Oracle® Solaris Studio 12.3: DLight チュートリアル

2011 年 12 月

ORACLE®

DLight チュートリアルでは、DLight を使用して実行可能ファイルを起動し、DLight の監視グラフおよびツールを使用して実行中のプログラムからのプロファイリングデータを調べる方法を示します。チュートリアルでは、プロセスに接続し、プロセスとそのすべての子プロセスおよびスレッドからのプロファイリングデータを調べる方法も示します。チュートリアルには次のトピックが含まれています。

概要

DLight は、Oracle Solaris で実行されているプログラムに関する情報を Oracle Solaris 動的トレース (DTrace) テクノロジを使用して収集する、対話型グラフィカル可観測性ツールです。DLight を使用すると、D スクリプト作成言語の使用方法の知識がなくても、DTrace でプログラムを分析できます。DLight は、複数の DTrace スクリプトを同期された方法で実行しつつ、その出力をグラフィカル表示することで、ユーザーがアプリケーションの実行時の問題を追跡してその根本原因を特定するのを支援します。

DLight の基本的なワークフローは次のとおりです。

1. ターゲットを作成して、プロファイルするプロセスまたは実行可能ファイルを定義します
2. ターゲットを実行します
3. ターゲットを実行すると開く DLight ツールで、データを調べます
4. ツールと対話してコードの問題点を確認します
5. 必要に応じてターゲットを停止します

ターゲットを作成するときに、ターゲットの種類およびターゲットを実行するべきシステム (ローカルまたはリモートシステム) を指定できます。

DLight ターゲットの種類です。

実行可能なターゲット	まだ実行されていないプログラム
プロセスターゲット	実行中のプロセス
プロセスツリーターゲット	実行中のプロセスおよびそのプロセスによって生成されたすべての子プロセス

実行可能なターゲットまたはプロセスターゲットを実行すると、各ツールによって「DLight 実行モニター」ウィンドウのグラフに使用状況情報が表示されます。

プロセスツリーターゲットを実行すると、ツールは、複数のプロセスおよびスレッドの活動に関する詳細情報を提供する「プロセスツリーのプロファイリング」ウィンドウに表示されます。

「実行モニター」ウィンドウと「プロセスツリーのプロファイリング」ウィンドウのツールについては、このマニュアルで後述します。

次の節で説明されているように、ユーザーは DLight を実行するシステム上で特定の DTrace 関連の Oracle Solaris 権限を持っている必要があります。

ヒント – ターゲット、リモートホストのプロセスのプロファイリング、およびツール構成の変更については、「ヘルプ」⇒「ヘルプの目次」を参照してください。

DLight 用の Oracle Solaris 権限の設定

DLight を実行するユーザーには、Oracle Solaris 権限 `dtrace_proc`、`dtrace_user`、および `dtrace_kernel` が割り当てられている必要があります。これらの権限は、DTrace 機能へのユーザーのアクセスを制御します。ユーザーのユーザー名にこれらの権限が割り当てられていない場合、DLight は、権限を設定できる root などの管理者アカウントのパスワードを要求します。このプロンプトは、DLight セッションではじめて実行可能なターゲットを実行するかターゲットプロセスに接続するときに表示されます。

正しいパスワードを入力すると、DLight を実行するプロセスに必要な権限が、その DLight セッションの期間、管理者アカウントを使用して割り当てられます。自分のユーザー名にこれらの権限が恒久的に割り当てられているシステムで DLight を使用する場合、または管理者としてログインする場合は、DLight で管理者パスワードは要求されません。

注 – これらの権限は、ユーザーがカーネルの実行をのぞき込むことを可能にするため、注意して使用するようにしてください。ソフトウェア開発に使用するシステムでのみユーザーに対してこれらの権限を有効にするようにし、本稼働システムでは有効にしてはいけません。

自分の Solaris 権限を確認するには、コマンドプロンプトで次のように入力します。

```
$ /bin/ppriv $$
```

アカウントに必要な権限がある場合は、`ppriv` コマンドで次のように返されるはずです。

```
E: basic,dtrace_kernel,dtrace_proc,dtrace_user
I: basic,dtrace_kernel,dtrace_proc,dtrace_user
P: basic,dtrace_kernel,dtrace_proc,dtrace_user
L: all
```

`I:` で始まる行は、ユーザーのシェルから起動されるプログラムによって継承される権限を指定するため、重要です。必要な継承可能権限が自分のアカウントになく、管理者権限やシステムに対する root アクセス権を持っていない場合は、システム管理者に依頼して `dtrace_proc`、`dtrace_user`、および `dtrace_kernel` 継承可能権限をアカウントに追加してください。

管理者権限またはシステムに対する root アクセス権を持っている場合は、次に説明するように、必要な権限を自分のアカウントに付与できます。

必要な DTrace 権限をユーザーアカウントに恒久的に付与するには:

1. 権限を変更するユーザーアカウントがシステムからログアウトしていることを確認してください。
2. スーパーユーザー (root) または別の管理者ユーザーとしてログインします。
3. コマンドプロンプトで次のように入力し、*username* は変更するユーザーアカウントで置き換えます。

   ```
   $ usermod -K defaultpriv=basic,dtrace_kernel,dtrace_user,dtrace_proc username
   ```

ユーザーの実行中のシェルプロセスに権限を割り当てて、これらの拡張された権限をユーザーに一時的に付与することもできます。

必要な DTrace 権限をユーザーアカウントに一時的に付与するには:

1. ユーザーのコマンドプロンプトで次のように入力して、ユーザーのシェルプロセスのプロセス ID を調べます。

   ```
   $ echo $$
   ```

2. 別のターミナルで、スーパーユーザー (root) または別の管理者ユーザーとしてログインします。
3. 次のように入力し、*process-ID* は `echo` コマンドから返されたプロセス ID で置き換えます。

   ```
   $ ppriv -s I+dtrace_user,dtrace_proc,dtrace_kernel process-ID
   ```

   これで、*process-ID* で指定されたシェルに入力されるすべてのコマンドに、必要な権限が継承されるようになります。ユーザーはこのシェルで DLight を起動するようにしてください。

Oracle Solaris Studio サンプルアプリケーションのダウンロード

このチュートリアルでは、`ProfilingDemo` というサンプルプログラムを使用します。

サンプルプログラムのソースコードは、Oracle Solaris Studio 12.3 Sample Applications Web ページ (`http://www.oracle.com/technetwork/server-storage/solarisstudio/downloads/solaris-studio-samples-1408618.html`)で、サンプルアプリケーションの zip ファイルから入手できます。

まだ行なっていない場合、サンプルアプリケーションの zip ファイルをダウンロードし、選択したディレクトリに展開します。サンプルアプリケーションは、`SolarisStudioSampleApplications` ディレクトリの DLight サブディレクトリにあります。

サンプルアプリケーションの構築

この手順では、すでにサンプルアプリケーションの zip ファイルをダウンロードし、選択したディレクトリに展開してあると仮定します。

1. `ProfilingDemo` ディレクトリを、ホームディレクトリなどユーザー専用の作業領域にコピーします。

   ```
   % cp -r SolarisStudioSampleApplications/DLight/ProfilingDemo ~/ProfilingDemo
   ```

2. ユーザーのプログラムコピーを構築します。

   ```
   % cd ~/ProfilingDemo
   % make
   ```

   プログラム `profilingdemo` は、デバッグ情報.を生成する `-g` オプションを使用して構築されます。このオプションなしでコンパイルすると、DLight によってプログラムの実行時データの収集と表示は行われますが、関数のソースコードを表示する機能は動作しません。

DLight の起動

次のように入力して DLight を起動します。

```
% dlight
```

Oracle Solaris Studio のディレクトリをパスに追加していない場合は、次のように入力して DLight を起動できます。

% */studio-installation-directory*`/bin/dlight`

デフォルトのインストールディレクトリは `/opt/solarisstudio12.3/bin` です。

サンプルアプリケーションのプロファイリング

DLight 実行可能ターゲットを使用すると、まだ実行されていないアプリケーションを DLight で起動してプロファイルできます。この節では、実行可能ターゲットを作成して実行し、サンプルアプリケーションの実行中に DLight によって表示される動的グラフを観察します。

1. DLight で、「新規 DLight ターゲット」ボタンをクリックします  。「新規実行可能ターゲットを作成」ダイアログボックスが開き、「実行可能なターゲット」タブが選択されます。
2. 「新規実行可能ターゲットを作成」ダイアログボックスで:
   a. 「実行」フィールドに `profilingdemo` 実行可能ファイルへのパス名を入力するか、「参照」をクリックして `profilingdemo` 実行可能ファイルに移動してこれを開きます。
   b. 「実行」をクリックします。

