日曜研究室

技術的な観点から日常を綴ります

[xv6 #51] Chapter 4 – Scheduling – Real world

テキストの60〜61ページ

本文

xv6のスケジューラは、各プロセスを順番に実行するだけという簡単なスケジューリングのポリシーを実装している。
この方法はラウンドロビンと呼ばれる。
実際のOSではより洗練されたポリシーが採用されていて、例えば、プロセスが優先度を持つことが出来るようになっている。
このアイデアは、実行可能な高優先度なプロセスが、その他の実行可能な低優先度なスレッドからスケジューラによって優先されるという事である。
それらのポリシーは、しばしば矛盾したゴールを目指さなければならないので、急速に複雑化する可能性がある。
例えば、高いスループットと公平さの保証の両立など。
それに複雑なポリシーは、プライオリティインバージョン（優先度が逆に作用してしまうこと）やコンボイ（かなり大雑把に言うと順番待ちの列のこと）のような意図しない相互作用をもたらす可能性もある。
プライオリティインバージョンは、高優先度のプロセスが低優先度のプロセスが保持しているロックを待っているときに起こりうる。（高優先度のプロセスより、低優先度のものが優先されてしまう。またその他の無関係の中優先度のプロセスも優先される可能性がある。）
長いコンボイは、たくさんの高優先度のプロセスが低優先度のプロセスが保持しているロックを待っているときに形作られる可能性がある。
一度コンボイが形成されてしまうと、待っているたくさんのプロセスは処理を続行出来ず、長期にわたって残存することになる。
このような機構の追加による問題を避けることは、洗練されたスケジューラに必要なことである。

sleepとwakeupは、シンプルかつ効果的な同期化メソッドであるが、たくさんある方法のうちの一つでしかない。
それらの方法すべての最初の挑戦は、この章の最初の方で見た”起き損ないの問題”を避ける事である。

オリジナルのUnixカーネルのsleepは、単純に割り込みを無効化するだけだったが、当時はUnixはシングルCPU上で実行されていたので、それで十分だった。
xv6はマルチプロセッサ上で実行されるので、sleepのための直接的なロックを一つ追加した。
FreeBSDのmsleepはこれと同じアプローチを採用している。
Plan 9のsleepは、スリープし終わる直前に、スケジューリングのロックを保持した状態で実行されるコールバック関数を使っている。
その関数は、起き損ないを避けるために、sleepする条件の最後の瞬間におけるチェックを提供する。
Linuxカーネルのsleepは、ウェイトチャンネルの代わりに、直接的なプロセスのキューを使う。
そのキューは、内部ロックを持つ。（高機能なキューで、呼び出すだけでそのキューの操作のためのロックをいい感じに扱ってくれるインターフェイスを持ってるということかな？）

wakeupの中で、chanが合致するプロセスを捜すために、プロセスリストの全体を走査するのは、非効率的である。
より良いやり方としては、sleepとwakeupにおけるchanを、データ構造に置き換えて、そのデータ構造（チャンネル）上でスリープしてるプロセスのリストを、そのデータ構造自身に持たせる方法がある。
Plan 9のsleepとwakeupでは、そのような構造体を、ランデブーポイントまたはrendezと呼ぶ。
多くのスレッドライブラリは、そのような構造体を条件変数（condition variable）として使う。
そのような文脈では、sleepとwakeupは、waitとsignalと呼ばれる。
そのような機構のすべては、同じような考え方を共有している。
スリープするための条件は、スリープ中に自動的に働く何らかのロック的なもので保護される。

wakeupの実装は、個別のチャンネル上でスリープしているすべてのプロセスを起こす。
そして、同じチャンネル上でたくさんのプロセスが待つという場合もありうる。
OSはそれらすべてのプロセスをスケジュールし、、スリープする条件をチェックするために同時に実行されるだろう。
このように振舞うプロセスは、ときどきthundering herd（凄いことになってる群れ、と言ったところか）と呼ばれ、このような事態は避けたほうが良い。
多くの条件変数は、wakeupやsignalのために2つのプリミティブを持つ。
signalは、ひとつのプロセスを起こし、そして待っているプロセスすべてを起こすブロードキャストを行う。

セマフォは、他のよく知られた協調機構である。
セマフォは、インクリメント・デクリメント（もしくはアップ・ダウン）の2つの操作ができる整数値である。
セマフォは常にインクリメント可能だが、ゼロ未満になることはできない。
ゼロになってるセマフォをデクリメントする事は、他のプロセスがそのセマフォをインクリメントするまで、スリープすることになり、そしてそれら2つの操作は結果的に相殺される。
セマフォの整数値は、典型的には実際のカウントに一致するように使う。
例えば、パイプのバッファで読み込み可能となっているデータのバイト数や、あるプロセスが持って

いるZOMBIEな子プロセスの数など。
抽象概念の一部として、実際にwakeupした回数を直接数え上げるということは、”起き損ないの問題”を避ける。
そのような数え上げは、見せかけのwakeupとthundering herd問題を避ける。

感想

まとめると、
・スケジューラを高機能にするのは大変。
・主なOSにおけるsleepとwakeupの実装方法の概要。
・sleep/wakeupの仕組み”条件が整うのを待つ”ような他の用途にも流用できるし、他にも良いやり方がある。
ってところでしょうか。

各OSにおけるsleepとwakeupの実装方法の違いについてですが、Linuxのはどちらにしろロックを使ってるなら、おおまかなコードパスはxv6と似たようなものになるんじゃないかなと思います。
Plan 9の方法は、なんかもう…想像つきませんw

Thundering herd問題について調べると、有名なのは、キャッシュサーバのキャッシュが切れた場合に、多数のクライアントから一斉に再キャッシュ要求が集中してしまうこと、なんて例だったりしますが、本文における使われ方からして、たぶんキャッシュサーバとかに全く限定されない話みたいですね。
もしかしたら、コンピュータサイエンス界隈に限らず、なんらかの事象をきっかけにして、おおくのものが一斉に同じような事を行って混雑してしまう状態全般を指すのかもしれません。

なぜセマフォが、起き損ないの問題を解決出来るのかが、本文では分かりづらいので、ちょっと考えてみました。
そもそも起き損ないの問題とは、例えば、あるプロセスAがsleepする事を決めてから実際にsleepする直前までの間に、他のプロセスBがwakeupを呼ぶことによって、プロセスAが起きれなくなるような問題を指すわけです。
で、sleepをセマフォのデクリメントに、wakeupをセマフォのインクリメントに対応付けて考えると、sleepとwakeupの順番がちぐはぐになってしまった（sleep→wakeupであるべきなのがwakeup→sleepになってしまった）としても、セマフォの値はどちらにしろ最終的には元の値に戻る（ゼロ未満になることがないので、スリープしっぱなしにはならない。というか厳密にはこの場合はスリープしない）ので、起き損ないの問題が起きないという事かなと思います。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/3/26 月曜日 [http://peta.okechan.net/blog/archives/1571] |