gUSA: Simple and Efficient User Space Atomicity Emulation with Little Kernel Modification

g新部 裕
<gniibe@m17n.org>
独立行政法人 産業技術総合研究所 (AIST)

2002 年 9 月 19 日

概 要

本論文では, gUSA(“g” User Space Atomicity) と名付けた, ユーザ空間における排他制御の仕組みについて, 原理と実装を述べる。gUSA は, メモリロックの機構がない単一プロセッサシステムを対象とし, 排他制御をエミュレートする。

gUSA は, General (排他制御一般に利用可能), Generic (広範囲のプロセッサに対応可能), Gentle (カーネルの変更は最小限であり, かつ適用が容易) という特徴を持った方式である。簡潔で明解な仕組みであり, 実行時の性能も高い優れた方式である。

1 はじめに

SuperH, ARM, MIPS の一部など, 組み込み用途のプロセッサでは, ハードウェアによるメモリロックの機構が貧弱であったり, 存在しない実装が多く見られる。例えば, SuperH では TAS (test and set) 命令だけがサポートされる。MIPS R5900 ではメモリロックの機構が存在しない。ARM 7, ARM 9 では SWP (data swap) 命令だけがサポートされるが, 短縮系命令セットの Thumb では SWP 命令は存在しない。

これは, これらのプロセッサが単一プロセッサのシステム構成を前提としていること, および, メモリロックの機構の省略によりハードウェアの実装を簡略化できることなどによる。[1]

メモリロックの機構がないプロセッサでは, スレッドプログラミングの基本要素である排他制御はソフトウェアでエミュレートして実装することになる。しかし, これまでの方式は十分なものはなく, 特定の排他制御の機構だけを対象とする仕組みであったり, 特定のプロセッサに限定されたものであったり, カーネ

[1]SMP (Symmetric Multi-Processor: 対称マルチプロセッサ) をサポートするプロセッサでは, ll/sc (Load Locked / Store Conditional) 命令に代表される強力なメモリロックの機構がサポートされている。これを適切に使うと, 排他制御が簡単に効率よく実装できる。[1]

ルの大規模な変更を必要とするものであった。このため, これらの方式は限定的な利用に限られ, 広く採用されるに至っていない。

一方, 組み込み用途, 単一プロセッサのシステムであっても, プログラミングモデルとしてのスレッドは有用である。プログラミングモデルとしてスレッドを利用する言語やミドルウェアの利用が発展してきており, そのニーズは高まっている。実行時にスレッドを多用し, 同時に多くのスレッドが走るプログラムは現実的ではないが, 組み込みシステムにおいても, スレッドの利用は必要性が高い。

gUSA(“g” User Space Atomicity) はこういった要求を満足するために考案された仕組みである。メモリロックの機構がない単一プロセッサシステムを対象とし, ユーザ空間における排他制御を実現する。

gUSA は, 簡潔で明解な仕組みであり, 実行時の性能も高く, 以下の 3 つの特徴を持つ。

General : 排他制御のいろいろなルーチンの実装に対して利用できる, 一般性の高い仕組みである。

Generic : 特殊な機能を用いず広範囲のプロセッサに対応可能である。

Gentle : カーネルの変更は最小限であり, 適用が容易である。また, 既存の機能に与える影響が少ない。

これまで, カーネルにおける排他制御の実装は一般的なインタフェースと機能が検討されてきたが [2][3], gUSA はユーザ空間における排他制御の仕組みとしてユニークである。

gUSA により, 組み込みシステムにおいても, スレッドのプログラミングモデルを利用する展望が開けることが期待される。

以下では, gUSA について, その原理を述べ, GNU/Linux における実装例を紹介する。そして実装の評価を行い, 仕組みについて考察を行う。

2 gUSA の原理

ユーザ空間において, スレッド相互の排他制御を考える。

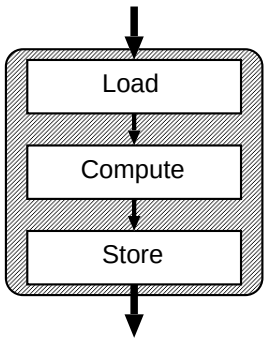

図 1: 排他領域 (critical region)

排他領域 (critical region) の命令の列が図 1 のように, Load でメモリからデータを読み出し, Compute で計算を行い, Store でメモリにデータを書き込むという 3 段階で構成されるものとする。

