システムソフトウェアと
オペレーティング・システム 89－8
シ ス テ ム 評 価 2－8
（２００２．２．１４）

並列分散オペレーティングシステムCEFOSにおける準プリエンプション機能

棚林 拓也†[1]　中山 大士†[1]　日下部 茂‡[2]　谷口 秀夫‡[2]　雨宮 真人‡[1]

†九州大学 大学院 システム情報科学府
‡九州大学 大学院 システム情報科学研究院

[1] 〒816-8580 福岡県 春日市 春日公園 6-1
[2] 〒812-8581 福岡県 福岡市 東区 箱崎 6-10-1
tana@al.is.kyushu-u.ac.jp

あ ら ま し

プロセス切り替えの処理負荷を削減し、また、プロセス切り替え回数を削減する、準プリエンプション機能を提案する。準プリエンプション機能とは、優先度逆転が生じても、直ちにはプロセス切り替えを行わず、現在走行しているプロセスの適当な切れ目を契機として優先度逆転を解消するためのプロセス切り替えを行う機能である。本論文では、準プリエンプション機能について説明し、その効果の見積もりをしたのち、CEFOS における実装方式について述べる。

キーワード　オペレーティングシステム, 並列分散処理, 細粒度マルチスレッディング, プロセス切り替え

Semi-preemption mechanism of CEFOS

Takuya TANABAYASHI†[1], Hiroshi NAKAYAMA†[1], Shigeru KUSAKABE‡[2],
Hideo TANIGUCHI‡[2], and Makoto AMAMIYA‡[1]

Graduate School of Information Science and Electrical Engineering, Kyushu University

[1]6-1 Kasuga-koen, Kasuga, Fukuoka, Japan, 816-8580
[2]6-10-1 Hakozaki, Higashi-ku, Fukuoka, Fukuoka, Japan, 812-8581
tana@al.is.kyushu-u.ac.jp

Abstract

We propose Semi-preemption mechanism which reduces the overhead of a process switch and reduces the number of process switches. Semi-preemption mechanism does not perform a process switch immediately, even if a priority inversion arises. Instead, it performs a process switch at the suitable point of the current process, in order to resolve a priority inversion. This paper describes the design of Semi-preemption mechanism. We also estimate the effectiveness of the mechanism and discuss implementation issues in CEFOS.

Key words　Operating System, Parallel and Distributed Computing, Fine-grain Multi-threading, Process Switch

1 はじめに

近年、計算機の性能は著しく向上し、取り扱う情報は莫大な量となっている。それにともない、入出力装置の性能も向上し、ネットワーク装置では1Gbps以上のバンド幅を持つものが安価に提供されている。このような高性能の入出力装置の応答時間は非常に短いため、有効に利用するためには頻繁に入出力操作を行う必要がある。

一般に、プロセスが入出力要求を行う時、入出力装置の処理が完了するまでプロセッサを必要としない場合、他のプロセスがプロセッサで演算処理を行えるようにするためにプロセッサを放棄し、入出力装置からの結果が得られるまで休眠する。したがって、入出力の頻度の増加はプロセスの休眠/起床回数の増加を伴い、プロセス切り替え回数が増加する。

また、近年の高速なプロセッサは、主として連続した処理を高速に実行するように設計されており、プロセス切り替え処理のような非連続な処理は、相対的に時間のかかる処理である。

したがって、プロセス切り替えが頻発すると、システムのスループットが低下する恐れがある。このため、プロセス切り替え回数を削減し、プロセス切り替え処理を効率よく実行することは重要である。

そこで、我々は、プロセス切り替えの処理負荷を削減し、また、プロセス切り替え回数を削減する、準プリエンプション機能を提案する。

本論文では、準プリエンプション機能について述べ、その効果の見積もりをしたのち、並列分散オペレーティングシステム CEFOS[1, 2] における実装方式について述べる。

2 準プリエンプション機能

2.1 基本方式

従来のプリエンプション方式では、優先度の高いプロセスが起床されると、直ちにプロセスの切り替えを行う。

従来のプリエンプション方式による場合の処理の流れを図1に示す。従来のプリエンプション方式では、割り込みが発生 (1) するとカーネルに制御が移行し、対応する割り込み処理が行われる。割り込み処理の結果、プロセス A が起床されると (2)、プロセス優先度の確認が行われる。現走行プロセス (プロセス B) より優先度が高い場合、プロセススケジューリングが行われ、優先度の高いプロセスに切り替えられる (ディスパッチ)(3)。

