令和 4 年度 特別研究報告書

トレースポイントを用いた Linux カーネルの動作の監視

龍谷大学 理工学部 情報メディア学科

学籍番号 ： T190524
氏名 ： 林　彩菜

指導教員 ： 三好 力 教授，芝 公仁 助教

内容梗概

一般的に，オペレーティングシステムは，潜在的なバグや脆弱性が存在し，機能の追加や修正によって新たなバグや脆弱性が発生する可能性がある．オペレーティングシステムの状態把握や異常検知を可能にするためには，オペレーティングシステムの主要なコンポーネントである Linux カーネルの挙動を監視することが最も重要である．そこで本論文では，カーネルのデバッグ用の機能であるトレースポイントを用いて，カーネル内での実際の動作を監視し，現在の状態を把握するシステムの作成を目的とする．

目 次

1　はじめに

Linux は世界中で開発が行われ続けているオペレーティングシステムである．オペレーティングシステムは計算機全体の動作に影響を及ぼすため，オペレーティングシステムに対する信頼性の向上が求められている．オペレーティングシステムに障害が発生した場合，オペレーティングシステム上のプロセスはすべて停止する．しかし一般的に，オペレーティングシステムは，潜在的なバグや脆弱性が存在し，機能の追加や修正によって新たなバグや脆弱性が発生する可能性がある．オペレーティングシステムの状態把握や異常検知を可能にするためには，オペレーティングシステムの主要なコンポーネントである Linux カーネルの挙動を監視することが最も重要である．そこで本論文では，カーネルのデバッグ用の機能であるトレースポイントを用いて，カーネル内での実際の動作を監視し，現在の状態を把握するシステムを作成すること目的する．Linux の Berkeley Packet Filter を拡張したカーネル内仮想マシンである eBPF を使用すると，トーレスポイント実行時に動作させるフィルタを作成でき，カーネル内の様々な情報を取得することができる．

以下，本論文では，2 章で関連研究について述べる．3 章では，トレースポイントおよびシステムの構成について述べる．4 章では，システムの処理について述べる．5 章では，3 つの状況で行った実験について述べる．6 章では評価を述べ，7 章で本論文のまとめついて述べる．

2　関連研究

[1] では，Linux システム上で，ユーザ空間とカーネル空間の両方における最新のトレーサを実際に比較し，異なるトレーサのオーバーヘッドを定量化するだけでなく，トレーサ内部への洞察を明らかにし，各トレーサのオーバーヘッドの原因を示す細かい指標をサンプリングするマイクロベンチマークを実装している．この研究では，ユーザがトレーサのオーバーヘッドを減らし，トレーサを最大限に活用するために，ユーザ固有の要件に基づいてトレーサを選択し，設定できるようにすることを目的としている．

[2] では，プロセス耐障害性向上システム Orthros を提案している．Orthros は，単一計算機上に複数の OS を同時に実行する．この特徴を活かし，カーネル間でメモリを監視する障害検知機構を実装する．この機構は，監視対象となる OS のカーネルメモリ上に存在するカーネルの実行状態を示すパラメータを定期的に監視する．

3　システムの構成

本章では，システムの背景，トレースポイントおよびトレースポイントを用いたカーネルの動作を監視するためのシステムの構成について述べる．

3.1　システムの背景

オペレーティングシステムの状態把握や異常検知を行うためには，図 1 に示すような，監視して取得した状態を事前に取得したものと比較し，得られた類似度をもとに現在の状態を推測することが必要である．本論文では，その推測を可能にするためのカーネルの動作を監視するシステムを作成する．