排他領域は, アトミックに動いて欲しい。つまり, この 3 段階は, 制御が分断されることなく一つの固まりとして処理が進んで欲しい。しかし, 3 段階を常にアトミックに処理するという条件は強すぎて実現が難しい。そこで, データの一貫性において同等である, より緩い条件を考え, 以下の条件とする。

Store で書き込まれるデータは, 分断されることなく 3 段階で処理されたデータである。

この緩い条件は実現可能である。

3 段階の命令列が途中で割り込まれて制御が他に移った場合, 割り込みから戻る時に戻り先を先頭の Load の命令列とすることにより, 常に Load, Compute, Store の一連の動作が行われることが保証できる。Load, Compute は複数回実行されることがあるし, 途中までしか実行されないこともあるが, Store されるデータは必ずアトミックな処理で行われる。

この動作を図示すると, 図 2 のようになる。図中, 点線は通常の制御の流れであり, 条件が満たされず排他制御が失敗する場合を示している。実線が, gUSA での制御の流れであり, 条件が満たされ, 排他制御が成功する場合を示している。

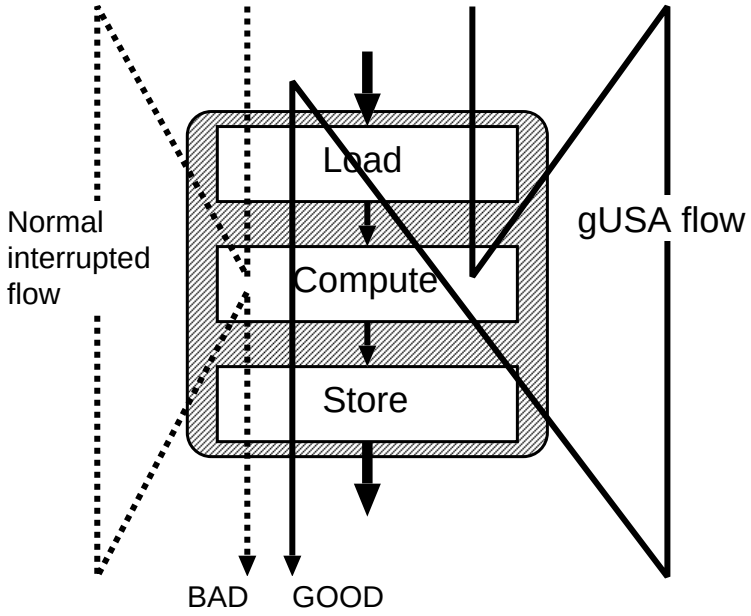

図 2: gUSA の原理

これは, つまり, 以下の簡潔で明快な動きである。

割り込まれたら最初からやり直し (rewind)。

この「最初からやり直し」の機能はカーネルで実現される。これを実現するためには, ユーザ空間において ABI (Application Binary Interface) を拡張し,

以下の 2 つが出来るようにすればよい。

- 排他領域の弁別
- 戻り先の同定

排他領域の弁別は簡単に実装できる。例えば,「スタックポインタがユーザ空間を指していない」というアブノーマルの状況をこれにあてる。戻り先の同定は, そのためのレジスタ (あるいはスタック上の配置) を定義するだけでよい。

カーネルでは制御が割り込まれるすべての場合に対してやり直す必要はない。Unix の場合では, 以下の 2 点を扱えばよい。[2]

- シグナルハンドラに制御が移る場合
- preemption で他のスレッドに制御が移る場合

よって, gUSA のためのカーネルの変更は最小限となり, Linux の場合, シグナルの処理のところ (handle_signal) と, 再スケジュールのところ (reschedule) の 2 点だけで良いことになる。プロセスの拡張もなく, 特別な仮想ドライバなども必要なく, このために用いるカーネルのリソースも増えることなく, 軽い実装となる。

3 gUSA の実装

GNU/Linux システムにおいて, SuperH (SH-3, SH-4) プロセッサと MIPS R5900 プロセッサに対して, gUSA の実装を行った。それぞれについて以下に述べる。

SuperH, MIPS R5900 ともにスタックポインタが (符号付き整数として) 負の値を取ることによって排他領域を示すものとした。Linux のシグナルの処理では, スタックが用いられるため, スタックポインタはシグナルの処理の際に復帰する必要がある。このため, 戻り先は 2 種類あることになる。

3.1 SuperH プロセッサ (SH-3, SH-4) での実装

SuperH プロセッサは, TAS (test and set) 命令を持つが, 一般的なメモリロックの機構は持たない。ユーザ空間での排他制御は, TAS 命令を元にして構成する方法が考えられるが, gUSA の方が効率がよいと考えられる。

