OS 第 2 回課題レポート

1510151 柳 裕太

2017 年 7 月 20 日

1 課題 1: 和訳

1.1 デッドロックの条件

4 つの条件がデッドロック発生を引き起こす。[C+71]

- **相互排除:** 複数スレッドが必要なリソースの排他制御を要求するとき (例: 単一スレッドがロックの所有権を獲得する)
- **保持&待ち:** 複数スレッドが追加リソースを待つ間に、(例: これが、彼らがリソースの獲得を望んでいるように見える) 割り当てられたリソースを保持する (例: これが、彼らが既にリソースを獲得したもののように見える)
- **先取なし:** 複数リソース (例: ロック) を、所有されているスレッドから強制消去することができない
- **循環待機:** 次のスレッドが必要としている、1 つあるいはそれ以上のリソース (例: ロック) を持つスレッドらによる、循環連鎖が存在する

もしも 4 つのうちいずれかのの条件に出くわしていないのなら、デッドロックは発生することができない。それゆえ、我々はまずデッドロックを回避する技術調べる…これらの各戦術が、これらの条件の 1 つを発生から回避することを求めており、それゆえ、これが 1 つのデッドロック問題に対する対応方法となっている。

1.2 防止策

1.2.1 循環待機

概ね、多くの実践的な防止技術 (そして、確かにその 1 つはよく採用されている) は、あなたに循環待機を決して引き起こさないようなロックするコードを書かせるだろう。最もわかりやすくそれを遂行する方法は、たった 2 つのロックを取り入れたシステム (L1 と L2) で、あなたは常に L1 が L2 の前に獲得させることで、デッドロックを防止することができる。厳格な順番こそが、環状待機発生なしを確証させ、それこそがデッドロックなしとなる。

もちろんのこと、更に複雑なシステム内では、2 つ以上のロックが存在するだろうし、またゆえに全体のロックの順序も達成は難しくなるかもしれない (そしてあるいは不要なのかもしれないのはさておき)。それゆえ、**部分配列**はデッドロックを回避するための、ロック獲得構造を使いやすい方法になりえるのである。1 つの素晴らしい部分ロック順の現実の例は Linux[T+94] にお

ける、メモリマッピングコードにて閲覧可能であり、ソースコード上部のコメントには、10 の異なるロック獲得順のグループを明らかにしており、”i_mmap_mutex の前に i_mmap”のようなシンプルな 1 つや”mapping->tree_lock の前に swap_lock の前に private_lock の前に i_mmap_mutex” のようなより複雑な順序を含んでいる。

あなたが想像できるように、双方の全体と一部の獲得順はロックを行う戦術における慎重なデザインであり、また念入りな注意によって構築されなければならない。更に言えば、順番はたかが慣習であり、また杜撰なプログラマーは簡単にプロトコルをロックすることを無視し、そしてデッドロックを引き起こす可能性もありえる。最終的に、ロック順はコードベースと、そしていかに多くのルーチンが呼び出されるか…における深い理解必要とし、たった 1 つのミスが”D”(Deadlock の象徴) の世界に入る結果に終わる。

1.2.2 保持&待ち

デッドロックにおける保持&待ちの条件は、すべてのロックを 1 度獲得することで回避することができる、原始的だが。実践では、これは以下のコードで達成することが可能である。

```
1  pthread_mutex_lock(prevention);    // begin lock acquistion
2  pthread_mutex_lock(L1);
3  pthread_mutex_lock(L2);
4  ...
5  pthread_mutex_unlock(prevention); // end
```

最初の prevention ロックを獲得することで、このコードは中央のロック獲得や、そしてそれによるデッドロックがが再発することが引き起こしかねない時期尚早なスレッド切替ががないことを確約し、防ぐ。もちろんのこと、これはいつでもどのスレッドがロックを獲得することを要求し、それが最初に得るグローバル防止ロックとなる。例えば、もし別のスレッドがロック L1 と L2 を、それぞれ別の順で獲得しようとした時、問題はない、何故ならばこれは防止ロックを実行中に保持しているためである。

ここで留意すべきは、この解決策は多数の理由において問題があることだ。従来では、作品を我々とは反対にカプセル化した…ルーチンが呼ばれた時、このアプローチは我々に、まさにどのロックが保持されそして獲得するかを前もって知ることを要求する。この技術はまた、彼らが本当に必要とされたときに代わって、すべての早い段階 (一度) で獲得されなければならないロックの同時並行性を落としそうなのである。