

“Low-intrusion”モデルによるスクリプト言語用デバッガの開発

Low-intrusion multi-thread debugger for network software written in scripting languages

B-6

伊藤 泰† 永井 和宏† 佐藤 規男†
Yasushi Itoh Kazuhiro Nagai Norio Sato

1. はじめに

OS やハードウェアの性能の向上によって十分な実行速度が得られるようになったことや、その生産性の高さからスクリプト言語が普及してきた。ネットワークプログラミングにおいてはネットワークがボトルネックとなるのであまりプログラム自体の性能を気にしなくてよいことなどから、スクリプト言語が使用されることが多い。このため多くのスクリプト言語ではマルチスレッドプログラミングがサポートされている。

本論文で提案するデバuggは、主にネットワークプログラミングにおけるマルチスレッドプログラムをデバuggする際に、柔軟なスレッドの扱いをすることでより簡単にデバuggできる環境を提供する。対象言語は先進的で明快な言語構造と豊富なライブラリモジュールを持ち、オープンソースである Python^{[1][3]}とした。

続く 2 章では本デバッグの骨子となるスレッドの扱いや適用範囲について述べ、3 章では本デバッグの要求条件を示す。具体的な機能及び実装方法については 4 章で述べ、5 章でまとめを行う。

2. システムの特徴と適用領域

本論文で提案するデバッグはネットワークタイプのデバッグである。デバッグを介してターゲットホストでデバッグを自動的に起動することが可能である。手でデバッグを起動しておいて他ホストからデバッグを接続することも可能である。また、同時に複数のプロセスを扱えるので、ネットワークプログラムにて通信元と通信先の双方を同時にデバッグ可能である。

従来あるデバッガの振る舞いは“Stop-the-world”モデルである。“Stop-the-world”モデルではデバッグ操作を行うため対象プロセス内の全てのスレッドが停止する。Python の標準的なデバッガ Pdb もシングルスレッドプログラムにしか対応していない。

これに対し本論文で提案する“Low-intrusion”モデルではデバッグ操作を行うために 1 つのスレッドを停止することはあっても、プログラム(プロセス)全体を停止する必要が無く、他のスレッドは実行を継続する。また、他のスレッドに対しても個別にデバッグすることも可能である。

“Low-intrusion”モデルが有用となるのは、追跡すべき対象スレッド以外のスレッドに対して影響を最小限にした状態で、デバッグ対象スレッド以外のスレッドが何らかの処理を依頼または受け付けるような場合である。例としては、Boss-Worker パターンの並行サーバプログラムや Producer-Consumer パターンのプログラムが挙げられる。このような場合には全スレッドをユーザの制御下に置く必要がある。

3. 要求条件

“Low-intrusion”であることの他に本システムに求められる条件として以下の A, B が挙げられる.

A) スレッドモジュールの利用する C レベルのスレッド実装に依存しないこと。

Python のスレッドモジュールは C レベルのスレッドライブラリに依存して実装され、そのスレッドライブラリを複数の種類から選択できるようになっている。このため、デバグガの実装に特定のネイティブスレッドライブラリを利用することを想定できない。また、Python のみで書かれたプログラムはプラットフォーム移植性が高いのでデバグガもそうである必要がある。

B) GUIを備えていること.

本システムは多数のスレッドを並行してデバッグすることを想定しているので、スレッドの指定と操作を容易にする GUI が必須である。

4. 機能と実装

本章では本論文で提案するデバッガのシステム構成を述べる。

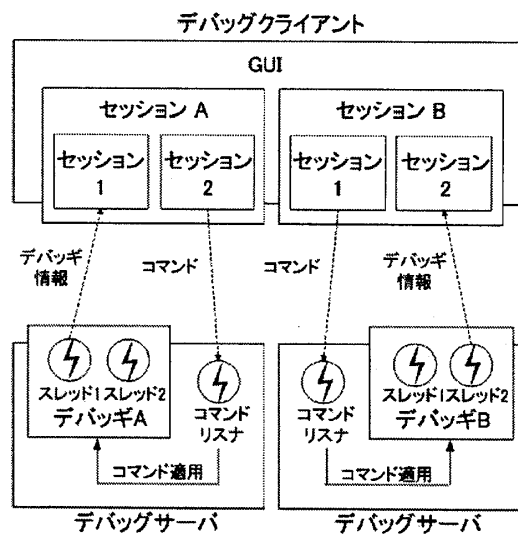


図 1. システム構成.

本システムは図 1 のようにユーザインタフェースを提供するクライアントとローカル・リモートホストで動作しデバッグを制御する一つ以上のサーバから構成される。

“コマンド”は GUI 経由で送られるデバッグコマンドで、“デバッグ情報”はコマンドにより変化したデバッグの状態やコマンドによる問い合わせへの応答である。“Low-intrusion”モデルではコマンドとデバッグ情報は非同期にやり取りされる。

4.1 機能

図2に示すスクリーンショットを基に説明する。

ウィンドウ①にデバッグ対象のプロセスとスレッドがツリーで表示される。スレッドノードを選択することにより、他の3つのウィンドウがそのスレッドとのデバッグセッションに対応したものに切り替わる。他の3つのウィンドウはソースブラウザとコマンドシェルとデバッグのアウトプットである。