      新しい実行可能ターゲットが保存されて「DLight ターゲット」ウィンドウに表示され、DLight によってターゲットが実行されます。
   c. 3 ページの「DLight 用の Oracle Solaris 権限の設定」で説明されているように、DTrace のための十分な権限を持っていない場合、root パスワードを要求されます。`root` ユーザー名を適切な管理者名で置き換えれば、root 以外の管理者ユーザーの資格を使用することもできます。
3. `profilingdemo` アプリケーションが実行を開始し、実行中のアプリケーションに関して収集されたプロファイリングデータの動的グラフが「実行モニター」ウィンドウに表示されます。

ヒント – すべてのグラフを表示できない場合は、「ホスト情報」ウィンドウを閉じてください。「実行モニター」ウィンドウの横にあるスクロールバーを使用して、より多くのグラフを表示することもできます。

---

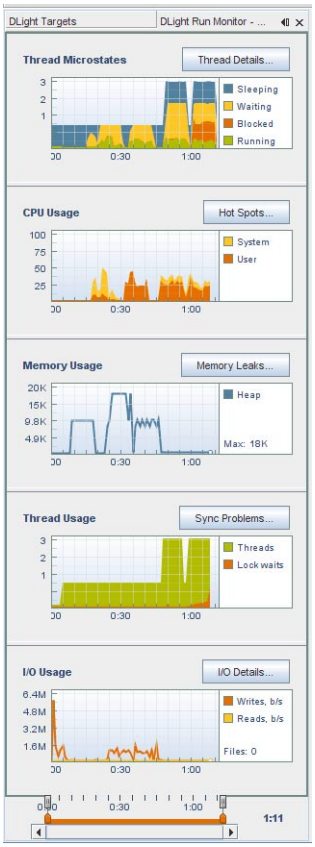
4. 出力ウィンドウに `profilingdemo` プログラムの作業内容が示されるので、この出力とグラフに示されたデータを照合できます。出力ウィンドウをクリックし、実行が終了するまで、プログラムに促されるたびに Enter キーを押します。

スライダコントロールの使用

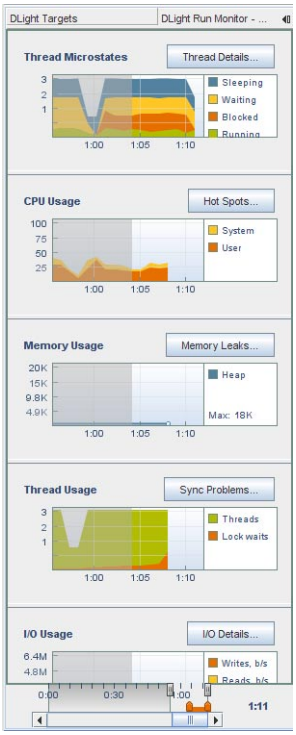
「DLight 実行モニター」ウインドウの下部には、グラフの表示を制御するための表示スライダ、詳細スライダ、および時間スライダの各スライダがあります。

1. 各スライダに関するツールチップ情報を表示するには、マウスカーソルをスライダの終了ポイントの上に合わせます。
2. マウスカーソルで時間スライダをクリックし、マウスボタンを押したままスライダを左側にドラッグすると、実行の開始位置を確認できます。すべてのグラフが連動してスライドするため、任意の時間に各領域 (CPU、メモリー、スレッド、I/O) がどのような状況になっているかを確認し、それらの領域の関係を見ることができます。
3. 時間スライダを左から右にドラッグすると、実行の全体を確認できます。

4. 時間単位で重なったコントロール、表示スライダにマウスカーソルを移動します。表示スライダコントロールを使用すると、実行時間のどの部分をグラフに表示するかを選択できます。
5. 表示スライダの開始ポイントである左ハンドルをクリックし、実行の開始位置 0:00 までドラッグします。これで実行全体がグラフに一度に表示されるようになりました。可能なところまで縮小表示する場合も、同様の結果になります。実行時間全体を選択した場合、時間スライダは機能しません。データ全体をすでに表示しているため、スクロールする部分がありません。



6. 表示スライダはズームに使用できます。表示スライダの開始ポイントを右にドラッグします。ハンドルをドラッグすると、グラフは実行の領域から終了ポイントに向かって拡大していきます。時間スライダは、ふたたび実行時を前後にスクロールできるようになっています。
7. スライダの使用方法の説明を表示するために、オレンジ色の詳細スライダの終了ポイントの上にマウスカーソルを合わせます。詳細スライダコントロールを使用すると、実行時の一部を選択して詳細を調べることができます。
8. 詳細スライダの開始ポイントを、表示スライダの開始ポイントよりも後の時間にドラッグします。グラフは、開始ポイントの前の領域でグレー表示になり、それによって開始ポイントと終了ポイントの間の領域が強調表示されます。



9. グラフの詳細ボタン (「スレッドの詳細 (Thread Details)」、「ホットスポット (Hot Spots)」、「メモリーリーク (Memory Leak)」、「同期の問題 (Sync Problems)」、または「I/O の詳細 (I/O Details)」) のいずれかをクリックすると、強調表示された領域のデータが詳細タブに表示されます。言い換えると、詳細スライダは、詳細タブに表示するデータの選択に使用します。
10. 表示スライダの開始ポイントをドラッグして実行の開始位置まで戻し、すべてのデータを表示します。

スレッドマイクロステートの調査

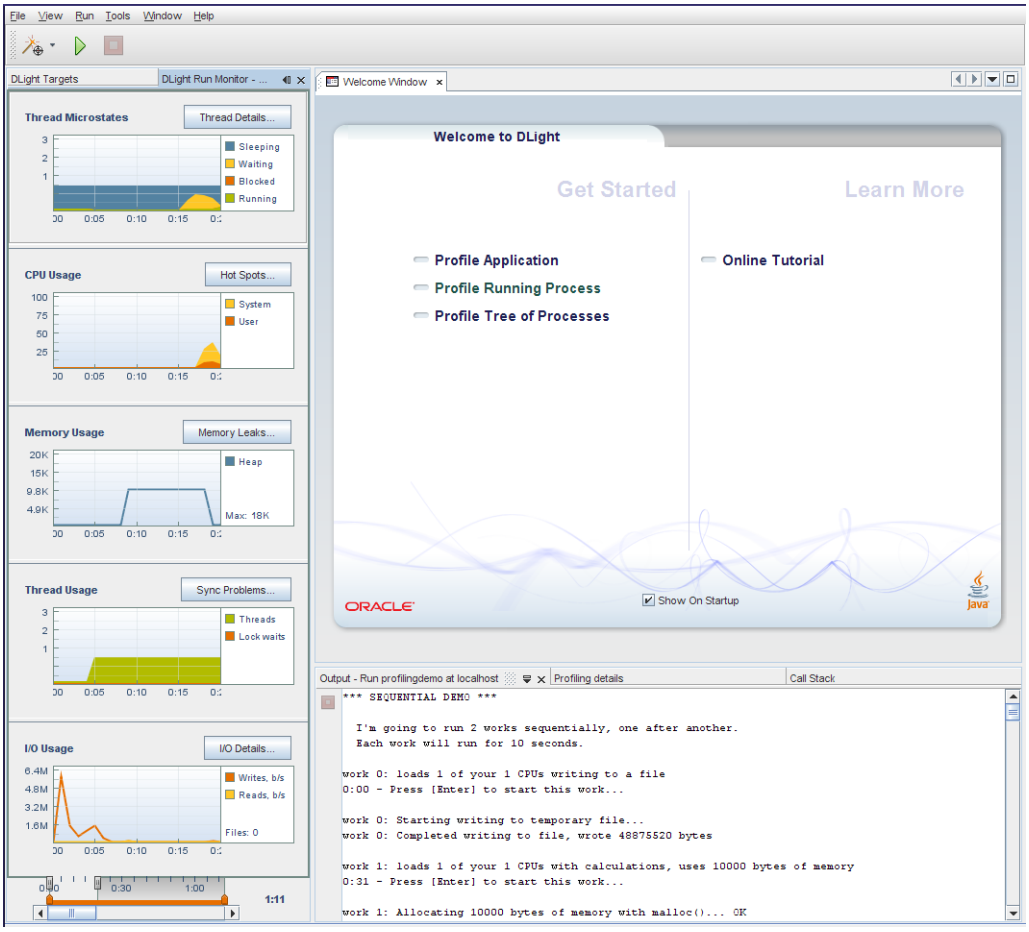
「スレッドマイクロステート (Thread Microstate)」グラフには、プログラムの実行中の実行状態の変化など、プログラムのスレッドの概要が表示されます。Solaris マイクロステートアカウンティング機能は、DTrace 機能を使用して、10 の異なる実行状態に入ったりその状態から出たりする各スレッドの状態について、詳細な情報を提供します。

