組込みシステムにおけるLTTngの性能検討

原　聡美†　内野　聡†　本間　亨†
†株式会社　東芝　デジタルメディアネットワーク社

1　はじめに

情報家電機器などにおける組込みシステムのアプリケーションは年々複雑化し，全体像を把握することが難しくなっており，性能改善やデバッグに用いる解析ツールの重要性が増している．開発者はトレーサを用いることでシステムの状態を把握でき，ネックとなっている処理の解析を行いやすくなる．一方で，トレーサを組込むことによるシステム性能への影響も無視できない．

Linux上で動作するトレーサとしてLTTng（Linux Trace Toolkit Next Generation）[1]，ftrace event tracer[2]，SystemTap[3]などさまざまなものが存在しているが，本稿における評価環境に適したLTTngを用いて評価検討を行う．

本稿では，組込アプリケーション開発への適用を前提として，さまざまな条件下におけるLTTngの負荷を評価し，実システムへの適用における問題点や改善策について検討した結果を報告する．

2　LTTng

LTTngにはLTTV(LTTng Viewer)というトレースを解析するためのGUIが用意されていること，アプリケーション開発において重要となるユーザ空間に任意のイベントを作成することができるという利点がある．また，サポートしているCPUの種類が多く，Linuxの多くのバージョンに対応可能である．

LTTngはカーネルのソース内に直接情報を取得するコード（以下，トレースポイントと呼ぶ）が埋め込まれているトレーサのひとつである．この方法では例外などを利用するトレーサに比べると負荷が小さくなる．その反面，実際にトレースを取得していない状態であってもカーネル上では仕組みが有効であるため，なんらかの負荷が発生してしまう．

3　システム負荷評価

表1に示す環境を用いて，LTTngの負荷の測定をUnix系OSの基本機能の性能を評価するベンチマークであるLMbench(Version 3.0 alpha 1) [4]を用いて行う．また，今回の評価はNFSrootを用いて行っている．

表1：評価環境

	仕様
CPU	MIPS32®24KEf™ 533MHz, I-cache 32KB, D-cache 32KB
RAM	256MB（DDR2-800D）
Linux	2.6.20.19（LTTng-0.9.0）

3.1 トレースポイントの削減

本章では負荷を軽減するためのひとつの手段として個別にトレースポイントを選択し，不要と判断したトレースポイントを削除して実験を行う．有効とするトレースポイントは以下のものとした．

- LTTVで各プロセスの状態を判断するために使用しているもの
  例：kernel_irq_entry/exit,
  　　kernel_process_fork　　など
- アプリケーション開発に有益と思われるもの
  例：fs_write/readなどのファイルシステム
  　　操作などに関するトレースポイント

これにより例えば，timer系のトレースポイントのように頻繁に通過するが，アプリケーションの開発においてはそれほど意識する必要のないトレースポイントなどの影響を抑えることができるのではないかと考える．上記により，MIPSで埋め込まれる90のトレースポイント中47のトレースポイントを削除できる．

3.2 カーネルへのコード組込による負荷評価

LTTng組込みによる負荷について，表2のような条件で測定を行う．

表2：　コード組込による負荷測定条件

	CONFIG_MARKER	CONFIG_LTT	有効トレースポイント数
(1)	無効	無効	
(2)	有効	無効	0
(3)	有効	有効	0
(4)	有効	有効	43
(5)	有効	有効	90

LMbenchのテストのうち“Context switching” 及び“Processor,Processes”のテスト結果を，(1)の平均処理時間を1とした倍率として表3，表4に示す．また，参考のために(1)の場合は実測値を括弧内に表記している．

表3の“Context switching”テストでは2プロセス/バッファサイズ16KBの場合にやや性能低下が大き

Performance evaluation of LTTng in the embedded system.
Satomi Hara†, Satoshi Uchino†, and Toru Homma†
†Toshiba Corporation Digital Media Network Company

いが，(5)の状態であっても (1)の状態に対してほとんどのテストで1.1倍程度，その他にも“Memory Latency”，“Local Communication bandwidths”の各テストでは，(2)から(5)の全てでほぼ(1)と同程度の性能が得られている．一方で，表 4の“Processor, Processes”テストで(1)と(5)を比較すると 1.2～2.9倍程度，また“File & VMSystem”テストでも 1.1～3倍程度の時間がかかっている．本稿では“Processor, Processes”のテスト結果を用いて比較を行うこととする．

表 3：“Context Switching”結果(実測値の単位:usec)