実装を付録 A に示す。ここでは, カーネルの変更の全てと, アトミックな加算を示した。

ユーザ空間の ABI は排他領域に関し, 以下の拡張を行った。

- スタックポインタ r15 は (符号付き整数として) 負の値を取り, 絶対値が排他領域の大きさを示す。
- レジスタ r0 は排他領域の出口を指す。
- レジスタ r1 に元のスタックポインタ r15 の値が退避される。

[2] ハードウェアのドライバによる割り込みは対象としなくても良い。

「排他領域の弁別」についてはスタックポインタ r15 が符号無し整数として 0xc0000000 より大きい値を持つことによって行う。

「戻り先の同定」については, 戻り先は二種類あり, それぞれ以下のように求める。

シグナル処理からの戻り先 レジスタ r0 とスタックポインタ r15 を加えた値から 2 番地 (一命令分) 前である。

再スケジュールからの戻り先 レジスタ r0 とスタックポインタ r15 を加えた値である。

カーネルの変更は上記の設計にそってそのまま素直に実現されている。

ユーザ空間での利用も排他領域を ABI にそってレジスタを設定するだけの簡単な実現である。ここで, アトミックな加算の実装において, 排他制御の出口のラベルは 4 バイトアラインされている必要があるため, nop で調整を行っている。

3.2 MIPS 5900 プロセッサでの実装

MIPS R5900 プロセッサは排他制御の命令がサポートされない。ユーザ空間での排他制御は, カーネルを拡張する方法がいくつか考えられるが, gUSA が効率よい実装になると考えられる。

実装を付録 B に示す。ここでは, カーネルの変更の全てと, アトミックな test and set, および, アトミックな加算を示した。

ユーザ空間の ABI は排他領域に関し, 以下の拡張を行った。

- スタックポインタ sp は (符号付き整数として) 負の値を取り, 絶対値が排他領域の大きさを示す。
- レジスタ t0 は排他領域の出口を指す。
- レジスタ t1 に元のスタックポインタ sp の値が退避される。

「排他領域の弁別」についてはスタックポインタ sp (r29) が符号無し整数として 0xc0000000 より大きい値を持つことによって行う。

「戻り先の同定」については, 戻り先は二種類あり, それぞれ以下のように求める。

シグナル処理からの戻り先 レジスタ t0 とスタックポインタ sp を加えた値から 4 番地 (一命令分) 前である。

再スケジュールからの戻り先 レジスタ t0 とスタックポインタ sp を加えた値である。

カーネルの変更は上記の設計にそってそのまま素直に実現されている。

ユーザ空間での利用も排他領域を ABI にそってレジスタを設定するだけの簡単な実現である。

4 gUSA の実装の評価

gUSA の実装を評価し, その仕組みの妥当性を検証するため, カーネルの変更の影響と, ユーザ空間での利用の性能を調べた。

4.1 カーネルの変更の影響

カーネルの変更は付録 A, 付録 B で示したものですべてであり, 字面上はとても少ない。また, 変更は, handle_signal と reschedule という比較的重い処理の場所に若干の命令列を加えているだけなので, 理論上はカーネルの性能劣化はほとんどないと考えられる。

SuperH (SH-3, SH-4) の Debian のシステムにおいて, gUSA 適用後のカーネル (gusa) gUSA 適用前のカーネル (plain) とで, lmbench-2.0-patch2-2 のベンチマークにおけるプロセスとコンテクストスイッチの指標を求めた。gUSA 適用後のカーネル (gusa) は 9 回, gUSA 適用前のカーネル (plain) は 3 回走らせた。

この結果を付録 C に示す。この計測では, カーネルの性能劣化はほぼ見られず, gUSA の優秀さを示していると言える。

4.2 ユーザ空間での利用の性能

ユーザ空間での利用は, 簡潔で明快な実装となるため, 理論上は高い性能をあげると考えられる。

計測のために `eaa.c` [4] というプログラムを用意し, SuperH (SH-4) の Debian のシステムと MIPS 5900 の PS2 Linux のシステムにおいて, それぞれ実行し, プログラムの経過時間を計測した。

計測では, アトミックな加算の実行の対象となるオブジェクトの数 (O) と一つのオブジェクトを共有するスレッドの数 (T) がパラメータとなる。[3]