3.2　トレースポイント

トレースポイントとは，カーネルのデバッグ用の機能で，イベントをフックする仕組みである．enable のときにハンドラが実行される．カーネルコード内に関数呼び出し

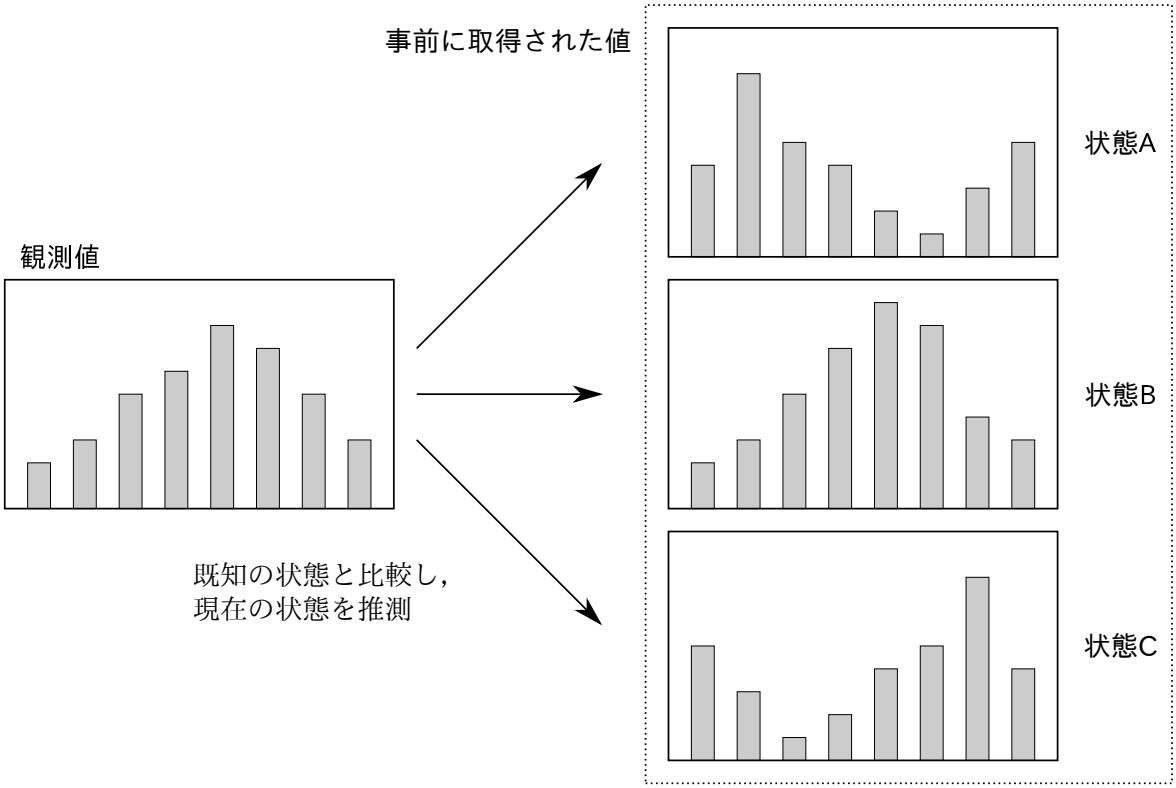

図 1　現在の状態の推測

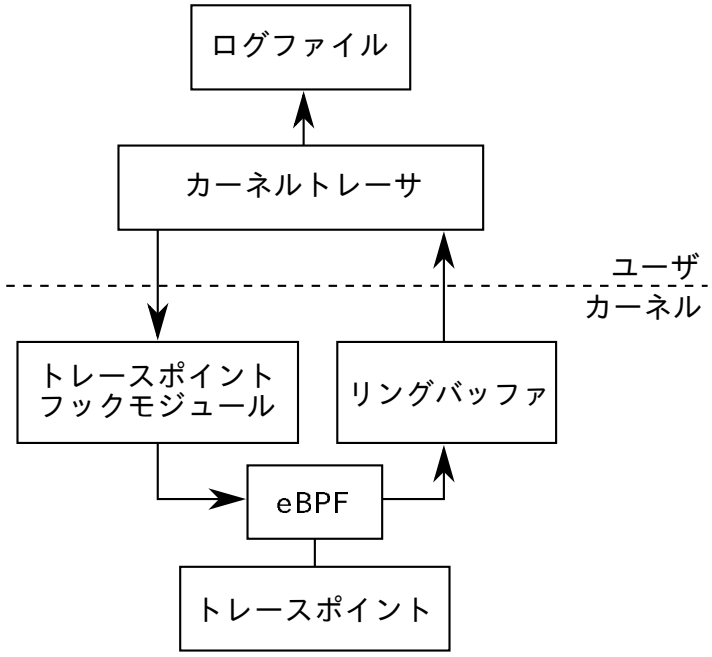

図 2　システムの構成

が埋め込まれているため，ローカル変数も参照でき，イベントとして定義された値にのみアクセスが可能である．カーネル内仮想マシンである eBPF を使用すると，トレースポイント実行時に動作させるフィルタを作成でき，カーネル内の様々な情報を取得することができる．本論文では，この機能を用いてカーネルの動作を監視する．

3.3　システムの構成

図 2 にシステムの構成を示す．この構成は，カールトレーサからトレースポイントフックモジュールを呼び出し，一定数のトレースポイントをユーザ空間にログとして保存する．カーネルトレーサは python スクリプトで，トレースポイントの表の作成やカーネルモジュールの呼び出し，データを保存するログファイルの作成を行う．トレースポイントフックモジュールは Linux カーネルモジュールで，イベント発生時毎にそのトレースポイントのデータとしてトレースポイントの識別子とタイムスタンプを格納する．