ユーザー実行中	ユーザーモードでプロセスが費やした時間の割合
システム実行中	システムモードでプロセスが費やした時間の割合
ほかの実行中	システムトラップなどの処理でプロセスが費やした時間の割合
テキストページフォルト	テキストページフォルトの処理でプロセスが費やした時間の割合
データページフォルト	データページフォルトの処理でプロセスが費やした時間の割合
ブロック中	ユーザーロックの待機にプロセスが費やした時間の割合
スリープ中	休眠でプロセスが費やした時間の割合
待機中	CPU の待機にプロセスが費やした時間の割合

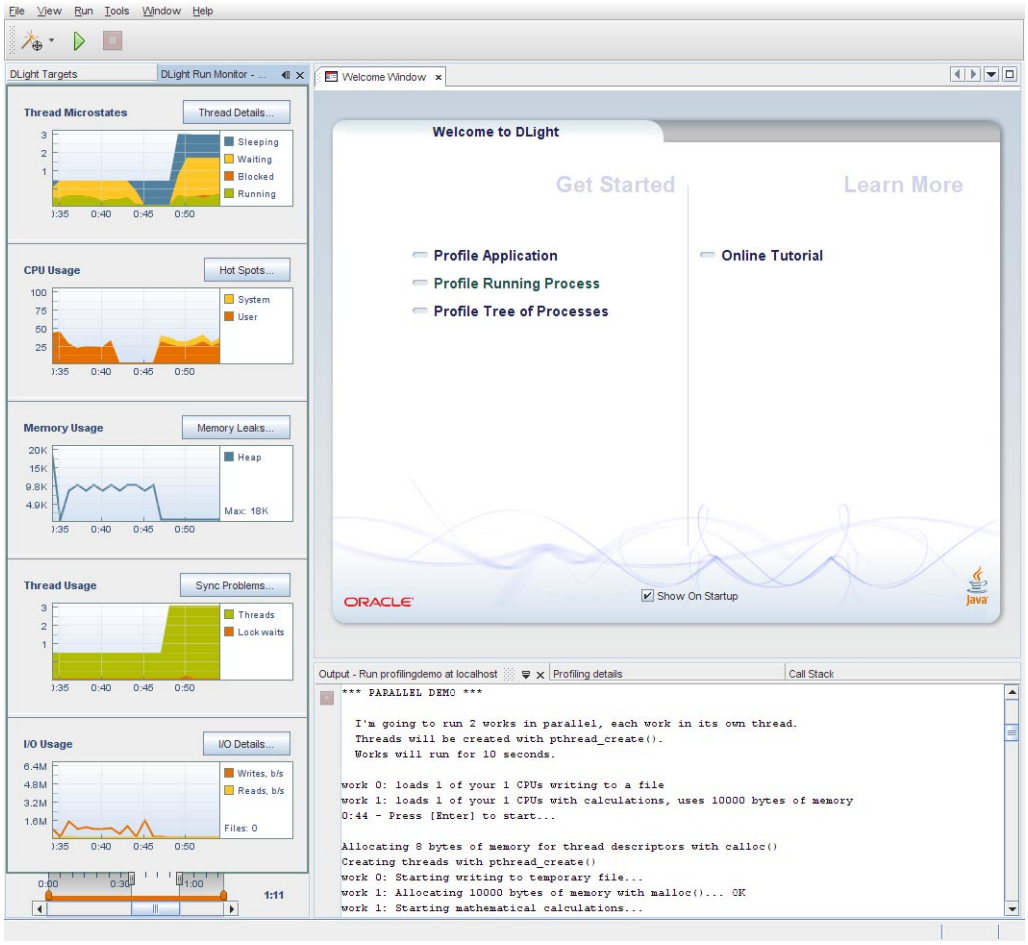
スレッドマイクロステートツールは、プログラムの実行中に作成されたすべてのスレッドの状態の概要情報をグラフィカルに表示します。表示される状態は、「休眠中 (Sleeping)」、「待機中 (Waiting)」、「ブロック (Blocked)」、「実行中 (Running)」の 4 つだけです。これら 4 つの状態は、可能性のある 10 のマイクロステートを簡略化または概略的に表したもので、プログラムで実行中のすべてのスレッドの状態の概要を示します。たとえば、「実行中 (Running)」状態で使

用される時間は、ユーザーモードで実行中、システムコールで実行中、ページフォルトで実行中、トラップで実行中と、すべての種類の実行状態を表します。

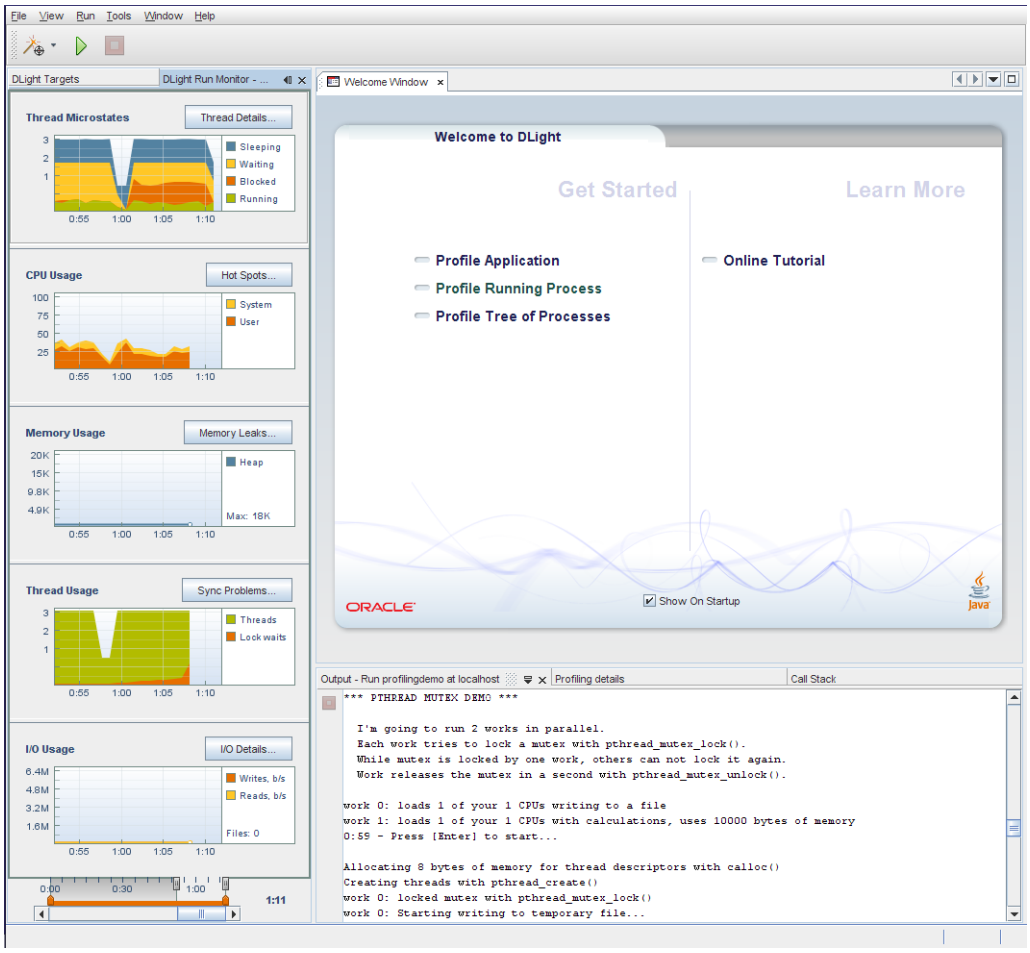
1. 表示スライダの左ハンドルを左に移動して、次の図のようにグラフが実行時の約 20 秒間を示すようにします。次のイメージではプログラムの実行開始位置、SEQUENTIAL DEMO 部分の期間が表示されています。この部分ではシングルスレッドで 2 つのタスクが交互に実行されます。スレッドが休眠中のポイントは、ユーザーが Enter キーをクリックするのをプログラムが待機している時点に相当します。