このように、従来のプリエンプション方式では、優先度逆転が生じると直ちにプロセスを切り替える。しかし、プロセスが頻繁に起床するような環境では、プロセス切り替えが多発し、切り替え処理の負荷や、切り替えによるプロセッサ効率の低下によって、全体のスループットに悪影響をおよぼす。

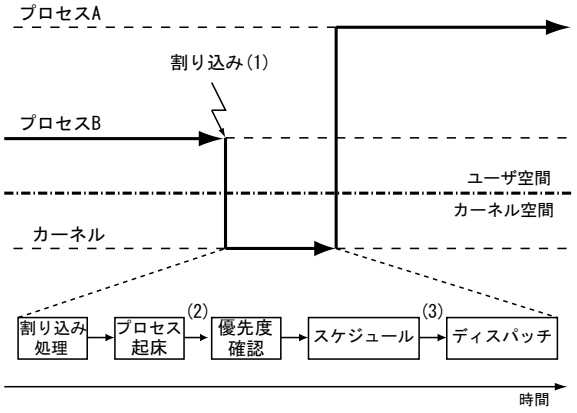

図 1: プリエンプションの処理の流れ

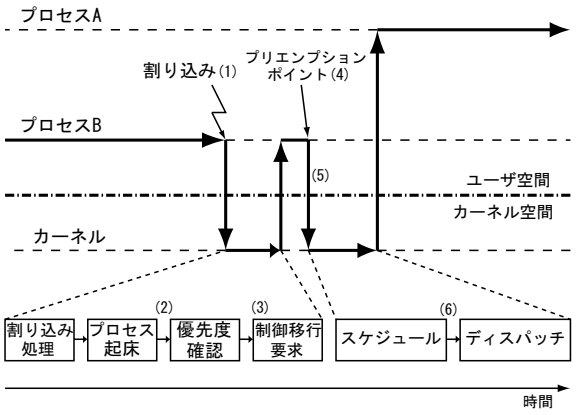

図 2: 準プリエンプションの処理の流れ

そこで、優先度逆転によるプロセス切り替えを遅延することによってプロセス切り替えの処理負荷を削減し、また、プロセス切り替え回数を削減する準プリエンプション機能を提案する。

準プリエンプション機能とは、優先度逆転が生じても、直ちにはプロセス切り替えを行わず、現在走行しているプロセスの適当な切れ目を契機としてプロセス切り替えを行い、優先度逆転を解消する機能である。ここで、「適当な切れ目」とは、今まで行ってきた処理と異なる処理を開始する個所であり、たとえば、ユーザーレベルスレッド実装でのスレッド切り替え処理といったものが考えられる。以降、「適当な切れ目」をプリエンプションポイントと呼ぶ。

準プリエンプション方式による場合の処理の流れを図2に示す。準プリエンプション方式でも、プロセス優先度の確認までは従来のプリエンプション方式と同様の処理を行う。しかし、割り込み処理の結果起床したプロセスが現走行プロセスより優先度が高い場合でも、プロセスの切り替えは行わない。この場合、現走行プロセスに対し、カーネルに制御を移行するよう要求 (制御移行

要求)(3) を出し、プロセスの処理を再開する。現走行プロセスはプリエンプションポイント (4) において制御移行要求の確認を行い、必要に応じてカーネルに制御移行 (5) を行う。

2.2 期待される効果と問題点

準プリエンプション方式では、次のような効果が期待される。

効果 1 プロセス切り替えオーバヘッドの削減

効果 2 プロセス切り替え回数の削減

一方、準プリエンプション方式を用いると、次のような問題が生じる恐れがある。

問題 1 応答時間の長大化

それぞれの詳細について、以降、説明する。

2.2.1 プロセス切り替えオーバヘッドの削減

準プリエンプション方式を用いると、再開に必要なコンテキストの待避/回復処理の一部を省略することができ、プロセス切り替えオーバヘッドを削減できる。

従来のプリエンプション方式では、プロセスは任意の場所で切り替えが行われる可能性があるため、カーネルは、すべてのコンテキストの待避/回復処理を行う必要がある。

一方、準プリエンプション方式では、プロセスはプリエンプションポイントにおいてのみ切り替えられるため、カーネルは最低限のコンテキストのみ待避/回復処理を行えば良い。その他のコンテキストについては、必要に応じてプロセス側で待避/回復処理を行う。待避/回復処理を省略可能なコンテキストとしては、汎用レジスタといったものが考えられる。