`eaa.c` では, アトミックな加算 (exchange_and_add) の実現を以下の異なるタイプから選択することができる。

1. gUSA による実装
2. test and set による実装 (単一ロック)
3. test and set による実装 (分散ロック)
4. システムコールによる実装
5. test and set (sony) による実装 (単一ロック)
6. test and set (sony) による実装 (分散ロック)

タイプ 0 は gUSA によるアトミックな加算を直接実現する。タイプ 1, 2, 4, 5 は, test and set を排他制御の要素とし, アトミックな加算を構成する。タイプ 1 は一つのロックをスレッドで共有し, タイプ 2 は O 個の別々のロックを持つ。タイプ 3 は, システムコールによる実現である。

ここで, SuperH ではタイプ 1 から 3 だけが意味を持つ (test and set は TAS 命令を用いる)。MIPS 5900 では, タイプ 3 はサポートされず, test and set は gUSA によるもの (タイプ 1 と 2) と Sony の特殊な実装 [5] (タイプ 4 と 5) の 2 種類がある。

計測した結果を SuperH の場合を表 1(a) に, MIPS 5900 の場合を表 1 (b) にそれぞれ示す。

全ての場合において gUSA による実装が高い性能を出している。スレッドの競合が起きてくると分散ロックの方が単一ロックよりも高い性能を出すよう

[3] 全体でのスレッドの数は $O \times T$ である。

表 1: ユーザ空間での性能計測 (eaa.c) (単位: 秒)

(a) SH-4

＼＿＿＿ (O,T) タイプ ＼	(1,1)	(2,2)	(4,4)	(8,8)
0 (gUSA)	0.986	4.71	18.9	76.5
1 (TAS 1)	2.14	17.6	191	2710
2 (TAS O)	2.19	12.2	76.7	555
3 (SYSCALL)	4.68	19.7	95.5	347

(b) MIPS 5900

＼＿＿＿ (O,T) タイプ ＼	(1,1)	(2,2)	(4,4)	(8,8)
0 (gUSA)	0.869	3.47	13.9	55.4
1 (TAS gUSA 1)	1.12	6.53	69.1	688
2 (TAS gUSA O)	1.13	4.73	26.4	151
4 (TAS Sony 1)	1.37	9.81	84.6	1060
5 (TAS Sony O)	1.41	6.33	36.9	203

になる。システムコールによる実装も場合によっては有用であると言える。test and set の実装を比較すると Sony の特殊な実装よりも gUSA による実装が高い性能を出している。

5 考察

2 節の原理の説明において, gUSA が General な仕組みであり排他制御のいろいろなルーチンの実装に対して利用できる, 一般性の高い仕組みであることが示された。また, プロセッサに大きく依存する点が少ないことから, 多くのプロセッサに対応可能であることが期待され, 実際, 3 節で複数のプロセッサに問題なく実装されることが示され, Generic な仕組みであることがわかった。4 節では, カーネルの既存の機能に与える影響が少ないことが計測され, Gentle な仕組みであると確かめられ, 実行時の性能も高いことが確かめられた。

本節では, さらに gUSA について詳しくその性質を考察する。

5.1 他のプロセッサに実装する際の注意点

Linux においてカーネルの変更箇所は, どのプロセッサも共通と考えられる。ABI の拡張において, スタックポインタを排他領域の表現に利用することもほぼ共通に使える手法であろう。

排他領域の出口を示すレジスタとして, アドレスを載せやすいレジスタを選

択すること, および命令長に留意すれば, 実現は容易であろう。

5.2 ユーザ空間における注意点

gUSA はユーザ空間の ABI を拡張する。SuperH, MIPS 5900 での実装では, スタックポインタをアブノーマルの値を持たせることで, 排他領域を表現することとしたが, この実装では, スタックポインタを一般のレジスタとして一時流用し, 値を退避するなどの利用はできない。流用した場合にカーネルのスケジュールあるいはシグナル処理に入ると, プログラムの制御は, 予期しないとんでもないところに移ってしまう。

このようなスタックポインタの利用は, コンパイラを通常に使っている限りありえず, アセンブラでユーザが明示的に実装する際に限定された問題であると言える。このため, 危惧する必要はない。スタックポインタを一般のレジスタとして利用する意味は, ほとんどないため, スタックポインタの値による排他領域の表現は妥当と考えられる。

排他領域の中では, スタックポインタを用いたアクセスはできないため, C 言語で自動変数にアクセスすることはできない。実際には, 排他制御はアセンブラで記述されることが普通であるため, これは問題とならないだろう。