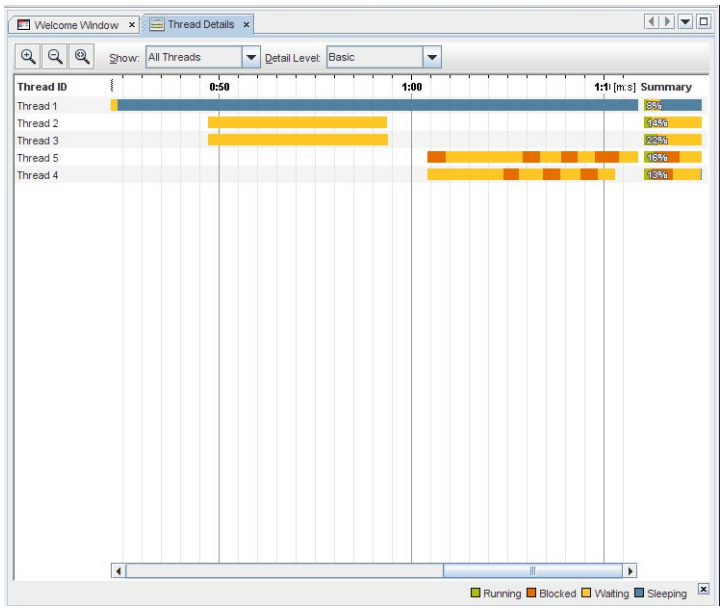
2. 時間スライダをクリックして右にドラッグし、スレッドマイクロステートツールに表示されるスレッド数が 3 に増加するところまで移動します。
3. プログラムが PARALLEL DEMO 部分に入るとスレッドの数は 3 つになっています。メインスレッドから 2 つの追加スレッドが起動し、それぞれのスレッドにある 2 つのタスクを並行して実行します。「待機中 (Waiting)」状態 (黄色) と「休眠中 (Sleeping)」状態 (青色) でかなりの時間が費やされ、「実行中 (Running)」状態 (緑色)にはそれほど時間が使われていないのがわかります。PARALLEL DEMO 部分に、「ブロック中」状態 (オレンジ色) に使用された時間がないのは、プログラムのこの部分に、スレッドをブロックする相互排他ロックなどのスレッド同期手段が実装されていないためです。



4. 時間スライダを実行時の終了ポイントまでドラッグします。オレンジ色で示される「ブロック (Blocked)」状態のマイクロステートは、プログラムが `PTHREAD MUTEX DEMO` の部分に入る時点に現れます。ここで、各スレッドは相互排他ロックを使用し、特定の時点でほかのスレッドから干渉されることを回避します。各スレッドは、相互排他ロックを取得した後にのみアクティブに実行できます。別のスレッドが相互排他ロックを所有しているときに、スレッドがコードのロックされた部分にアクセスを試みると、スレッドはブロックされます。相互排他ロックを使用すると、同じデータへのアクセスが重なってスレッドがデータの競合状態になることを防ぎます。

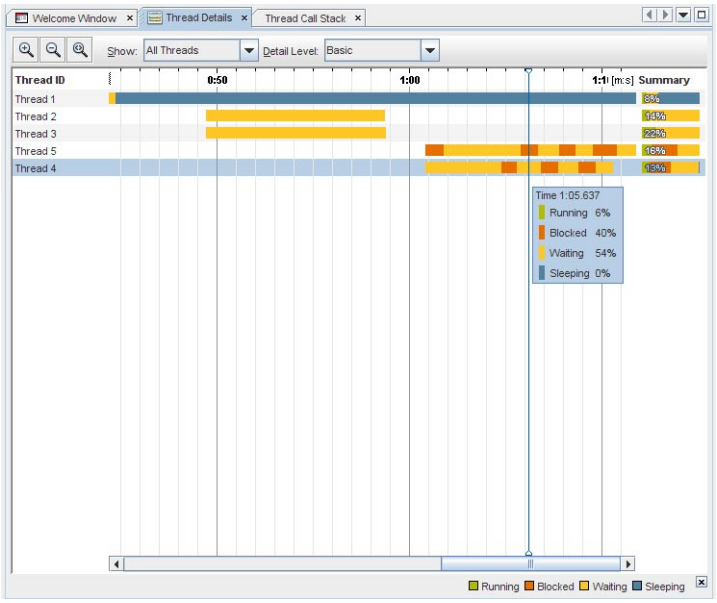


5. 「スレッドの詳細 (Thread Details)」ボタンをクリックすると、スレッドマイクロステートに関する詳細が表示されます。「スレッドの詳細 (Thread Details)」が開き、状態の詳細情報とともに、プログラムで実行されたすべてのスレッドがタイムラインでグラフィカルに表示されます。

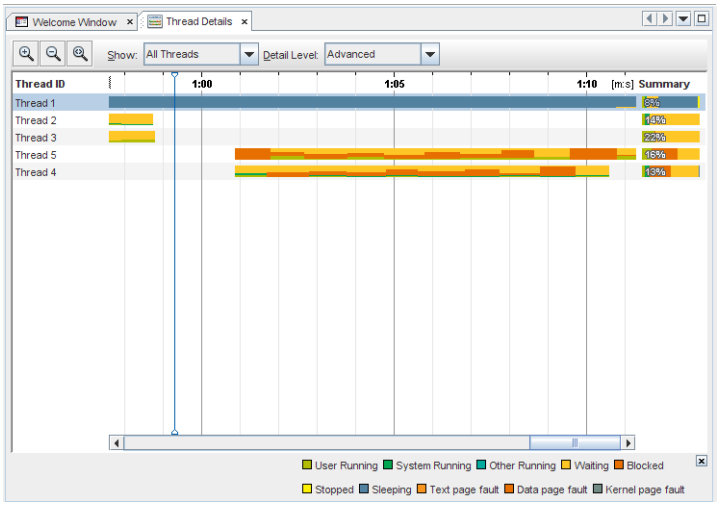


「スレッドの詳細 (Thread Details)」ウィンドウには、プログラムの実行時間全体での各スレッドに関する状態の推移が表示されます。

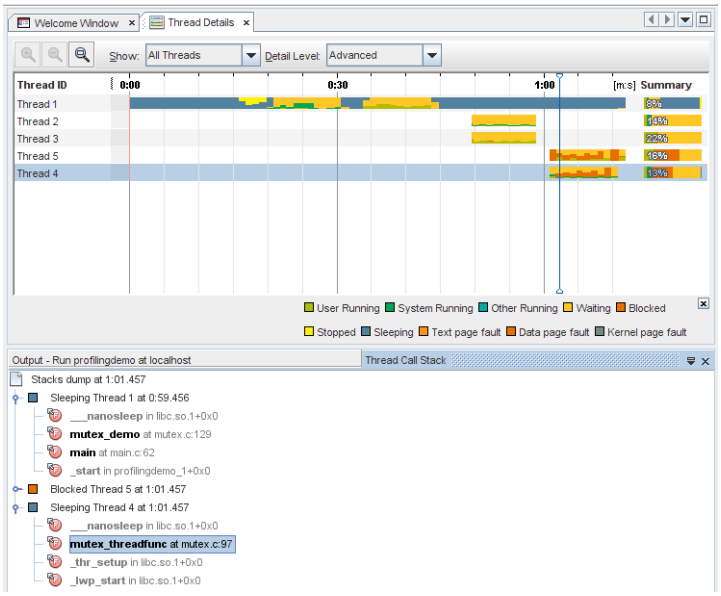
6. カーソルを、スレッドの色が付いた領域のいずれかに置きます。その特定の時点でそのスレッドで行われていることについての詳細を示すポップアップウィンドウが表示されます。詳細には、データが収集された時間と、その時点で各スレッド状態で費やされた時間の割合が含まれます。カーソルをウィンドウ右側にある「概要 (Summary)」領域に置くと、そのスレッドの実行全体における各状態の割合が、ポップアップウィンドウに表示されます。