2.2.2 プロセス切り替え回数の削減

準プリエンプション方式では、優先度逆転によるプロセス切り替えを遅延し、その間に起床したプロセスを起床順ではなく、優先度順に実行することによって、プロセス切り替え回数を削減する。この様子を図 3 に示す。

プロセス C 走行中に、よりプロセス優先度が高いプロセス B が起床し、さらにプロセス優先度が高いプロセス A が起床する場合を考える。

従来のプリエンプション方式の場合、プロセス B が起床すると、直ちにプロセス切り替えが行われ、プロセス B が実行される。プロセス B の処理が完了する前にプロセス A が起床すると、さらにプロセス切り替えが行われ、プロセス A が実行される。プロセス A の処理完了後、プロセス B の処理が再開され、プロセス B の処理完了後、プロセス C の処理が再開される。この例の場合、プロセス切り替え回数は 4 回となる。

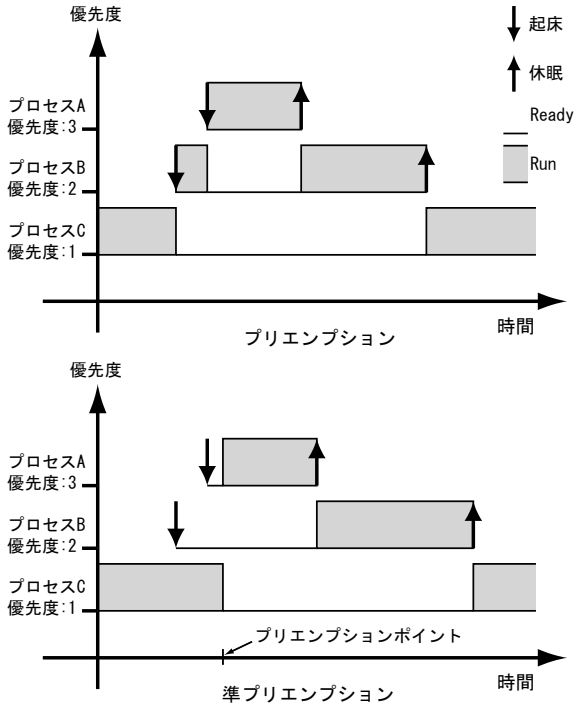

図 3: プロセス切り替え回数の比較

準プリエンプション方式の場合、プロセス B の起床後、プロセス A が起床するのがプリエンプションポイントより前である場合、プリエンプション方式よりプロセス切り替え回数が少なくなる。これは、プロセス B の処理開始を遅延し、プロセス A の処理を先行して行ったため、プロセス B がプロセス A によってプリエンプトされるのが回避されたためである。このため、プロセス切り替え回数はプリエンプション方式より 1 回少ない 3 回となる。

プロセス切り替え回数を削減すると、スループットの向上が期待できる。これは、スケジューリング処理、コンテキスト待避/回復処理、メモリ空間切り替え処理等、プロセス切り替えに伴う処理のオーバヘッドを削減できるだけでなく、プロセスの実行がより連続的になることによって参照の局所性が増し、TLB(Translation Look-aside Buffer)、データ/命令キャッシュなどのヒット率向上が見込めるためである。

2.2.3 応答時間の長大化

準プリエンプション方式を用いると、応答時間が長大化する恐れがある。従来のプリエンプション方式では、プロセスの任意の場所でプロセス切り替えを行うことができるので、優先度逆転が生じると直ちにプロセスを切り替え、優先度逆転を解消する。これに対し、準プリエンプション方式では、プロセスのプリエンプションポイントでのみプロセス切り替えを行うので、プロセス切り替えが遅延され、結果として応答時間が長大化するおそ

