセスを生成し、プログラムを割り付ける。 生成したプロセスに資源を割り付ける。 生成したプロセスの親子（主従）属性は指定による。 生成したプロセスはDormant状態となる。
exec（プロセス起動） 自プロセスに新たにプログラムを割り付ける。 自プロセスに新たに資源を割り付ける。 割り付けたプログラムを実行する。	STA_PRC（プロセス起動） 指定したDormant状態のプロセスを実行する。
exit（プロセス終了） プロセスグループに属するプロセスに終了通知をする。 自プロセスの子プロセスをinitプロセスの子プロセスとする。 自プロセスの資源を解放する。 自プロセスの親プロセスに終了通知をする。	EXT_PRC（プロセス終了） プロセスを終了する（Dormant状態に戻す）。 プロセス終了の待ち合わせ 終了通知有を指定し生成したプロセスからは終了通知がある。
wait（プロセス終了の待ち合わせ／プロセス消滅） 子プロセスの終了通知を待ち合わせ、終了通知のあった子プロセスを消滅させる。	DEL_PRC（プロセス消滅） 指定したDormant状態のプロセスの資源を解放する。 消滅するプロセスが終了同期を指定し生成したプロセスも消滅する。 消滅するプロセスが親プロセスであれば、そのプロセスグループに属するプロセスは消滅する。 消滅するプロセスの子プロセスは、消滅するプロセスの親プロセスの子プロセスとなる。
brk／sbrk（プロセスデータ域拡張） プロセス実行中にデータ域を拡張（／縮小）できる。	

TRONの「タスク」を生成することにより実現する。このとき、自<プロセス>（すなわち、自「タスク」）のコンテクストをコピーして、子<プロセス>（子「タスク」）のコンテクストとしなければならない。しかし、CTRON基本OSのタスク管理のタスク生成機能ではコンテクストのコピーは行われない。そこで、CTRON基本OSに、タスクのコンテクストをコピーし、タスクを生成するタスク複製機能を個別機能として追加した。

ファイル等資源の引き継ぎ、プロセスIDの付与などは、UNIXインタフェーサ内に構築したプロセス管理機能で行う。これは、UNIXと同等である。

なお、「タスク」が生成（複製）されるとタスクIDが得られる。タスクIDは、CTRON基本OSのタスク管理機能を利用する際に指定する必要があるので、UNIXインタフェーサで付与するプロセスIDと対に管理する。

3．2　プロセスの起動（exec）

UNIXインタフェーサ上の<プロセス>の起動は、CTRONの「タスク」を起動することにより実現する。これは、CTRON基本OSのタスク管理のタスク起動機能を利用した。

<プロセス>への新たな資源の割り付け、プログラムのロードなどは、UNIXインタフェーサ内に構築したプロセス管理機能で行う。これは、UNIXと同等である。

3．3　プロセスの終了（exit）／待ち合わせ（wait）

UNIXインタフェーサ上の<プロセス>の終了は、CTRONの「タスク」を終了させることにより実現する。これは、CTRON基本OSのタスク管理のタスク終了機能を利用した。

プロセスグループに属するプロセスへのsignal（SIGTERM）通知、資源の解放、自<プロセス>の子<プロセス>の親子関係の付け替え、親<プロセス>へのsignal（SIGCLD）通知などは、UNIXインタフェーサ内に構築したプロセス管理機能、signal機構[4]により行う。

UNIXインタフェーサ上の<プロセス>終了の待ち合わせは、CTRONの「タスク」を消滅させる。これは、waitシステムコールの発行を契機にCTRON基本OSのタスク管理のタスク消滅機能を利用した。

子<プロセス>実行中のwaitシステムコールでは、子<プロセス>が終了するまで、自<プロセス>をsleep させるが、これは、UNIXインタフェーサ内に構築したプロセス管理機能で行い、UNIXと同等である。

3．4　プロセススケジュール

プロセスのスケジュールは、「タスク」のスケジュールとしてCTRON基本OSのタスク管理に委ねた。しかし、UNIXインタフェーサ上の<プロセス>は、CTRONのオンライン処理と同時に走行するので、オンライン処理のCPU資源を圧迫しないようにスケジュールされなければならない。基本OSタスク管理が行うスケジュールは、「タスク」の実行レベルと優先度に基づき行われるので、UNIXインタフェーサでは、<プロセス>の実行レベルをCTRONのオンライン処理の実行レベルより低くした。

また、UNIXインタフェーサ上の<プロセス>相互については、nice値と基本OSへの問い合わせで得られるCPUの消費時間を用いて<プロセス>のプライオリティを計算し、求められたプライオリティを優先度にマッピングした。

このようにして、UNIXインタフェーサの<プロセス>よりCTRONのオンライン処理を優先し、かつ、UNIXインタフェーサの<プロセス>相互についてはUNIXと同等のスケジュールを実現した。

4　まとめ

UNIXインタフェーサでは、<プロセス>を「タスク」に対応付け、CTRON基本OSのタスク管理機能を利用して、UNIXと同等のプロセス管理機能を実現した。

また、<プロセス>の生成処理に関して、親<プロセス>のコンテクストのコピーを可能とするためには、CTRONの機能が不足していることが明かとなった。この機能は、UNIXインタフェーサを実現するためだけではなく、CTRON上で一つのプログラム内での並列処理を実現するためにも必要であると考えられる。

参考文献

[1]ﾄﾛﾝ協会編;「CTRON概説」ｵｰﾑ社(1988)
[2]M.J.Bach; The Design of the UNIX Operating System Prentice-Hall, Inc. (1986)
[3]和佐野，小林;「情報通信ﾈｯﾄﾜｰｸ向きOSｲﾝﾀﾌｪｰｽCTRON」情報処理 VOL.30 No.5 (1989) pp. 553～564
[4]工藤，竹内;「CTRON/UNIXｲﾝﾀﾌｪｰｻにおけるsignalの実現」本大会