5.3 セキュリティ

gUSA の実装における ABI 拡張とカーネルの変更が, システムのセキュリティレベルを低下させてはいけない。

スタックポインタの値を偽造することは可能であるが, これによってカーネルや他のプロセスに影響をあたえることはできず, 自分自身のスレッドだけに影響は限定される。このため, gUSA の実装においてシステムのセキュリティレベルを低下はないと考えられる。

5.4 スレッドライブラリ

gUSA はカーネルが管理するスレッドを利用するスレッドライブラリ (LinuxThreads など), ユーザ空間で実現されるスレッドライブラリ (Pth など) の両方に対して利用することができる。

gUSA の提供する排他制御の仕組みは, カーネルのスケジューラとシグナルハンドラの両方に対して有効なので, カーネルによりスレッドの preemption が起こる場合でも, シグナルによってユーザ空間のスレッドの制御が移る場合でも, 問題なくやり直し (rewind) が起こる。preemption のないスレッドライブラリであれば, そもそも排他制御の問題はないのでこれも問題ない。

5.5 カーネルの性能劣化

Linux においては reschedule はプロセスが制御を他のプロセスに移す時に呼ばれる場所である。handle_signal は, シグナルのフレームを作って制御をシグナルハンドラに移す場所である。このどちらも比較的重い処理である。

重い処理に対して最小限の追加を行っているため, カーネルの性能劣化がほとんど無い実装となっていると考えられる。SH-3 は SH-4, MIPS 5900 と比較するとレジスタの数が少ないため, コンテクストスイッチとシグナルの処理の重さが若干軽いと考えられる。SH-4, MIPS 5900 と比較して, 計測結果をさらに吟味すると SH-3 は, ほんの少しであるが性能劣化が見える。

5.6 Live lock の可能性

gUSA は一般的な排他制御の機構を提供するが, 巨大な排他領域は live lock となり, 排他領域から制御が出て来れなくなってしまう可能性がある。カーネルの preemption が必ず起こってしまうくらい複雑な排他領域は構成することができない。

そのような複雑な処理を排他領域とすることは, gUSA でなくとも問題であり, 避けなくてはならない。このためこの問題は実用上問題とならないと考えられる。

5.7 アトミックな Store

gUSA は Store がアトミックに実現されることに依存している。この条件が成り立たない場合, Load で一部 Store された不整合のデータを読み出して処理することがありうるため, gUSA は機能しない。

現実のプロセッサでは, アトミックな Store が保証される (あるいは保証されるバイト長がある) ため, 問題とならない。

5.8 Linux 以外のカーネル

gUSA は一般的な仕組みであり Linux 以外のカーネルにも適用可能であると考えられる。Unix 一般に, 3 節で示した実装が適用できるだろう。

5.9 排反する Linux の機能

gUSA は PREEMPTIVE カーネルの機能と共存できない。gUSA の実装は, ユーザ空間のスレッドが reschedule の場所でだけ再スケジュールが行われることに依存している。PREEMPTIVE カーネルの実装では, いろいろな所に preemption のポイントを設けカーネルが再スケジュールされるため, この前提が成り立たなくなり, 正しい動作が保証できなくなる。

6 おわりに

gUSA の実装は SuperH のカーネルにおいて, 2.4 シリーズで既に採用され, メインストリームへのマージを待つばかりである。2.5 シリーズへの適用と GNU C library への適用が次の課題である。

これまで, 組み込み用途ではスレッドの利用は一般的ではなかったが, gUSA により, その道が開けたのではないかと考えられる。gUSA を用いることで, スレッドライブラリの実装は効率の良いものとすることができ, スレッドのプログラミングモデルを採用することにつながる。これは, クロス開発においてシームレスな環境を広げることなり, 組み込み GNU/Linux システムの生産性をあげることが期待される。

最後に, gUSA の研究開発に協力いただいた各人に謝辞を述べたい。

gUSA の着想と実装に関して応援いただいた小島 一元さん, 田中 哲さんに感謝する。彼ら無くして, gUSA は生まれなかった。

実装に関して Greg Banks さん, Ulrich Drepper さん, 戸村 哲さんに感謝する。彼らの経験に基づく批評により, 議論を深め, より良い仕組みとすることができた。

SuperH, MIPS 5900, ARM の環境と技術情報に関して協力いただいた八重樫 剛史さん, 後藤 正徳さん, 野首 貴嗣さんに感謝する。彼らの協力によって, gUSA がさまざまなアーキテクチャに有効であることが示せた。

Happy Hacking.