	2p/16K	2p/64K	8p/16K	8p/64K	16p/16K	16p/64K
(1)	(45.6)	(392.0)	(108.9)	(397.2)	(107.1)	(396.7)
(2)	0.947	0.991	0.972	0.989	0.987	0.990
(3)	1.356	1.055	1.142	1.061	1.151	1.062
(4)	1.338	1.060	1.129	1.068	1.145	1.061
(5)	1.364	1.050	1.099	1.052	1.114	1.052

表 4： “Processor,Processes”の結果(実測値の単位:usec)

	null call	null I/O	stat	open/clos	slct TCP
(1)	(0.35)	(1.05)	(6.63)	(9.79)	(33.8)
(2)	1.592	1.206	1.020	1.075	1.071
(3)	2.256	2.014	2.305	2.874	1.303
(4)	2.426	2.092	2.758	3.831	1.235
(5)	2.457	2.508	2.883	3.185	1.236
	sig inst	sig hndl	fork proc	exec proc	sh proc
(1)	(1.22)	(6.76)	(1059)	(6258)	(17K)
(2)	1.193	0.888	1.030	1.023	1.039
(3)	1.901	1.845	1.386	1.286	1.377
(4)	1.936	1.923	1.458	1.397	1.437
(5)	2.041	2.211	1.499	1.352	1.442

表 4から，(5)と (3)を比較するとLTTngの負荷はトレースポイント以外の影響のほうが大きい．項目別にみると，null call以外ではCONFIG_LTTによる影響が大きくみられ，open/closでは(2)と(3)の比較よりCONFIG_LTTによる影響は最大約 2.7 倍となっている．なお，null callでは (2)より，CONFIG_MARKERによる影響が約 1.6 倍程度となっている． CONFIG_MARKERを有効にすることで，syscall_trace_entryへの分岐が有効となるため，システムコールの負荷が大きくなると考えられるが，特にnull callではステップ数が少なく処理時間が短いため，システムコールの負荷が大きく影響したためと思われる．

また，表 3の(2)の値が(1)よりも小さくなっているのは，CONFIG_MARKERを有効にすることでload_module内でSHF_ALLOCが有効になることによる影響であると考えられる．

3.3 トレース処理時の負荷評価

トレース処理中の負荷を表 5のような条件で測定する．

表 5:トレース処理の負荷測定の条件

	カーネルの状態
(6)	(4)でトレース（トレースポイント選択削減）
(7)	(5)でトレース（トレースポイントはすべて有効）

測定した結果を表 6に示す．なお，本節でも表の値は(1)の平均処理時間を1とした倍率で表す．

表 6：トレース処理時のLMbench結果

	null call	null I/O	stat	open/clos	slct TCP
(6)	14.768	11.472	4.169	5.863	9.388
(7)	30.563	31.569	27.443	31.569	13.519
	sig inst	sig hndl	fork proc	exec proc	sh proc
(6)	4.906	3.090	1.753	1.580	1.702
(7)	24.985	24.318	4.632	4.196	6.993

表 6から実際にトレースの採取時において，トレースポイントを減らしたことによって負荷が下がっていることがわかる．

また，(6)，(7)のカーネルを用いて，実機上でプログラムを何も動作させずに 30 秒間トレースを取得した場合の容量を比較すると，表 7のようにトレース容量も削減されることがわかる．

表 7：トレースの容量比較

	トレース容量
(6)	27.8MB
(7)	4.8MB

4 結論

組込みシステムにおいてさまざまな条件下でのLTTng の負荷について LMbench を用いて評価・検討を行った．

LTTng をカーネルに組み込んだ時点でも負荷がかかっているが，トレースポイント自体の負荷はあまり高くないということが判明した．また，不要なトレースポイントを削除することが特にトレース処理中の負荷の軽減には有効であることがわかった．また，トレースポイントの削除は負荷を軽減するだけではなく，トレースの容量を小さくし，バッファあふれを抑えることにも有効である．

今後の課題としては，負荷となる処理についてさらに詳細に解析し，さらに負荷を軽減できるように検討を行う必要がある．また，最新カーネルでの評価や他のトレーサの評価も行いたい．

参考文献

[1] “LTTng Project”, http://ltt.polymtl.ca/ .
[2] “The Linux Kernel Archives”, http://www.kernel.org/ .
[3] “System Tap”, http://sourceware.org/systemtap/ .
[4] “LMbench – Tool for Performance Analysis”, http://www.bitmover.com/lmbench/ .