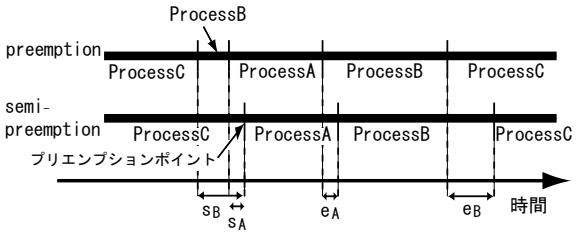

図 4: 他プロセスの影響を受けない場合

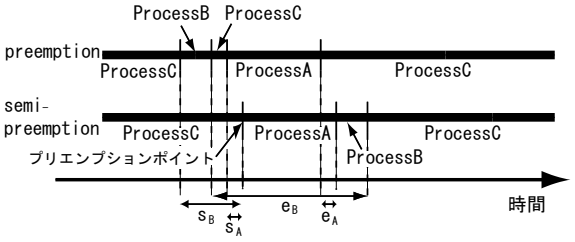

図 5: 他プロセスの影響を受ける場合

れがある。

図 4, 図 5 に、応答時間が長大化する場合の例を示す。これらの例では、3 つのプロセスが存在し、プロセス A の優先度がもっとも高く、プロセス C の優先度がもっとも低いとする。プロセス C の実行中にプロセス B、プロセス A の順に起床し、準プリエンプション方式のプリエンプションポイントは、プロセス A の起床後に通過するものとする。また、プロセス切り替えのオーバヘッドはないものとする。

ここでは、プロセス A の処理時間を c_A、プロセス B の処理時間を c_B、プロセス A の開始遅延時間を s_A、プロセス B の開始遅延時間を s_B、プロセス A の完了遅延時間を e_A、プロセス B の完了遅延時間を e_B と表す。

図 4 は、他プロセスの影響を受けず、準プリエンプション方式による切り替え遅延の影響のみが現れる場合である。この場合、プロセス A、プロセス B の完了遅延時間は

$$e_A = s_A \tag{1}$$

$$e_B = s_B \tag{2}$$

となる。

図 5 は、他のプロセスの影響を受け、応答時間がさらに長大化する場合である。具体的にはプロセス A、プロセス B の完了順序が入れ替わり、プロセス B の応答時間が長大化する場合の例である。この場合、プロセス A の完了遅延時間は図 5 の場合と同様、式 (1) となるが、プロセス B の完了遅延時間は

$$e_B = s_B + c_A \tag{3}$$

となる。

応答時間に関しては 3.3 にて、さらに検討を行う。

3 比較

準プリエンプション方式を用いると、(1) 切り替え当りのオーバヘッド、(2) 切り替え回数、(3) 応答時間、に影響があることが考えられる。そのうち、(1) 切り替え当りのオーバヘッドについては、その効果が実装する計算機や実装方式に依存するので、ここでの比較は行わない。以降、(2) 切り替え回数と、(3) 応答時間について比較を行う。

3.1 モデル

モデルとして、一定の負荷がかかった定常状態を考える。入出力操作を行っている場合、定常状態では、各プロセスの振る舞いは周期的になると考えられる。入出力装置の応答時間が一定であると仮定すると、入出力操作をするプロセスは周期的に休眠/起床を繰り返す。そこで、ここでは周期的に起動されるプロセス (以下、周期プロセス) について考える。

優先度が i 番目のプロセスを P_i とし、起動周期を T_i、各 run あたりの実行時間を C_i とする。図 6 に、このプロセス P_i が実行中に他のプロセスに割り込まれずに実行された場合を示す。プロセス P_i は、時刻 t_1 に起動されると実行可能となり、プロセッサが割り当てられ実行が開始される。プロセスの実行が開始されると、プロセスを構成するスレッドが実行される。実行すべきスレッドの処理が完了すると、プロセスは再び wait 状態になる。このときの時刻は $t_1 + C_i$ である。次にこのプロセスが起動されるのは時刻 $t_1 + T_i$ である。

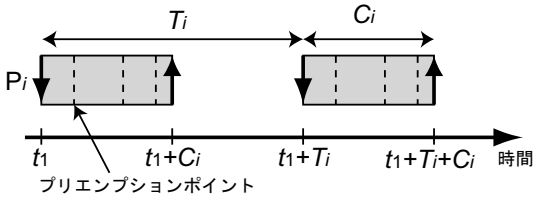

図 6: 周期プロセス

単純化するため、プロセスの集合について、以下のことを仮定する。

(1) 複数の周期プロセスから構成される。

(2) 各プロセスの各 run の実行時間はあらかじめわかっている。

(3) 各プロセスの周期起床が保証されている。

(4) プロセスは各 run の処理が完了するまで、自ら実行を中断することはない。

(5) プロセス切り替えのオーバヘッドはない。

(6) 単一のプロセッサで実行される。

また、静的優先度に基づくスケジューリングの場合で考え、タイムスライスは考えない。

3.2 切替え回数

プロセス切り替えを遅延することによって切り替え回数を削減することができるのは、次の 2 つの場合である。

場合 1 遅延されたプロセス間で、処理開始順序を変更した場合

場合 2 起床した時に走行していたプロセスの処理完了 (Wait 状態になる) を待って、切り替える場合

準プリエンプション方式では、次のように言い換えることができる。

場合 1 プリエンプションポイントを通過してから、次にプリエンプションポイントに到達するまでの区間に、現走行プロセスより優先度の高いプロセスが起床し、その区間の完了までにさらに優先度の高いプロセスが起床した場合