4　システムの処理

本章では，システムの全体の処理と各処理について述べる．

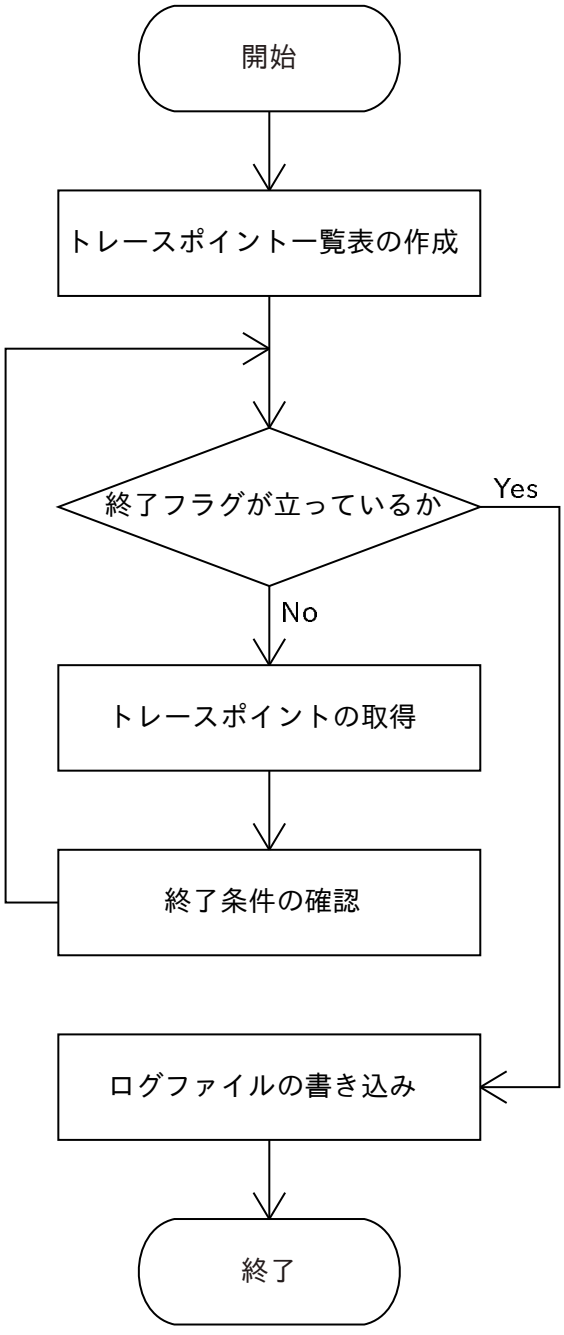

図 3　システムの処理

tpid	gid	tracepoint name
1	1	sst_ipc_msg_tx
2	1	sst_ipc_msg_rx
3	1	sst_ipc_inbox_rdata
.	.	.
.	.	.
.	.	.

gid	group name
1	btrfs
2	xfs
3	kvmmmu
.	.
.	.
.	.

図 4　トレースポイント表

4.1　処理全体の流れ

図 3 にシステムが行う処理の全体の流れを示す．/sys/kernel/tracing/events から取得したトレースポイントに識別子を割り当て，トレースポイント名とともに表として作成する．図 4 はトレースポイント表の例である．トレースポイントグループは，トレースポイントをサブシステムごとに分類したものである．

- tpid: トレースポイント識別子
- tracepoint name: トレースポイント名
- gid: トレースポイントグループ識別子
- group name: グループ名

以下のトレースポイントは eBPF に未対応であったり，取得率が非常に高く他のトレースポイントの監視に支障をきたしたりする為，対象外とする．

- ftrace: カーネルに組み込まれているトレース機構
- syscalls: システムコール
- irq_work_entry: 割り込みの入力
- irq_work_exit: 割り込みの出力
- write_msr: CPU 固有レジスタの書き込み
- cpu_idel: CPU が何もしていない状態
- row_syscalls: システムコールにおける記憶媒体のデータと元のデータの整合性の確認

timestamp	tpid	gid	tracepoint name
2069046799106	1213	100	tick_stop
2069046801014	1210	100	hrtimer_cancel
2069046801298	1207	100	hrtimer_start
.	.	.	.
.	.	.	.
.	.	.	.