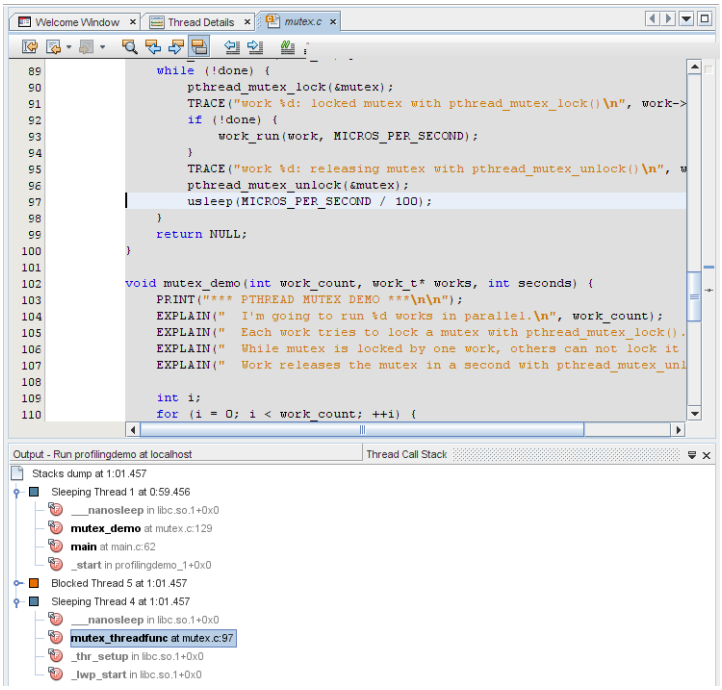
7. 表示内容を変更するには、ウィンドウのコントロールを使用してみてください。
   - デフォルトではウィンドウにはすべてのスレッドが表示されます。「表示 (Show)」ドロップダウンリストの右側にある下矢印をクリックします。終了していないスレッドのみを表示する「ライブスレッドのみ (Live Threads only)」を選択するか、プログラム実行中に終了したスレッドのみを表示する「終了スレッドのみ (Finished Threads only)」を選択できます。
   - デフォルトでは、ウィンドウには 4 つの汎用実行状態のみが表示されます。「詳細レベル (Detail Level)」ドロップダウンリストの右側にある下矢印をクリックします。これらの状態がより詳細に表示される「中 (Moderate)」を選択するか、10 のマイクロステートが表示される「詳細 (Advanced)」を選択できます。
   - 個々のスレッドをクリックし、そのスレッドが強調表示されることを確認します。Shift キーを押しながら別のスレッドをクリックすると、その範囲のスレッドを選択できます。隣接していないスレッドを複数選択するには、Ctrl キーを押しながらスレッドを選択します。目的のスレッドが強調表示されたら、右クリックして「選択されたスレッドのみ表示 (Show Only Selected Threads)」を選択します。もう一度すべてのスレッドを表示するには、「表示 (Show)」ドロップダウンリストで「すべてのスレッド (All Threads)」を選択します。
   - いずれかのスレッドを右クリックし、「スレッド名」⇒「スレッドエントリ」を選択し、表示されているスレッド名を、スレッドが実行を開始したときに入った関数に変更します。
8. もっと詳しく見るには、拡大ボタン  . このイメージには、「詳細レベル (Detail Level)」が「詳細 (Advanced)」に設定され、拡大されたウィンドウが表示されます。



9. 縮小ボタン をクリックすると、前のズームレベルに戻ります。
10. 「実行全体を表示 (Show Complete Run)」ボタン をクリックすると、「スレッドの詳細 (Thread Details)」ウィンドウに実行全体が一度に表示されますもう一度ボタンをクリックすると ふたたびスレッドの詳細をスクロール表示できます。
11. スレッド 4 上の 2 番目のオレンジ色の長方形をクリックします。「スレッド呼び出しスタック (Thread Call stack)」タブが開き、この時点でのスレッドの呼出スタックが表示されます。スタックのノードを拡張してそのスレッドで発生している呼び出しを確認できます。また、先頭のノードを右クリックして「すべて展開 (Expand All)」を選択すると、すべてのスレッドの呼び出しを表示できます。



12. `mutex_threadfunc` 関数をダブルクリックすると、関数が呼び出された場所のソースファイルが開きます(グレー表示されていないスタックのどの関数についても、呼び出しソースファイルを表示できます)。

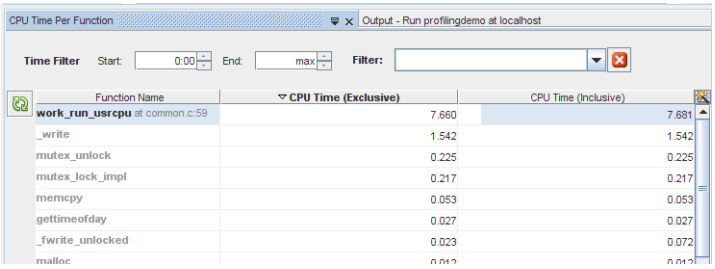


13. 「スレッドの詳細 (Thread Details)」タブをクリックすると、「スレッドの詳細 (Thread Details)」ウィンドウに戻ります。スレッドをクリックします。マウスまたはキーボードを使用して、スレッドのタイムラインに沿って移動できます。
    - マウスを使用する場合は、スレッドを右クリックし、「ナビゲート (Navigate)」メニューを使用してフォーカスを左または右に移動し、タイムライン上のポイントを設定し、「スレッド呼び出しスタック (Thread Call stack)」タブの内容を更新するか、「スレッド呼び出しスタック (Thread Call stack)」タブにフォーカスを切り替えます。
    - キーボードショートカットを使用してスレッドを移動するには、次のキーを使用します。
        - Ctrl+左矢印および Ctrl+右矢印で、スレッドのタイムラインを左右にスクロールします。
        - Ctrl+下矢印で、タイムライン上のポイントを選択すると、「スレッド呼び出しスタック (Thread Call stack)」が更新されます。
        - Alt+下矢印で、「スレッド呼び出しスタック (Thread Call stack)」ウインドウの入力値をフォーカスします。
    - 「スレッド呼び出しスタック (Thread Call stack)」で、矢印キーと Enter キーを使用すると、関数に関連付けられたソースファイルが表示されます。

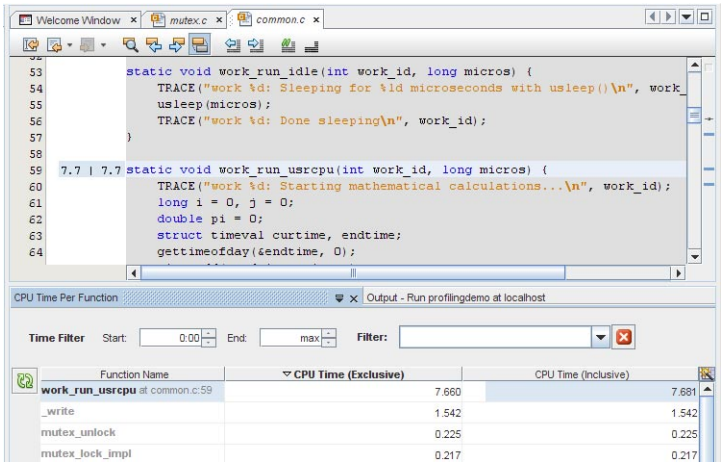
CPU 使用状況の調査

「CPU 使用 (CPU Usage)」グラフには、アプリケーションの実行中に使用される合計 CPU 時間がパーセントで表示されます。

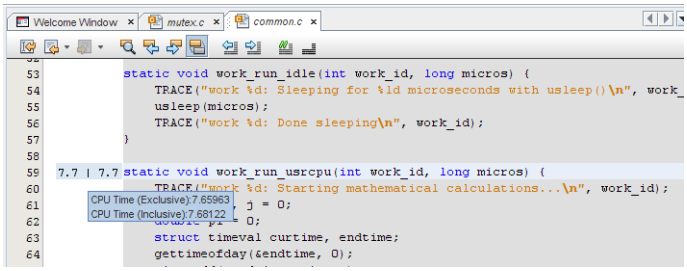
1. 「ホットスポット (Hot Spots)」ボタンをクリックすると、CPU 時間に関する詳細が表示されます。「関数あたりの CPU 時間 (CPU Time Per Function)」タブが開き、プログラムの関数が、各関数によって使用された CPU の時間とともに表示されます。関数は使用された CPU 時間の順に一覧表示されるため、使用時間がもっとも長い関数が最初に表示されます。プログラムがまだ実行中の場合は、初期状態で表示される時間は、ボタンをクリックした時点で費やした時間です。