場合 2 プロセスの各 run において、最後のプリエンプションポイントを通過してから、現走行プロセスが wait 状態になるまでの間に、現走行プロセスより優先度の高いプロセスが起床された場合

場合 1 を図 7 に、場合 2 を図 8 に示す。

場合 2 による単位時間あたりの切替え削減回数 S^{case2} を考える。プロセスの各 run における最後のプリエンプションポイントから、プロセスが wait 状態になるまでの間の処理を、以降、終端処理と呼ぶ。プロセス P_i の終端処理の実行時間を $c_{(i,term)}$ とすると、プロセス P_i のある run における終端処理の実行中に、より優先度の高いプロセス $\mathrm{P}_j(j<i)$ が起床する確率は、周期起床が保証されるので、

$$\min\left(\frac{c_{(i,term)}}{T_j},1\right) \tag{4}$$

と表すことができる。したがって、プロセス P_i のある run における終端処理の実行中に、一つでもより優先度の高いプロセスが起床する確率は、

$$1-\prod_{j=1}^{i-1}\left(1-\min\left(\frac{c_{(i,term)}}{T_j},1\right)\right) \tag{5}$$

となる。これは、P_i のある run における終端処理の実行時に削減されたプロセス切り替え回数を表す。

また、プロセス P_i の終端処理が実行されている確率は、

$$\frac{c_{(i,term)}}{T_i} \tag{6}$$

となるので、全プロセス数を N とすると、場合 2 による単位時間あたりの切替え削減回数 S^{case2} は、

$$S^{case2}=\sum_{i=1}^{N}\frac{c_{(i,term)}}{T_i}\left(1-\prod_{j=1}^{i-1}\left(1-\min\left(\frac{c_{(i,term)}}{T_j},1\right)\right)\right) \tag{7}$$

である。

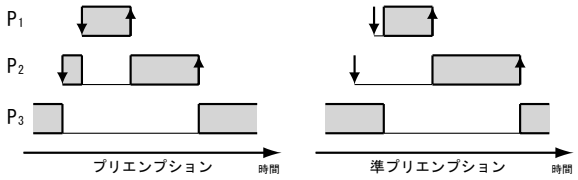

図 7: 切り替え回数の削減 (場合 1)

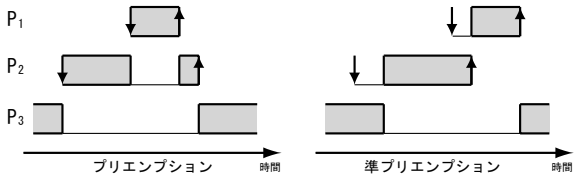

図 8: 切り替え回数の削減 (場合 2)

3.3 平均応答時間

従来のプリエンプション方式における平均応答時間 r_i、および準プリエンプション方式における平均応答時間 r_i' を求める。

従来のプリエンプションの場合、プロセス P_i の平均応答時間 r_i は、プロセスの実行時間 C_i に、プロセスの起床から完了するまでに割り込んだプロセスの実行時間を加えたものである。

まず、プロセス P_i の起床から完了するまでに割り込んだプロセスの実行時間を求める。プロセス P_i より優先度の高いプロセス $\mathrm{P}_j(j<i)$ のプロセッサ利用率は、

$$\frac{C_j}{T_j} \tag{8}$$

と表されるので、P_i より優先度の高いプロセスの総プロセッサ利用率は

$$\sum_{j=1}^{i-1}\frac{C_j}{T_j} \tag{9}$$

となる。ただし、

$$\sum_{j=1}^{i-1}\frac{C_j}{T_j}<1 \tag{10}$$

と仮定する。これは、P_i より優先度の高いプロセスについては処理負荷がプロセッサの処理能力を越えないことを意味する。この仮定は、応答時間が重要となるプロセスに関しては妥当なものである。