図 5　ログファイル

- rcu_utilization: 排他制御を実装する同期機構の利用
- sched_switch: 実行中のプロセスの状態の保存と別の保存されているプロセスの状態の復元
- sched_stat_runtime: プロセスの実行状態
- sched_waking: プロセスを起こしている状態
- sched_wakeup: プロセスを起こした状態
- rseq_update: rseq システムコールによる更新
- x86_fpu_regs_deactivated: fpu レジスタの無効化
- hrtimer_cancel: カーネルタイマのキャンセル
- hrtimer_start: カーネルタイマの開始
- tlb_flush: TLB のフラッシュ
- tick_stop: システムクロックの周期の停止

このトレースポイント表を用いて，イベント発生時のトレースポイントのデータを取得する．終了条件を満たした後，取得したトレースポイントのデータであるタイムスタンプおよびトレースポイント識別子をログファイルに書き込み，システムを終了する．ログファイルの例を図 5 に示す．

- timestamp: トレースポイント取得時の日時

4.2 各処理

(1) トレースポイント表の作成

/sys/kernel/tracing/events/以下にあるサブシステムのディレクトリをトレースポイントグループとする．サブシステムのディレクトリの下にあるディレクトリ名がトレースポイントである．各トレースポイントと各トレースポイントグループに識別子を割り当て，トレースポイント名やトレースポイントグループ名とともにトレースポイント表を作成する．

(2) カーネルトレーサからのトレースポイントフックモジュールの呼び出し

トレースポイントフックモジュールは，トレースポイント識別子およびイベント発生時のタイムスタンプを格納する構造体を作成し，リングバッファを作成する．そしてイベント発生時，リングバッファにデータを登録する．

カーネルトレーサは，リングバッファを開き，データを受け取った際に呼ばれるコールバック関数を指定する．コールバック関数で受け取ったデータを，トレースポイントフックモジュールの構造体をもとに BPF データに変換する．リングバッファがデータを受け取った後，コールバック関数を呼ぶ．それをトレースポイント数があらかじめ定義した数に達するまで繰り返す．

(3) ログファイルの作成

カーネルトレーサが終了条件を満たした後，ログファイルを作成し，タイムスタンプとトレースポイント識別子を書き込む．トレースポイント表からトレースポイント識別子に対応するトレースポイントグループ識別子およびトレースポイント名も同時に書き込む．

5　実験

本章では，行った実験について述べる．本システムを用いて実際のカーネルの挙動を可視化し，取得数の多いトレースポイントが，環境ごとに実際に使用されているか確認することを目的とする．

スクリプトを実行する際に必要なオープンファイル数の上限値を最大の 104856 に設定し，1256 個のトレースポイントを監視対象として 100000 個のイベントを取得する実験を行った．表 1 に実験環境を示す．

実験は 3 つの状況で行った．それぞれの結果を，トレースポイント識別番号と時間，トレースポイントグループ識別子と時間の 2 つのグラフで示す．

5.1　他の処理を行わない場合

スクリプト実行以外の処理を行わなかった場合の結果を図 6，7 に示す．最も取得数の多かったトレースポイント 5 つは以下のとおりである．トレースポイントグループは，CPU のホットプラグが一番多かった．

- read_msr: CPU 固有レジスタの書き込み
- workqueue_execute_start: カーネルスレッドが非同期処理をキューから取り出し実行
- cpuhp_exit: CPU 特定 core の無効化
- cpuhp_enter: CPU 特定 core の有効化
- cpuhp_multi_enter: CPU 特定 core の複数の有効化

表 1　実験環境

項目	値
CPU	Intel Core i5-9400F 2.90GHz
メモリ	DDR4 2666MHz 16GB
OS	Ubuntu 22.04.1 LTS
Kernel	Linux 5.15.0-58