参考文献

[1] John L. Hennessy, David A. Patterson, “Computer Architecture – A Quantitative Approach (2nd ed.)”, Morgan Kaufmann Publishers, 1996.

[2] Curt Schimmel, “UNIX Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers”, Addison-Wesley, 1994.

[3] Uresh Vahalia, “UNIX Internals :The New Frontiers”, Prentice-Hall, 1996.

[4] g新部 裕, “Exchange and Add”, `ftp://ftp.m17n.org/pub/super-h/eaa.c`, 2002 年 8 月.

[5] 町田 浩之, 篠原 孝夫, “PS2 Linux での安全なユーザレベル排他制御の実現”, Linux Conference 2001 (LC2001) 論文集, 日本リヌックス協会, 2001 年 9 月.

[6] 八重樫 剛史, 小島 一元ほか, “DODES Project”, `http://www.dodes.org/` 2002 年 3 月.

[7] g新部 裕ほか, “GNU/Linux on SuperH Project”, Linux Conference 2001 (LC2001) 論文集, 日本リヌックス協会, 2001 年 9 月.

[8] g新部 裕ほか, “GNU/Linux on SuperH Project”, Linux Conference 2000 (LC2000) 論文集, 日本リヌックス協会, 2000 年 11 月.

[9] “技術者のための UNIX 系 OS 入門”, インターフェース増刊 TECHI Vol. 5, CQ 出版, 2000 年 6 月.

[10] Andrew S. Tanenbaum, “Operating Systems: Design and Implementation”, Prentice-Hall, 1987.

A Linux / SuperH での実装

```
2002-08-15  NIIBE Yutaka  <gniibe@m17n.org>

        gUSA ("g" User Space Atomicity) support.
        * arch/sh/kernel/signal.c (handle_signal): Added gUSA handling.
        * arch/sh/kernel/entry.S (reschedule): Added gUSA handling.

--- linux-2.4.18.superh.no-gusa/arch/sh/kernel/entry.S        Wed Aug 14 08:54:12 2002
+++ linux-2.4.18.superh.gusa/arch/sh/kernel/entry.S        Thu Aug 15 23:14:47 2002
@@ -94,6 +94,7 @@ OFF_R5         =  20             /* New ABI: ar
 OFF_R6         =  24             /* New ABI: arg2 */
 OFF_R7         =  28             /* New ABI: arg3 */
 OFF_SP           =  (15*4)
+OFF_PC           =  (16*4)
 OFF_SR           =  (16*4+8)
 OFF_TRA    =  (16*4+6*4)

@@ -455,12 +456,28 @@ __INV_IMASK:

         .align        2
 reschedule:
+        ! gUSA handling
+        mov.l        @(OFF_SP,r15), r0        ! get user space stack pointer
+        mov        r0, r1
+        shll        r0
+        bf/s        1f
+         shll        r0
+        bf/s        1f
+         mov        #OFF_PC, r0
+        !                                SP >= 0xc0000000: gUSA mark
+        mov.l        @(r0,r15), r2                ! get user space PC (program counter)
+        mov.l        @(OFF_R0,r15), r3        ! end point
+        cmp/hs        r3, r2                        ! r2 >= r3?
+        bt        1f
+        add        r3, r1                           ! rewind point #2
+        mov.l        r1, @(r0,r15)                ! reset PC to rewind point #2
+        !
+1:        mov.l        2f, r1
         mova        SYMBOL_NAME(ret_from_syscall), r0
-        mov.l        1f, r1
         jmp        @r1
          lds        r0, pr
         .align        2
-1:        .long        SYMBOL_NAME(schedule)
+2:        .long        SYMBOL_NAME(schedule)

 ret_from_irq:
 ret_from_exception:
```

```
--- linux-2.4.18.superh.no-gusa/arch/sh/kernel/signal.c        Wed Aug 14 08:54:12 2002
+++ linux-2.4.18.superh.gusa/arch/sh/kernel/signal.c        Wed Aug 14 17:50:56 2002
@@ -533,6 +533,17 @@ handle_signal(unsigned long sig, struct
                        case -ERESTARTNOINTR:
                                regs->pc -= 2;
                }
+       } else {
+               /* gUSA handling */
+               if (regs->regs[15] >= 0xc0000000) {
+                       int offset = (int)regs->regs[15];
+
+                       /* Reset stack pointer: clear critical region mark */
+                       regs->regs[15] = regs->regs[1];
+                       if (regs->pc < regs->regs[0])
+                               /* Go to rewind point #1 */
+                               regs->pc = regs->regs[0] + offset - 2;
+               }
        }

        /* Set up the stack frame */
```

```
static int
gUSA_exchange_and_add (volatile int *mem, int val)
{
  unsigned long dummy;
  int result;

  __asm__ (".align 2\n\t"
       "mova    1f,r0\n\t"
       "nop\n\t"
       "mov     r15,r1\n\t"
       "mov     #-8,r15         ! critical region start: rewind point #1\n"
   "0:  mov.l   @%2,%0          ! rewind point #2\n\t"
       "mov     %3,%1\n\t"
       "add     %0,%1\n\t"
       "mov.l   %1,@%2\n"
   "1:  mov     r1,r15          ! critical region end"
           : "=&r" (result), "=&r" (dummy)
           : "r" (mem), "r" (val)
           : "memory", "r0", "r1");

  return result;
}
```