プロセス P_i の起床から完了するまでに割り込んだプロセスの実行時間は、期間 r_i における P_i より優先度の高いプロセスの総処理時間なので、式 (9) より、

$$r_i \sum_{j=1}^{i-1} \frac{C_j}{T_j} \tag{11}$$

となる。

したがって、プロセス P_i の平均応答時間 r_i は、

$$r_i = C_i + r_i \sum_{j=1}^{i-1} \frac{C_j}{T_j} \tag{12}$$

と表すことができる。これを整理すると、

$$r_i = \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \tag{13}$$

となる。ただし、式 (10) より、

$$1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j} \geq 0 \tag{14}$$

である。

準プリエンプションの場合も同様に考える。プロセス P_i の平均応答時間 r'_i は、P_i の実行時間 C_i に、P_i の起床から完了するまでに割り込んだプロセスの実行時間と、P_i の起床から次のプリエンプションポイントまでの平均遅延時間を加えたものである。

プリエンプションポイントを通過する平均間隔を I とすると、着目するプロセス P_i の起床から次のプリエンプションポイントまでの平均遅延時間は $\frac{I}{2}$ となる。また、プロセス P_i の終端処理の実行時間を $c_{(i,term)}$ とする。

着目するプロセス P_i より優先度の高いプロセス P_j が、P_i の応答時間に影響を与えるのは、P_j が、P_i の起床する直前のプリエンプションポイントから、P_i の終端処理を開始するまでの期間 t に起床した場合である。図 9 より、

$$t = I - \frac{I}{2} + r'_i - c_{(i,term)} \tag{15}$$

である。

したがって、プロセス P_i の平均応答時間 r'_i は、

$$r'_i = C_i + t \sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{I}{2} \tag{16}$$

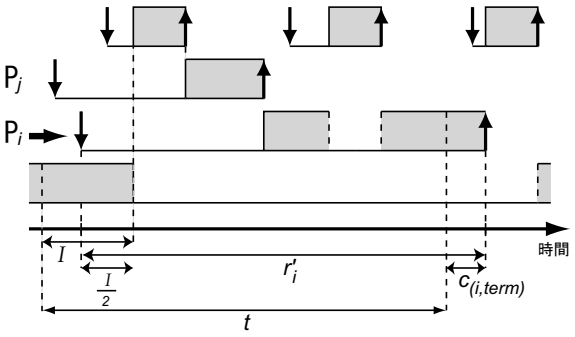

図 9: 準プリエンプションの場合の平均応答時間

と表すことができる。これを整理すると、

$$r'_i = \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} + \frac{(I - c_{(i,term)}) \sum_{j=1}^{i-1} \frac{C_j}{T_j}}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} + \frac{I}{2} \tag{17}$$

となる。

従来のプリエンプションと準プリエンプションの平均応答時間を比較する。式 (17)、式 (13) より差分を求めると、

$$\frac{(I - c_{(i,term)}) \sum_{j=1}^{i-1} \frac{C_j}{T_j}}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} + \frac{I}{2} \tag{18}$$

となる。

式 (18) において、$c_{(i,term)} = 0.2$、$\sum_{j=1}^{i-1} \frac{C_j}{T_j}$ を 0.2(低負荷), 0.5(中負荷),0.8(高負荷) とした場合の値を図 10 に示す。

図 10 より、プロセス P_i より優先度の高いプロセスの処理負荷が小さく、プロセッサ時間が十分に割り当てられる場合には、準プリエンプションによる応答時間の長大化の影響が小さくなることがわかる。

4 準プリエンプション機能の実装

我々の開発しているオペレーティングシステム CEFOS における、準プリエンプション機能の実装方式について述べる。制御移行の要求には要求情報表示 (DRD: Display Request and Data) 機能 [3] を利用する。プリエンプションポイントはスレッド切り替え時とし、制御移行要求の確認は、スレッドスケジューラにて行う。これらの詳細については後述する。