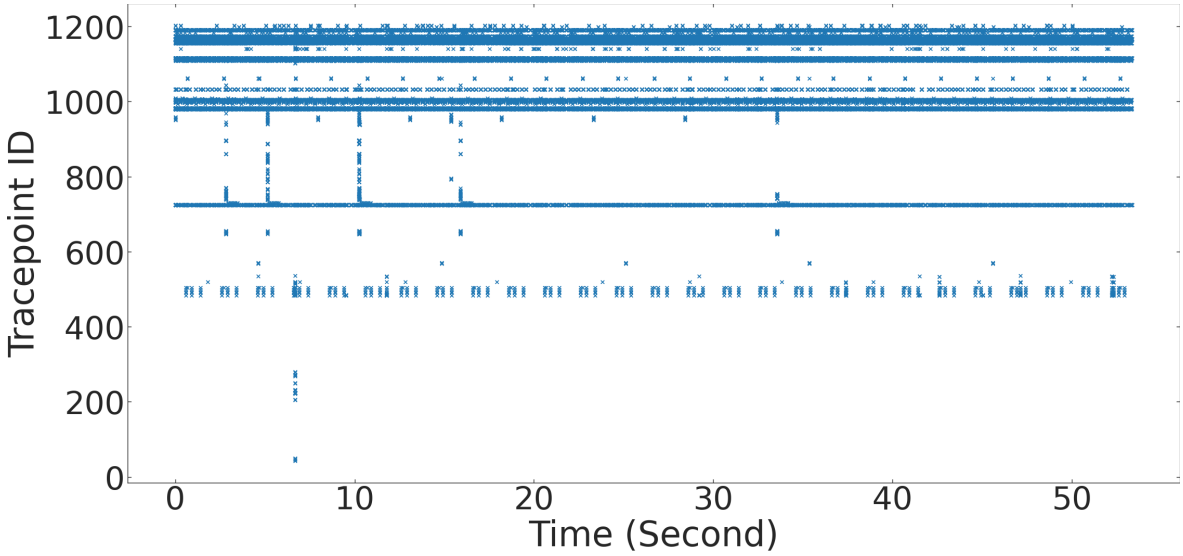

図 6　他の処理を行わない場合

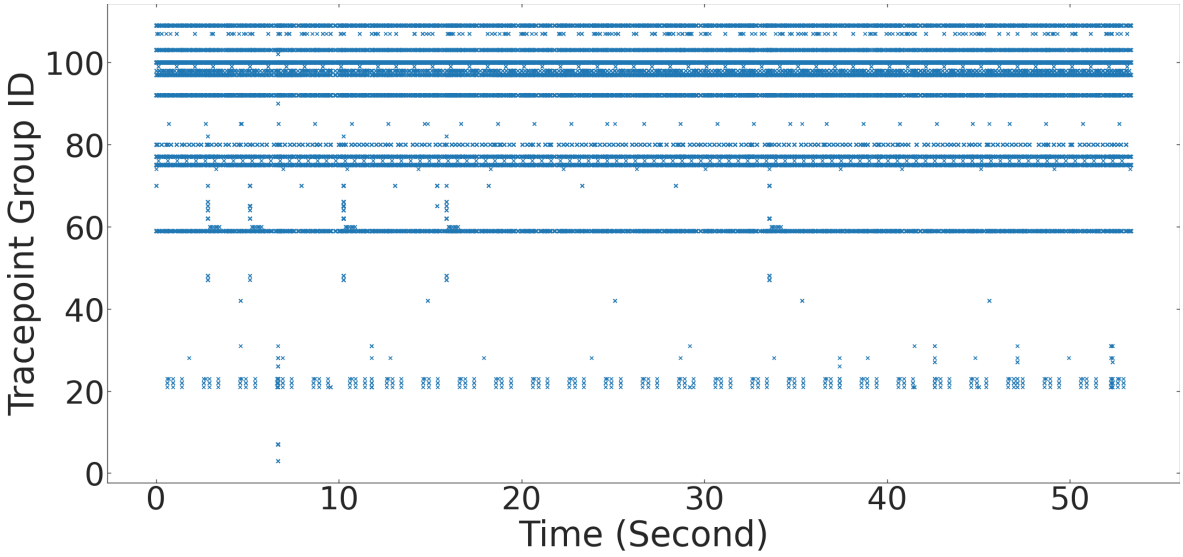

図 7　他の処理を行わない場合

5.2　Linux カーネルビルド実行時

Linux カーネルコンパイルパッケージのビルドの実行中の結果を図 9，10 に示す．カーネルビルドは，配布されているカーネルのソースコードをバイナリ形式に変換することで，図 8 に示すように，カーネルが備える各機能の有効化/無効化，モジュールファイル化する等の設定ができる．最も取得数の多かったトレースポイント 5 つは，以下のとおりである．トレースポイントグループは，CPU のホットプラグが一番多く，他