ウィンドウ②はデバッグの標準出力やサーバの出力を表示する。

ウィンドウ③はソースブラウザである。ブレークポイントの操作などが行える。

ウィンドウ④はコマンド入力用 CUI である。

ツールバーにはサーバとの接続などのコマンドや、頻繁に利用するステップ実行等に対応するアイコンがある。

4.2 クライアントの実装

クライアント(図1上)は GUI 部とサーバと通信するデバッグプロキシ部で構成され、GUI イベントとサーバからのイベントを監視する。GUI の記述に PyQt¹モジュール¹⁴を用いることにより、シングルスレッドでこれらのイベントを受信することが容易に出来、シンプルな構成になる²。

4.3 サーバの実装

サーバ(図1下)は、コマンドリスナとコールバックスクリプトと tmc モジュールから構成した。

コマンドリスナはクライアントからのコマンドを受け取る専用のデーモンスレッドである。コマンドリスナは受付けたコマンド³を実行し、コマンドの機能に応じたフィルタリング情報⁴を設定する。

コールバックスクリプトはスレッドがトレース状態であ

る時にインタプリタから呼び出される。コールバックスクリプトはデバッグ情報をフィルタリングしてクライアントに通知し、必要なら該当スレッドの実行を中断する。

tmc モジュールは Python インタプリタの拡張部でデバッグの他の部分に、スレッド毎のトレースフック設定機能を提供する。実装は C 記述のシェアドライブラリとした。

また、インタプリタは POSIX スレッド²をブラックボックスとして利用する実装である⁵ため、スレッドの状態情報は外部で補完する必要がある。これは Threading クラスをデバッグ時に動的に拡張することにより実現した。

5. まとめ

本研究にてスクリプト言語 Python 用のネットワークタイプマルチスレッドデバッグを試作した。“Low-intrusion”モデルを適用することでマルチスレッドプログラムをデバッグする際の新しい方向性を示した。

今後は Ruby 等他言語へのデバッグシステムの移植や、“Low-intrusion”と“High-intrusion”の切替え、言語の動的特徴を利用したフィルタリング等を予定している。

参考文献

- [1] Python Language Website, <http://www.python.org/>
- [2] David Butenhof, “Programming with POSIX Threads”, Addison Wesley Longman (1997), 邦訳: “POSIX スレッド プログラミング”, 油井 尊, 宗方 あゆむ 訳, アジソン・ウェスレイ・パブリッシャーズ・ジャパン株式会社 (1998).
- [3] David M. Beazley, “Python Essential Reference”, NEW-RIDERS (2000); 邦訳: “Python テクニカルリファレンス”, 習志野弥治朗 訳, ピアソン・エデュケーション (2000).
- [4] PyQt, <http://www.riverbankcomputing.co.uk/pyqt/>

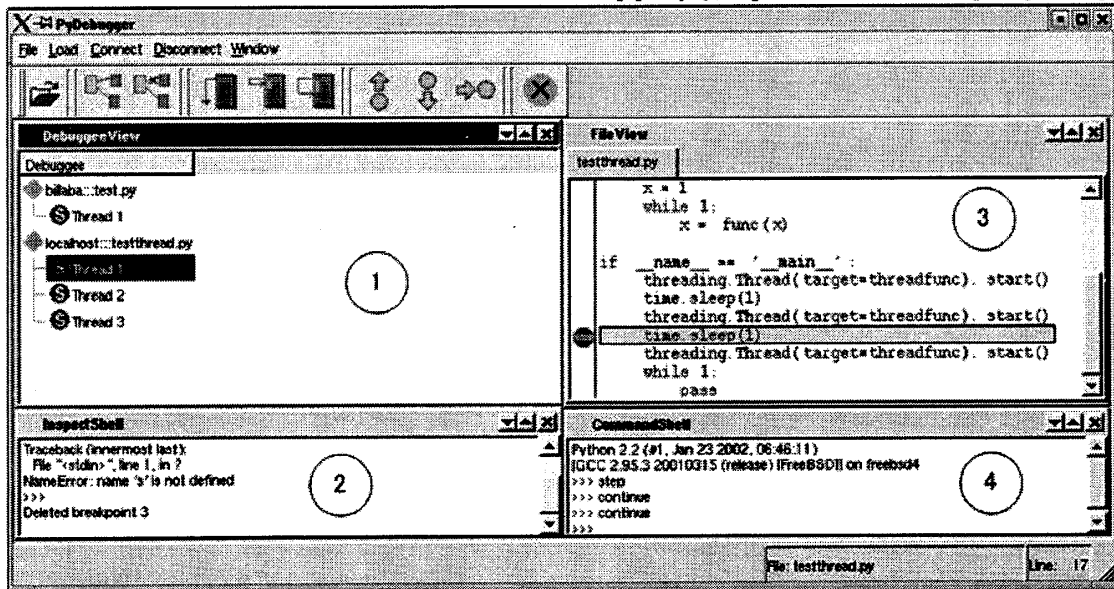


図2. GUIのスクリーンショット。

¹ Qt ツールキットの Python バインディング

² Qt によりウィジェットの一種として提供されているソケットを用いる。Qt はスレッドセーフでないのでこのウィジェットは有用である。

³ 式評価、スレッド、スタック、ソースの読出し

⁴ ブレークポイント設定とステップ実行要求

⁵ GNU Pth 等の他のスレッドライブラリにもオプションとして対応している。