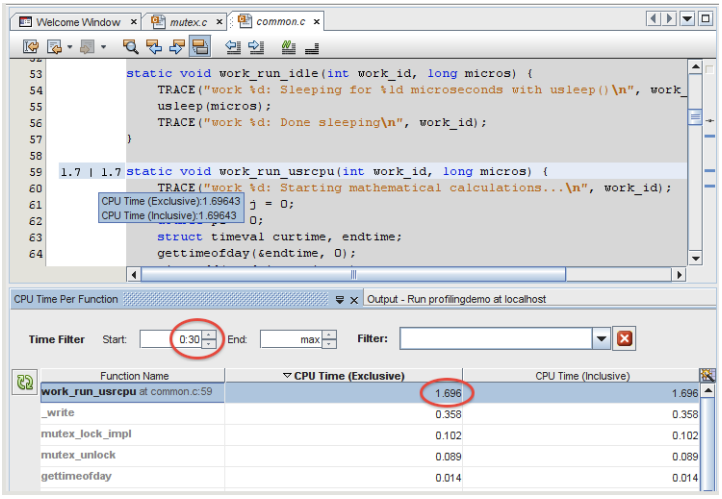
2. 「関数名 (Function Name)」列のヘッダーをクリックすると、関数がアルファベット順にソートされます。
3. 「CPU 時間 (排他的) (CPU Time (Exclusive))」列のヘッダーをクリックすると、個々の関数によって使用された時間の順に関数がソートされます。
4. CPU 時間の 2 つの列の違いを確認します。「CPU 時間 (包括的) (CPU Time (Inclusive))」には、関数の実行開始から終了までの間に使用された CPU 時間の合計が表示されます。また、一覧表示された関数によって呼び出されたその他すべての関数の時間も含まれます。「CPU 時間 (排他的) (CPU Time (Exclusive))」には、特定の関数のみに使用された時間が表示され、その関数から呼び出された関数は含まれません。
5. 「CPU 時間 (包括的) (CPU Time (Inclusive))」列ヘッダーをクリックして、使用時間がもっとも長い関数を先頭に戻します。`work_run_usrcpu` 関数は「CPU 時間 (排他的) (CPU Time (Exclusive))」よりも「CPU 時間 (包括的) (CPU Time (Inclusive))」の方をわずかに長く使用していることを確認します。これは CPU 時間のうちの少量が、実際はこの関数が呼び出した他の関数によって使用されていることを意味します。ただしほとんどの時間を使用しているのは `work run_usrcpu` 関数自体です。
6. 一部の関数はボールドテキストで表示されています。これらの関数のソースファイルを表示できます。`work_run_usrcpu` 関数をダブルクリックします。`common.c` ファイルが開き、`work run_usrcpu` 関数がある 59 行目にカーソルが置かれています。



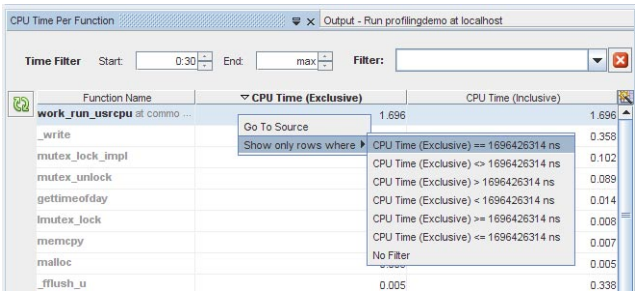
7. 左マージンにある数字の上にマウスカーソルを合わせます。これらの数字は、「関数あたりの CPU 時間 (CPU Time Per Function)」タブに表示される関数の包括的および排他的 CPU 時間と同じメトリックです。表示領域を少なくするためにメトリックは丸めて表示されますが、マウスをその上に合わせると、丸められていない値が表示されます。`work_run_usrcpu` 関数内で計算を行う `for` ループなど、CPU を消費する行のメトリックは、`common.c` ソースファイルにも表示されます。



8. 時間を入力して Enter キーを押すか、矢印を使用して秒をスクロールして、「関数あたりの CPU 時間 (CPU Time Per Function)」タブの「時間フィルタ (Time Filter)」の開始時間を 0:30 に変更します。「実行モニター」ウィンドウのグラフが、詳細スライダのハンドルを移動したときの状態に変更されます。ハンドルをドラッグすると、「関数あたりの CPU 時間 (CPU Time Per Function)」タブの「時間フィルタ (Time Filter)」の設定が一致するように更新されます。重要なのは、このタブの関数に対して表示されるデータがフィルタを反映して更新されるため、その時間中に使用された CPU 時間のみが表示されることです。



9. 特定のメトリックに一致するデータにフィルタを適用することもできます。`work_run_usrcpu` の「CPU 時間 (排他的) (CPU Time (Exclusive))」メトリックを右クリックします。「次の条件の行のみ表示(Show only rows where)」 > 「CPU 時間 (排他的) (CPU Time (Exclusive)) == `work_run_usrcpu` に示されているメトリック」の順に選択します。他の行がすべてフィルタで除外され、排他的 CPU 時間がこのメトリックと等しい行のみが表示されます。

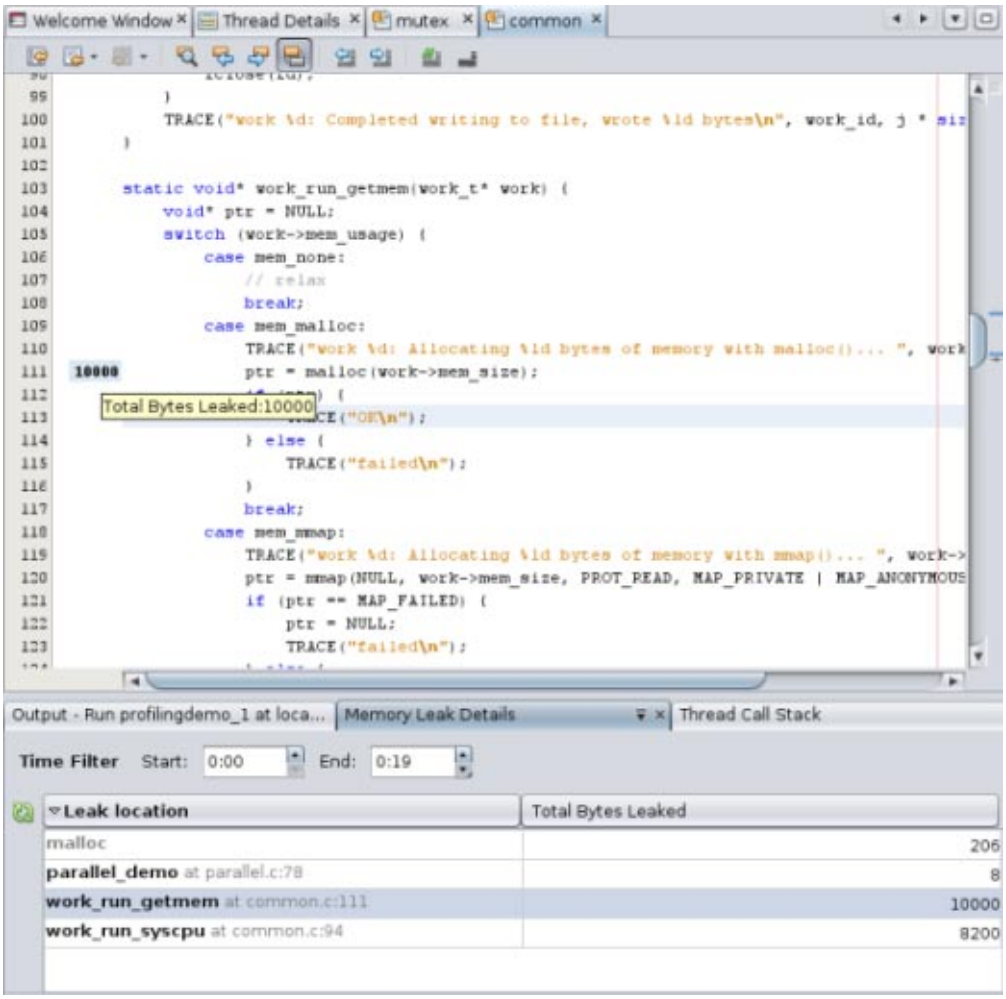


メモリー使用状況の調査

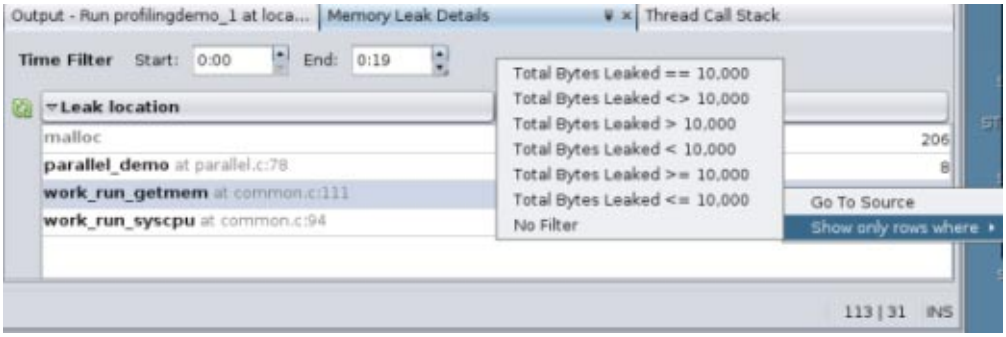
「メモリー使用 (Memory Usage)」ツールには、プログラム実行時のメモリーヒープの経時変化が表示されます。これは、プログラム内で不要になったメモリーの開放に失敗した場所をポイントする、メモリーリークの特定に使用できます。メモリーリークは、プログラムのメモリー消費が