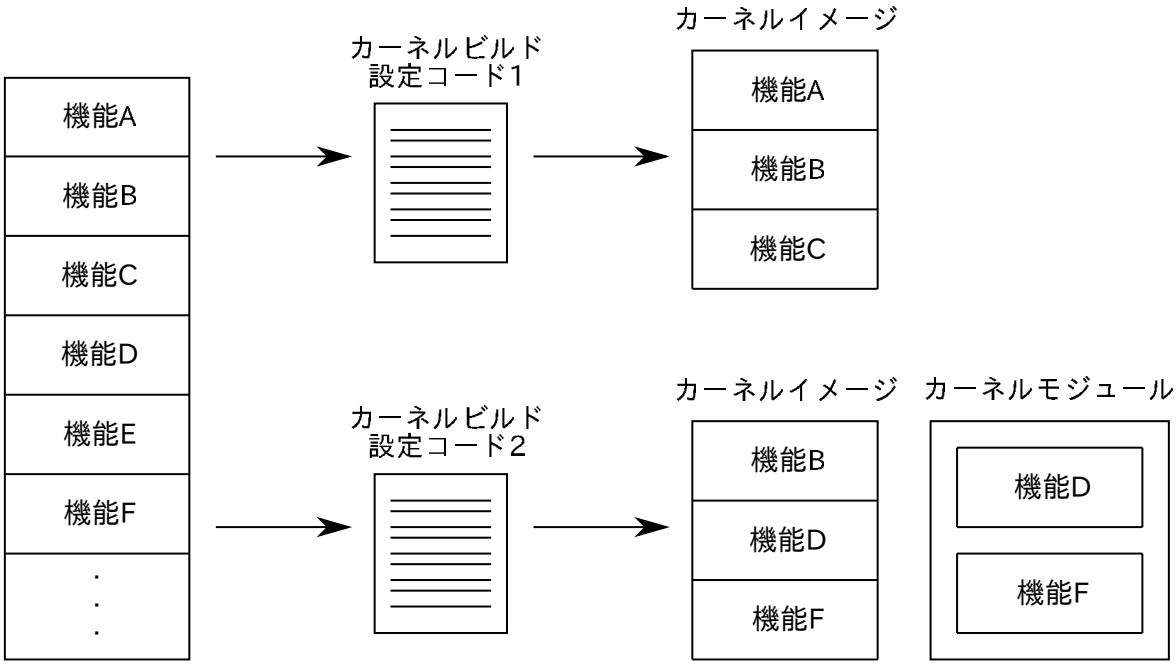

図 8　カーネルビルド

の処理を行わなかった場合と同様になった．

- read_msr: CPU 固有レジスタの書き込み
- workqueue_execute_start: カーネルスレッドが非同期処理をキューから取り出し実行
- cpuhp_exit: CPU 特定 core の無効化
- cpuhp_enter: CPU 特定 core の有効化
- cpuhp_multi_enter: CPU 特定 core の複数の有効化

5.3　Apache 実行時

Web サーバである Apache の実行中の結果を図 12，13 に示す．今回の実験では，Web サーバのベンチマークを行うためのツールである Apache Bench を使用した．Apache Bench は，図 11 に示すように，対象となる Web サーバにリクエストを行うことができ，リクエスト数を 100，同時接続数を 100 に設定し，実験を行った．最も取得数の多かったトレースポイント 5 つは，以下のとおりである．トレースポイントグループは，デフラグメンテーションが一番多かった．