4.1 CEFOS

CEFOS では、複数の計算機を高速な通信路で結んで構築したシステムを想定している。

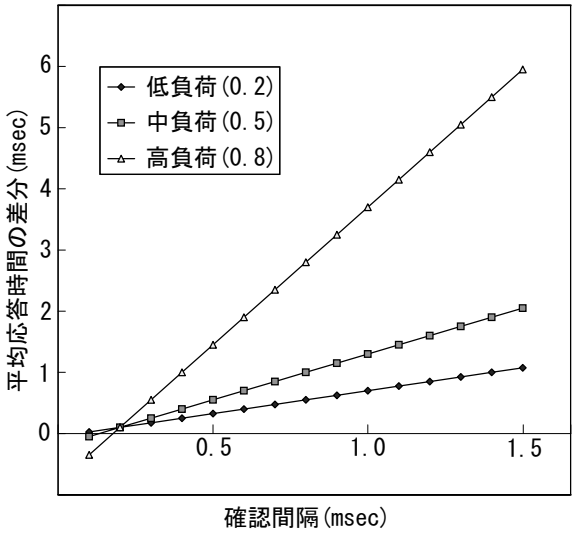

図 10: 平均応答時間と処理負荷

ソフトウェアは、ジョブとプロセスおよびスレッドから構成される。これらのを以下に説明する。

- システム内に数個のジョブが存在する。
- ジョブは、サービスを提供する主体であり、１個以上のプロセスから構成される。
- プロセスは、資源管理の単位であり、１個以上のスレッドから構成される。一つのプロセスを構成するスレッド群は、一つの計算機で実行される。
- スレッドは、プロセッサ割り当ての単位である。スレッドは、停止することなく連続して走行し、走行最長時間は１ミリ秒である。

CEFOS では、各ジョブが提供するサービスを偏りなく実現するために、プロセス間だけでなくスレッド間を含めた公平な処理環境を実現する。このため、CEFOS では次のような方針でプロセスをスケジューリングする。

(1) プロセス優先度の高いプロセスを優先

(2) 同一プロセス優先度のプロセスは、スレッド優先度が高い実行可能スレッドを持つプロセスを優先

(3) 同一プロセス優先度で、プロセス内の実行可能スレッドの優先度も等しい場合、現走行プロセスを優先

スレッドスケジュールの基本機構を図 11 に示し、以下に説明する。

(1) プロセススケジューラは、各計算機ごとに存在し、計算機内のプロセスを管理する。

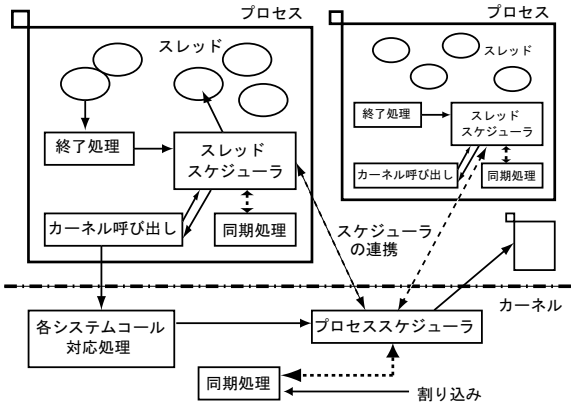

図 11: スレッドスケジュールの基本機構

(2) スレッドスケジューラは、各プロセスごとに存在し、プロセス内のスレッドを管理する。

(3) プロセススケジューラは、プロセスの実行優先度の逆転が生じると現走行プロセスのスレッドスケジューラに対して、制御移行を要求する。

(4) スレッドスケジューラは、制御移行要求を受け取るとカーネル呼び出しを行い、カーネルに制御を移行する。

スレッドスケジューラは各スレッドの実行終了後呼び出され、スレッドの走行最長時間は 1 ミリ秒なので、制御移行要求の確認は少なくとも 1 ミリ秒に 1 回は行われる。

4.2 DRD 機能

DRD 機能は、走行モードの変更やコンテキストの切り替えを伴わないため、処理オーバヘッドの少ない OS カーネル内外連携機能である。DRD 機能では、プロセスとカーネルの処理連携の手段として「表示 (Display)」を基本としている。

DRD 機能の処理概要を以下に示す。

(1) プロセスとカーネルは、共有するメモリ領域 (CA: Common Area) を保有する。

(2) 処理の要求 (Request) や、処理状態の内容 (Data) を CA に表示 (Display) する。

(3) プロセスとカーネルは、表示された情報を利用して処理の連携を行う。

(4) 必要に応じ、プロセスのカーネル呼び出し (システムコール) や、カーネルのプロセス呼び出し (上位呼び出し) を行う。

準プリエンプション機能では、カーネルは２つのデータ (cts, thr_pri) を CA に表示する。制御移行要求 ctr は、プロセス優先度の逆転がおこっていることを示すものである。スレッド優先度 thr_pri は、現走行プロセスと同じプロセス優先度で、ready 状態にあるプロセス内の実行可能スレッド優先度の最高値を示すものである。これらのデータを利用して、スレッドスケジューラは優先度逆転が生じているかを判断し、優先度逆転が生じている場合にはカーネルに制御を移行する。