増加する原因となります。メモリーリークが発生しているプログラムの実行時間が長くなると、結果的に使用できるメモリーが不足する場合があります。

1. 「実行モニター」の時間スライダを左右にスライドさせ、時間とともにメモリーヒープが増減する様子を確認します。このプログラムの実行の使用状況には、4 回の波があります。最初の 2 回は、`SEQUENTIAL DEMO` 中に、3 回目は `PARALLEL DEMO` 中に、4 回目は `PTHREAD MUTEX DEMO` 中に発生しています。
2. 「メモリーリーク (Memory Leak)」ボタンをクリックすると、「メモリーリーク詳細 (Memory Leak Detail)」タブが開き、ここに、メモリーリークを示す関数の詳細が表示されます。このタブにはメモリーリークが発生している関数のみが一覧表示されます。ボタンをクリックした時にプログラムが実行中の場合は、ボタンをクリックした時点で存在していたリークの場所が表示されます。時間が経過すると、メモリーリークが増加する可能性があるため、「再表示 (Refresh)」ボタン  をクリックしてリストを更新してください。実行の終了時までメモリーリークが検出されなかった場合は、「メモリーリーク詳細 (Memory Leak Detail)」タブにメモリーリークが見つからなかったことが示されます。
3. 「メモリーリークの詳細」タブで「開始」時間および「終了」時間を変更するか、「実行モニター」ウィンドウで詳細スライダを使用して、データをフィルタできます。
4. この実行では、`ProfilingDemo` プログラムは `work` `run_getmem` 関数に関連付けられたメモリーリークを示しています。`work_run_getmem` 関数をダブルクリックすると、`common.c` ファイルが開き、この関数でメモリーリークが発生している行にカーソルが置かれます。
5. メモリーリークメトリックが左マージンに表示されます。CPU 使用状況のメトリックで実行したとこと同様に、マウスをそれらの上にあわせると詳細が表示されます。



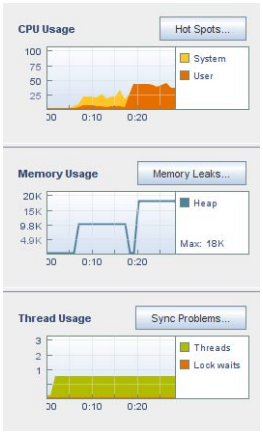
6. 「関数あたりの CPU 時間 (CPU Time Per Function)」タブと同じように、「メモリーリーク詳細 (Memory Leak Detail)」タブでメトリックを右クリックしてデータをフィルタする基準を選択します。



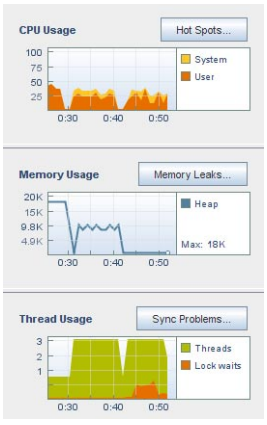
スレッド使用状況の調査

「スレッド使用 (Thread Usage)」ツールには、プログラムで使用されているスレッド数と、スレッドがタスクを続行するためにロックを取得するまで待機しなければならないすべての時点が示されます。このデータはマルチスレッドのアプリケーションで、コストのかかる待ち時間を回避するためにスレッドの同期を行う必要がある場合に有用です。

1. 時間スライダを実行の開始位置にスライドさせ、「スレッドマイクロステート (Thread Microstate)」グラフの場合と同じように、プログラムの `SEQUENTIAL DEMO` 部分が、プログラムが `PARA LEL DEMO` 部分に入ってスレッド数が 3 になるまでの間、スレッド数が 1 であることを確認します。
2. 実行の開始位置から、さらに 2 つのスレッドが開始される直前までのデータが表示されるように、表示スライダの終了ポイントのハンドルを移動させます。
3. 「CPU 使用 (CPU Usage)」グラフと「メモリー使用 (Memory Usage)」グラフで同じ期間を見て、1 つのスレッドが CPU の時間とメモリーを使用する何らかの活動を実行していることを確認します。この期間は、メインスレッドがファイルに書き込みを行い、いくつかの計算を連続して行う、プログラムの `SEQUENTIAL DEMO` 部分に対応します。ユーザーが Enter キーを押すのをプログラムが待機する間、CPU とメモリーの使用量は減少し、スレッド数は 1 のままです。

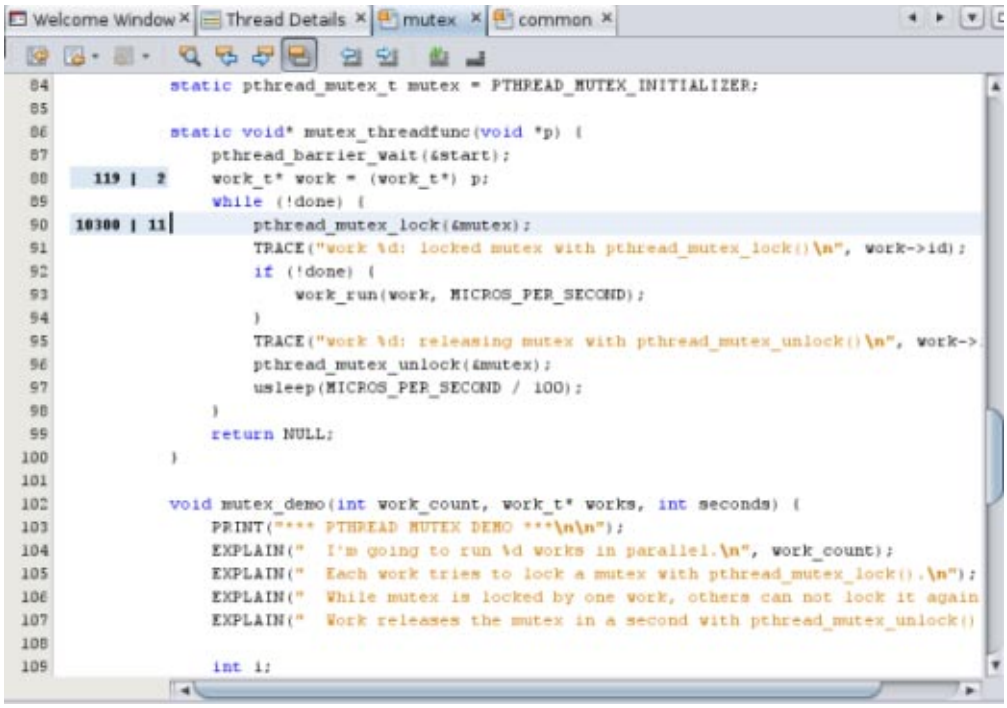


4. 時間スライダを右にスライドさせ、スレッドが 3 に増加する 2 か所のポイントを確認できます。スレッドの 1 回目の増加は、プログラムの実行の `PARALLEL DEMO` 部分に対応しえおり、メインスレッドがファイルへの書き込みと計算の実行の作業を並行して行うために、2 つの追加のスレッドを開始します。この部分でメモリー使用量と CPU 使用量が少し増加しますが、`SEQUENTIAL DEMO` 部分の場合よりもずっと短い時間で 2 つのタスクが完了します。



5. `PARALLEL DEMO` スレッドが終了した後で、スレッド数が 1に戻り、メインスレッドはユーザーが Enter キーを押すのを待機します。

6. プログラムの PTHREAD MUTEX DEMO 部分の実行時に、スレッド数はふたたび 3 に増加します。スレッド数が 3 に増加してまもなく、オレンジ色で示されるロック待機がオレンジ色で表示されます。PTHREAD MUTEX DEMO では、複数のスレッドによる特定の関数へのアクセス競合を防ぐために相互排他ロックを使用しますが、これがスレッドがロックの取得を待機する原因です。
7. 「同期の問題 (Sync Problems)」ボタンをクリックし、スレッドロックの詳細を表示します。「スレッド同期の詳細 (Thread Synchronization Details)」タブが開き、相互排他ロックを取得するために待機する必要がある関数が一覧表示されます。また、関数が待機に費やしたミリ秒数のメトリックと、関数がロックを待機する必要があった回数が表示されます。
8. プログラムの実行中に「同期の問題 (Sync Problems)」ボタンをクリックするときは、「再表示 (Refresh)」ボタン をクリックして、最新のスレッドロックで表示を更新する必要がある場合があります。
9. 「待ち時間 (Wait Time)」列のヘッダーをクリックすると、待機に費やした時間の順に関数がソートされます。
10. 「ロック待ち」列のヘッダーをクリックすると、関数でスレッドが待機していた回数の順に関数がソートされます。
11. ロック待機回数が最も多い mutex_threadfunc 関数をダブルクリックします。mutex.c ソースファイルが開き、pthread_mutex_lock 関数が呼び出された行にカーソルが置かれています。この関数は、メモリーの場所が読み取りまたは書き込みされる前にメモリーの場所をロックする役割を担い、そのメモリーにロックを持つスレッドがなくなるまで待機します。