B Linux / MIPS R5900 での実装

```
2002-08-18  NIIBE Yutaka  <gniibe@m17n.org>

--- linux-2.2.1_ps2/arch/mips/kernel/entry.S.orig        Thu May 11 16:21:35 2000
+++ linux-2.2.1_ps2/arch/mips/kernel/entry.S        Sun Aug 18 22:13:19 2002
@@ -49,7 +49,21 @@ EXPORT(handle_bottom_half)
                 b        9f
                  nop

-reschedule:         jal         schedule
+reschedule:
+        /* gUSA handling begin */
+                 L_GREG         t0, PT_R29(sp)                # User stack
+                 lui         t1, 0xc000
+                 sltu         t2, t0, t1                # gUSA mark?
+                 bnez         t2, 1f
+                 lw         t1, PT_EPC(sp)                # User PC
+                 L_GREG         t2, PT_R8(sp)                # User T0: End point
+                 sltu         t3, t1, t2                # In the critical region?
+                 beqz         t3, 1f
+                 /* Rewind */
+                 addu         t1, t0, t2                # User PC <- User SP + Endpoint
+                 sw         t1, PT_EPC(sp)
+        /* gUSA handling end */
+1:                  jal         schedule
                  nop

 EXPORT(ret_from_sys_call)
--- linux-2.2.1_ps2/arch/mips/kernel/signal.c.orig        Wed Mar 21 02:24:48 2001
+++ linux-2.2.1_ps2/arch/mips/kernel/signal.c        Sun Aug 18 19:59:36 2002
@@ -586,6 +586,16 @@ segv_and_exit:
 static inline void handle_signal(unsigned long sig, struct k_sigaction *ka,
         siginfo_t *info, sigset_t *oldset, struct pt_regs * regs)
 {
+        if (regs->regs[29] >= 0xc0000000) { /* gUSA Handling */
+                int offset = (int)regs->regs[29];
+
+                /* Reset stack pointer: clear critical region mark */
+                regs->regs[29] = regs->regs[9];
+                if (regs->cp0_epc < regs->regs[8])
+                        /* Go to rewind point #1 */
+                        regs->cp0_epc = regs->regs[8] + offset - 4;
+        }
+
         if (ka->sa.sa_flags & SA_SIGINFO)
                 setup_frame(ka, regs, sig, oldset, info);
         else
```

```
        .text
        .align  3
        .globl  gUSA_test_and_set
        .ent    gUSA_test_and_set
        .set    noreorder
gUSA_test_and_set:
        li      $3, -1
        la      $8, 1f
        move    $9, $sp
        li      $sp, -8         # critical region start: rewind point #1
   0:   lw      $2, ($4)        # rewind point #2
        sw      $3, ($4)
   1:   move    $sp, $9         # critical region end
        j       $31
        nop
        .end    gUSA_test_and_set


static int
gUSA_exchange_and_add (volatile int *mem, int val)
{
  unsigned long dummy;
  int result;

  __asm__ (".align 2\n\t"
       "la      $8, 1f\n\t"
       "move    $9, $sp\n\t"
       "li      $sp, -16        # critical region start: rewind point #1\n"
   "0:  lw      %0, (%2)        # rewind point #2\n\t"
       "move    %1, %3\n\t"
       "addu    %1, %0, %1\n\t"
       "sw      %1, (%2)\n"
   "1:  move    $sp, $9         # critical region end"
           : "=&r" (result), "=&r" (dummy)
           : "r" (mem), "r" (val)
           : "memory", "$8", "$9");

  return result;
}
```