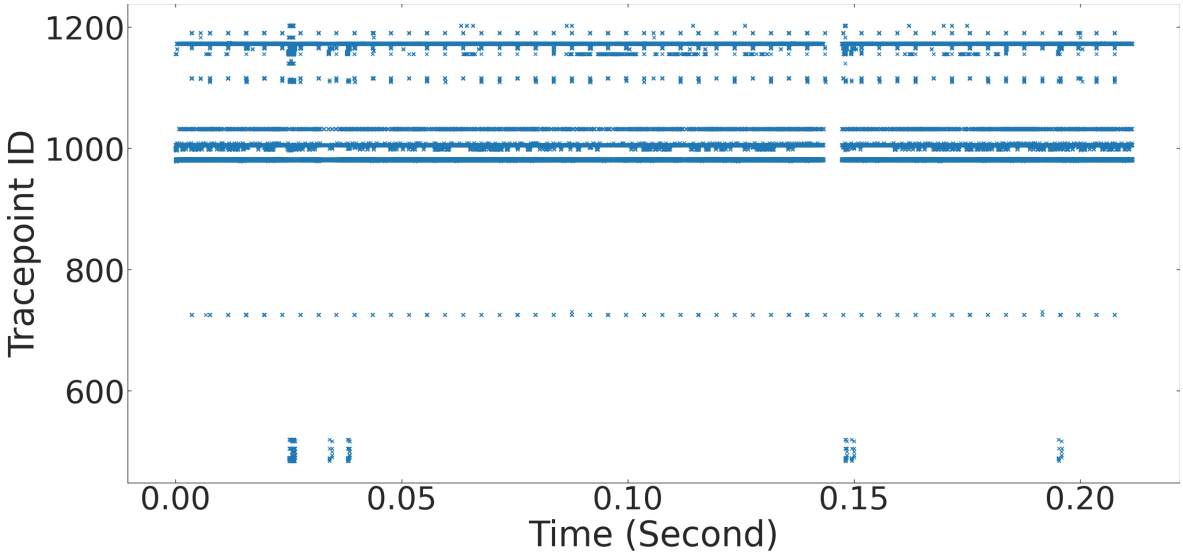

図 9　カーネルビルド実行中

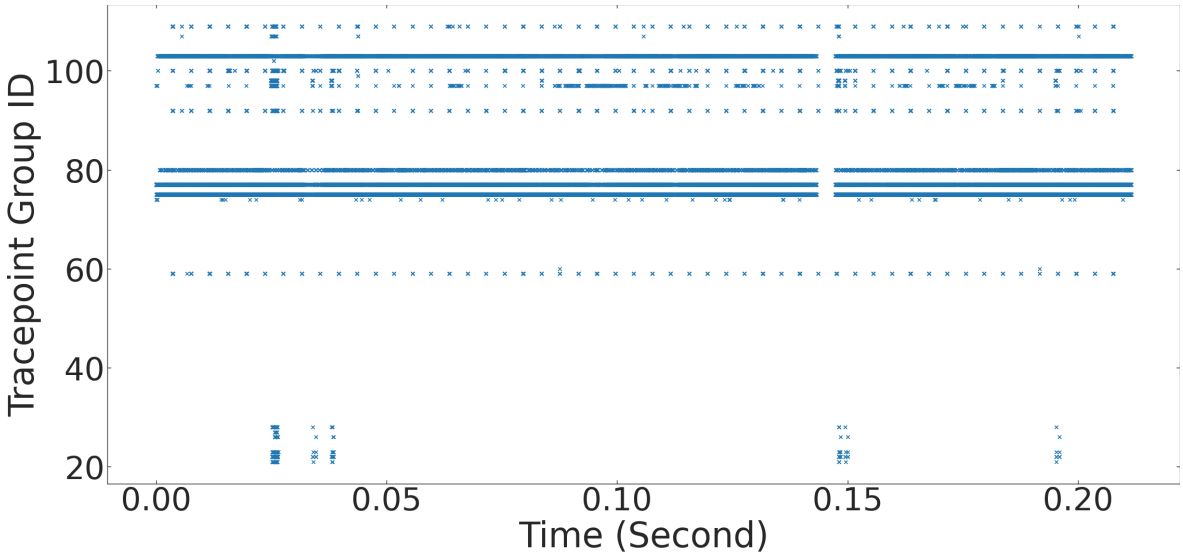

図 10　カーネルコンパイル実行中

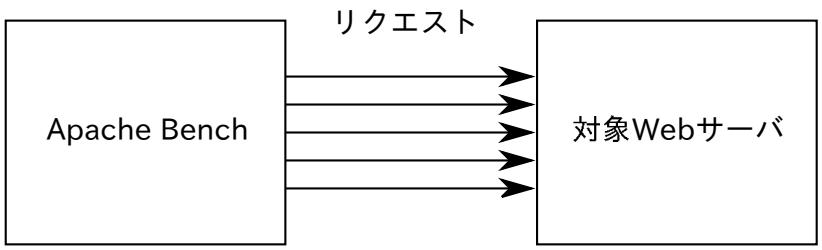

図 11　Apache Bench

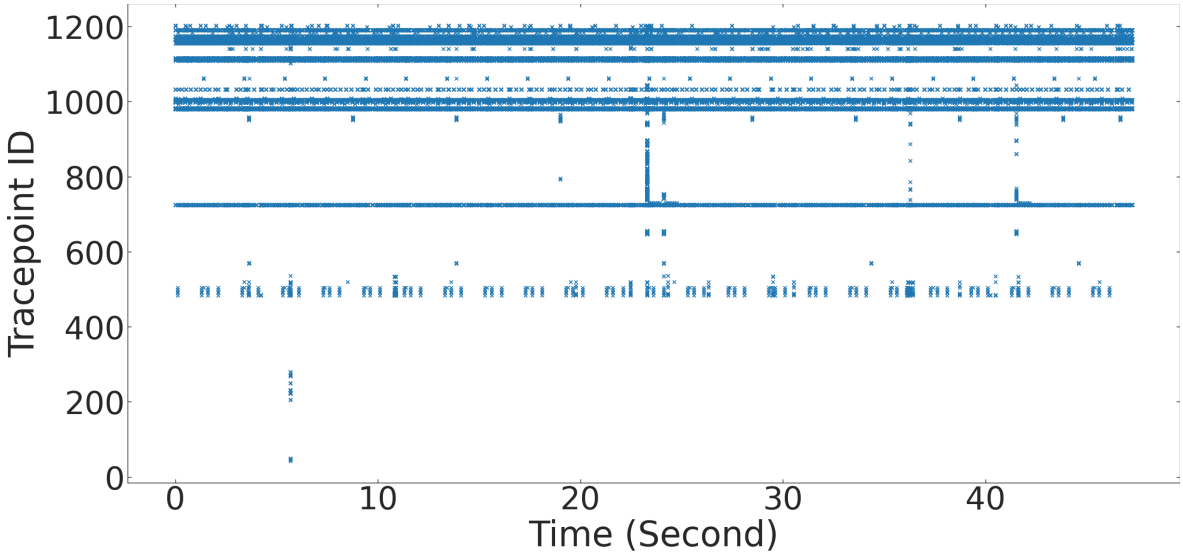

図 12　Apache の実行中

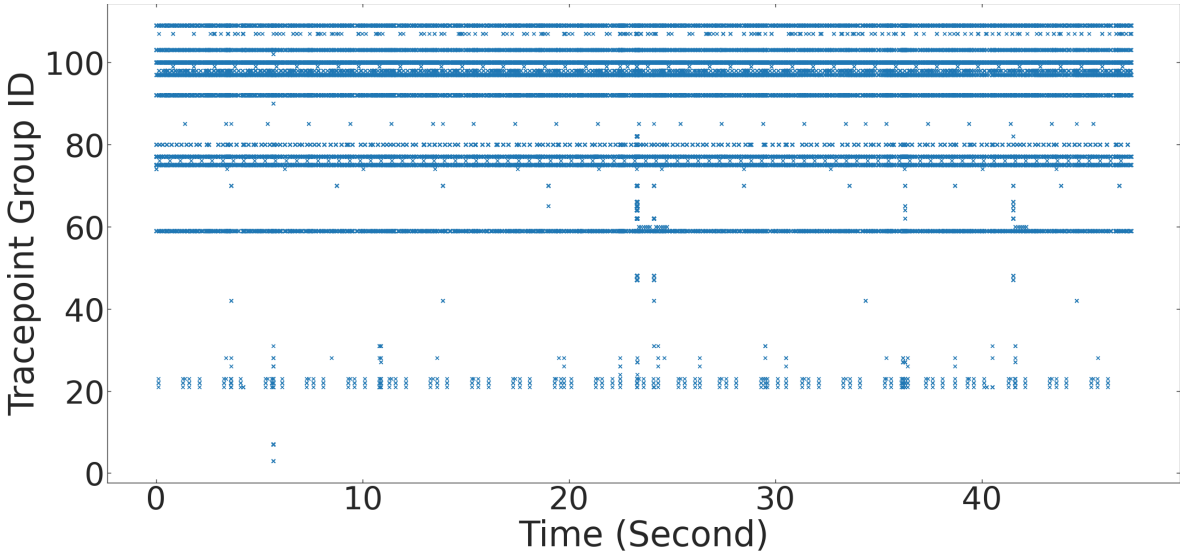

図 13　Apache の実行中

- mm_compaction_isolate_migratepages: デフラグ機能によるページマイグレーション
- mm_compaction_isolate_freepages: デフラグ機能によるフリーページの探索
- mm_compaction_migratepages: デフラグ機能によるページマイグレーション
- pseudo_lock_12: 疑似ファイルシステムのロック
- mm_page_alloc_extfrag: ext フラグのページ割り当て

6　評価

本章では，オーバヘッドによるシステムの評価を行うこと目的とする．測定を行った環境は，表 1 の実験を行った環境と同様である．本システムを使用している時と使用していない時の Linux カーネルパッケージのビルドの時間を測定し，評価を行った．表 2 に，各測定時間を示す．

システム使用時とシステム未使用時の時間の差は，248.89 秒となった．これは，およそ 1.12%の増加であり，オーバヘッドはあまり大きくないと考えられる．

表 2　測定時間の比較

項目	測定時間 (s)
システム使用時	2303.36
システム未使用時	2054.47

7　おわりに

本論文では，トレースポイントを用いて Linux カーネルの動作を監視するシステムについて述べた．本システムは，いくつかの環境で Linux カーネルがどのような動きをしているのか，トレースポイントを用いることで，具体的にどのような処理が多いのかを明らかにすることができた．これによって，Linux カーネルの動作の監視し，状態を把握することが実現される．

参考文献

[1] Gebai, M. and Dagenais, M. R.: Survey and Analysis of Kernel and Userspace Tracers on Linux: Design, Implementation, and Overhead, *ACM Computing Surveys*, Vol. 51, pp. 1–33 (2019).

[2] 松下　馨，岩間響子，瀧本栄二，毛利公一，齋藤彰一：多重 OS 実行環境におけるカーネル間メモリ監視による障害検知機構の実装，情報処理学会研究報告，Vol. 2017-OS-139, No. 2, pp. 1–8 (2017).