4.3 スレッドスケジューラ

スレッドスケジューラの処理の流れを図 12 に示し、説明する。

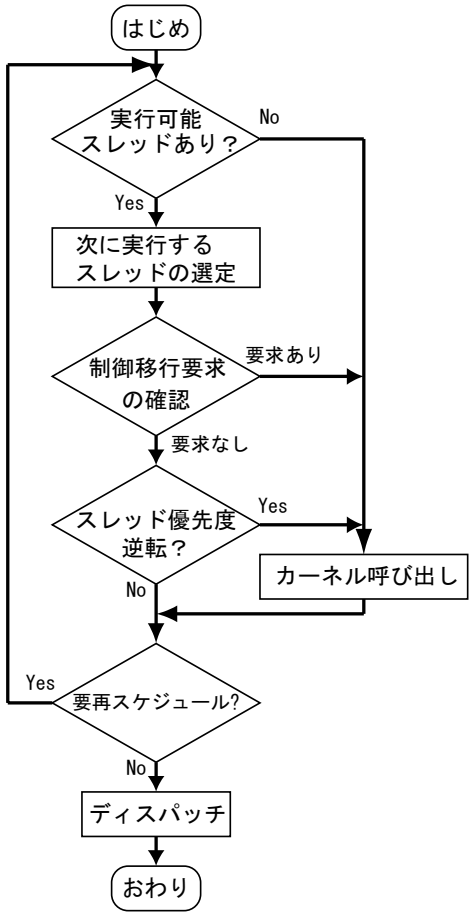

図 12: スレッドスケジューラ

最初に、プロセス内に実行可能なスレッドがあるかを確かめる。プロセス内に実行可能なスレッドがない場合、カーネル呼び出しを行う。この時、プロセス内に実行可能プロセスがないことをカーネルに通知する。プロセス内に実行可能スレッドがある場合、次に実行するスレッド (next) を選定する。

次に、カーネルが制御移行要求を表示していないか確認する。制御移行要求が表示されている場合、カーネル呼び出しを行う。この時、プロセス内の実行可能スレッドの最高優先度 (next のスレッド優先度) をカーネルに通知する。

次に、スレッド優先度逆転がおこっていないか確認する。これは、カーネルの表示するスレッド優先度とプロセス内実行可能スレッドの最高優先度 (next の優先度) の比較を行う。スレッド優先度の逆転が起きている場合、カーネル呼び出しを行う。この時、プロセス内の実行可能スレッドの最高優先度 (next のスレッド優先度) をカーネルに通知する。

5 おわりに

優先度逆転に伴うプロセス切り替えを遅延することによって、プロセス切り替えのオーバヘッドを削減し、また、プロセス切り替え回数を削減する準プリエンプション機能を提案した。

準プリエンプション機能とは、優先度逆転が生じても、直ちにはプロセス切り替えを行わず、現在走行しているプロセスの適当な切れ目を契機として、プロセス切り替えを行い、優先度逆転を解消する機能である。

本論文では、準プリエンプション機能を実現する際の問題点として、応答時間の長大化をあげ、その影響について検討を行った。

また、準プリエンプションの、我々の開発している CEFOS での実装方式について述べた。制御移行の要求には DRD 機能を利用し、プリエンプションポイントはスレッド切り替え時とし、制御移行要求の確認は、スレッドスケジューラにて行った。

現在、CEFOS に準プリエンプション機能の実装作業を行っている。残された課題として、実装した準プリエンプション機能の評価がある。

参考文献

[1] 日下部茂，富安洋史，村上和彰，谷口秀夫，雨宮真人，”並列分散オペレーティングシステム CEFOS(Communication-Execution Fusion OS)”，信学技報，Vol.99, No.251, pp.25-32 (1999).

[2] 谷口 秀夫, 日下部 茂, 棚林 拓也, 中山 大士, 雨宮 真人, “CEFOS オペレーティングシステムのスレッド管理機構,” 情処研報, 2000-OS-83, Vol.2000, No.21, pp.7-12 (2000).

[3] 谷口秀夫，”新しい OS カーネル内外連携機構:DRD 機構の提案”，信学論，D-I(掲載予定).