C lmbench の結果

```
SolutionEngine SH7709A (with 64MB memory)
sh3 133MHz 64MB: kernel 2.4.18.superh + gusa / plain,  NFS root
                    L M B E N C H  2 . 0   S U M M A R Y
      ------------------------------------------------------------------------
      Processor, Processes - average times in microseconds - smaller is better
      ------------------------------------------------------------------------
       null           null                          open/
       call  Error    I/O   Error    stat  Error   close  Error
      ------ ------  ------ ------  ------ ------  ------ ------
gusa   2.147  0.001   4.381  0.002  47.862  0.351  106.654  1.438
plain  2.145  0.002   4.382  0.002  47.085  0.183  108.220  1.120
      ....................................................................
                    signal         signal
      select Error  instll Error   catch  Error
      ------ ------  ------ ------  ------ ------
gusa   174.7   0.11  11.911  0.004  17.772  0.208
plain  174.4   0.30  11.909  0.001  16.886  0.005
      ....................................................................
       fork             exec             shell
       proc    Error    proc    Error    proc    Error
      ------- -------  ------- -------  ------- -------
gusa   4497.9    3.35  19212.6    9.92  71958.6   39.06
plain  4528.3    3.61  19753.3  493.85  73953.3 1946.68


      ------------------------------------------------------------------------
      Context switching - times in microseconds - smaller is better
      ------------------------------------------------------------------------
      2p/0K          2p/16K         2p/64K
             Error          Error          Error
      ------ ------  ------ ------  ------ ------
gusa    8.29  0.592  391.41  1.279  1217.76  0.860
plain   7.67  0.357  393.97  0.301  1217.80  1.406
      ....................................................................
      8p/0K          8p/16K         8p/64K
             Error          Error          Error
      ------ ------  ------ ------  ------ ------
gusa   21.22  1.631  407.25  1.631  1285.42  1.308
plain  19.99  2.860  402.61  2.860  1287.35  1.475
      ....................................................................
      16p/0K        16p/16K        16p/64K
             Error          Error          Error
      ------ ------  ------ ------  ------ ------
gusa   27.10  1.253  418.95  0.379  1297.86  0.347
plain  25.86  0.928  418.31  0.623  1297.31  0.765

      ========================================================================
```

```
SolutionEngine SH7750S
sh4 200MHz 64MB: kernel 2.4.18.superh + gusa / plain, NFS root
                    L M B E N C H  2 . 0   S U M M A R Y
      ------------------------------------------------------------------------
      Processor, Processes - average times in microseconds - smaller is better
      ------------------------------------------------------------------------
       null           null                         open/
       call  Error    I/O   Error    stat  Error   close  Error
      ------ ------  ------ ------  ------ ------  ------ ------
gusa   1.143  0.000   2.709  0.044  40.762  0.453  69.128  1.117
plain  1.143  0.000   2.772  0.121  37.264  0.324  67.690  0.975
      ..........................................................................
                    signal         signal
      select Error  instll Error   catch  Error
      ------ ------  ------ ------  ------ ------
gusa    99.1   2.05   9.390  0.041  20.944  0.116
plain  105.4   9.43   9.604  0.061  20.542  0.244
      ..........................................................................
       fork            exec            shell
       proc    Error   proc    Error   proc    Error
      ------- -------  ------- -------  ------- -------
gusa   1981.6   13.59   9825.2   18.22  41592.9   57.59
plain  1945.4   26.72  10464.7  358.93  44617.7 1490.07


      ------------------------------------------------------------------------
      Context switching - times in microseconds - smaller is better
      ------------------------------------------------------------------------
      2p/0K          2p/16K         2p/64K
             Error          Error          Error
      ------ ------  ------ ------  ------ ------
gusa    7.72  0.708   77.93  0.640  239.31  0.553
plain   9.28  1.760   78.30  0.867  238.32  1.164
      ..........................................................................
      8p/0K          8p/16K         8p/64K
             Error          Error          Error
      ------ ------  ------ ------  ------ ------
gusa   17.98  0.545  129.22  0.545  390.09  0.191
plain  18.58  0.286  130.40  0.286  392.51  1.499
      ..........................................................................
      16p/0K        16p/16K        16p/64K
             Error          Error          Error
      ------ ------  ------ ------  ------ ------
gusa   38.09  0.348  144.11  0.129  394.79  0.263
plain  39.49  0.228  145.63  0.617  398.11  3.186

      ========================================================================
```