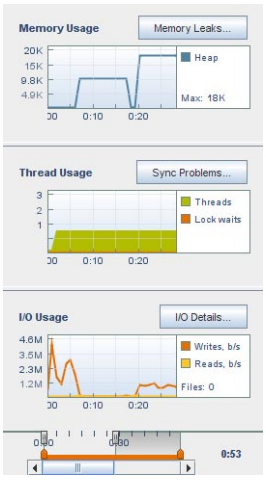


12. 待ち時間およびロック待機数のメトリックは、ソースファイルの左側の列の余白に表示されます。メトリックの上にマウスカーソルを合わせると詳細が表示されます。この詳細は「スレッド同期の詳細 (Thread Synchronization Details)」タブの表示内容と一致します。
13. メトリックを右クリックして「プロファイラメトリックを表示 (Show Profiler Metrics)」を選択解除します。これでメトリックはどのプロファイルツールのソースエディタにも表示されなくなります。
14. 「表示」⇒「プロファイラメトリックを表示」の順に選択し、ふたたびメトリックを表示します。

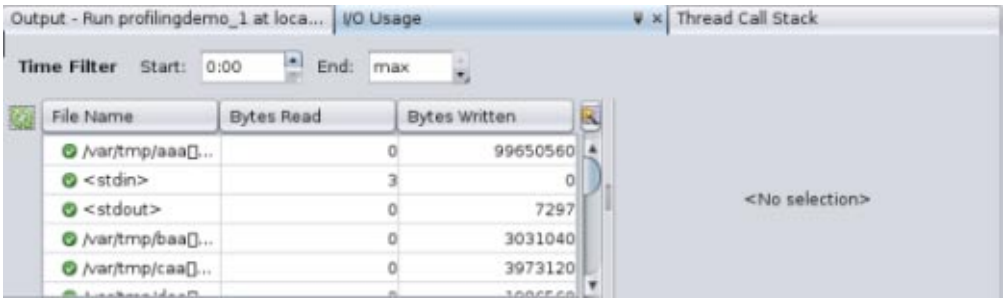
I/O 使用状況の調査

「I/O 使用 (I/O Usage)」ツールには、プログラム実行中の読み取りおよび書き込み活動の概要が表示されます。

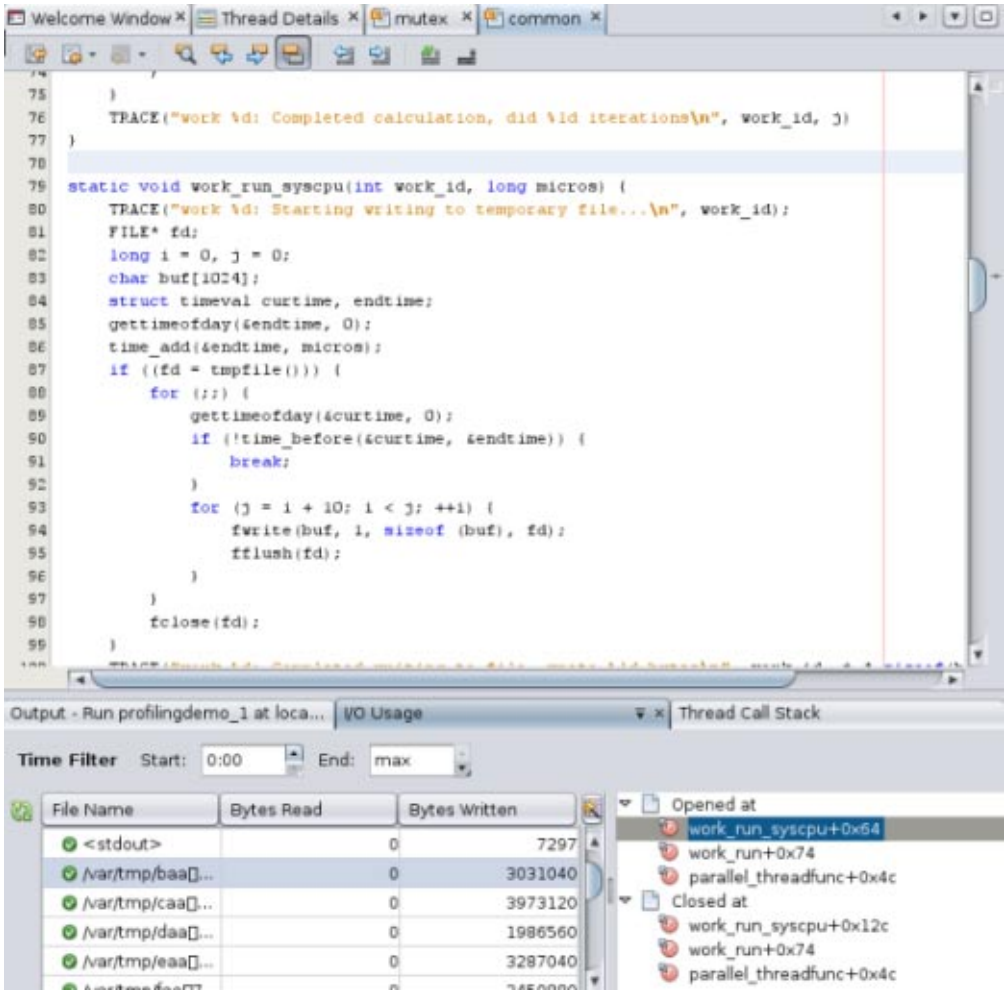
1. 次のイメージには、シングルスレッドで 2 つのタスクが交互に実行される `SEQUENTIAL DEMO` 部分における、実行の開始位置での I/O 使用状況が表示されています。最初の数秒間でプログラムが開始し、その後ユーザーが Enter キーを押すまで待機していました。ユーザーが Enter キーを押すと、プログラムによって一時ファイルに文字が書き込まれました。この活動は、書き込まれたバイト数を示すオレンジ色の線でグラフに反映されます。



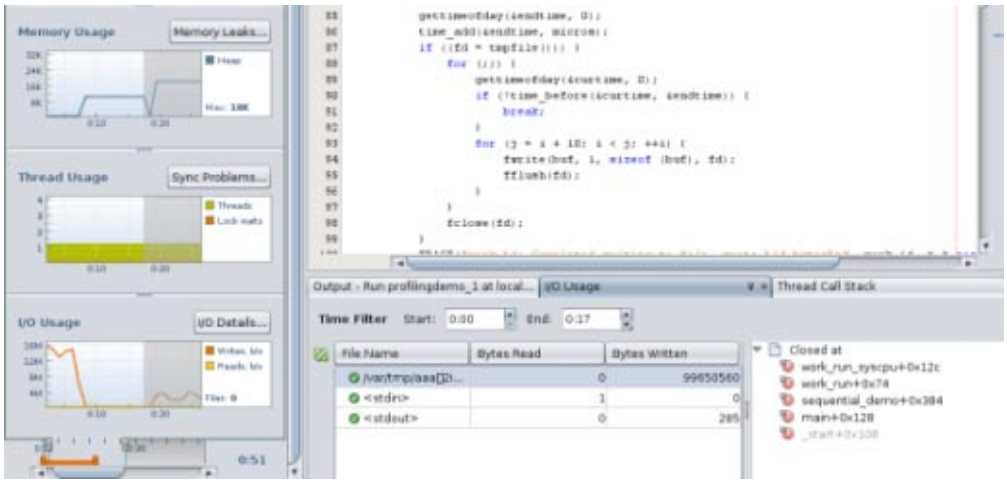
2. 次を確認します。
   - オレンジ色の線は、最近の 1 秒間に書き込まれたバイトの数を示しています。ProfilingDemo プログラム自体は、この `SEQUENTIAL DEMO` の実行中に合計 79984640 バイト、つまり約 76.3M バイトの書き込みを行なったことを「出力」ウィンドウに報告します。オレンジ色の線で示されたデータのポイントをすべて合計すると、その値は 76.3M バイト近くになります。
   - プログラムはシステムコールを利用してデータを生成しディスクに書き込むため、このフェーズの間は「CPU 使用 (CPU Usage)」ツールに示されるシステム時間は大きくなります。
   - 「メモリー使用 (Memory Usage)」ツールには、一定して 8K バイトのメモリーヒープが割り当てられたことが示されます。
3. 「I/O の詳細 (I/O Detail)」ボタンをクリックします。「I/O 使用 (I/O Usage)」の詳細タブが開き、標準入力、標準出力、およびプログラムが読み取り書き込みを行う一時ファイルが表示されます。チェックマークの付いたファイルは閉じられています。黄色いアイコンの付いたファイルは読み取りおよび書き込み操作でまだ開いています。



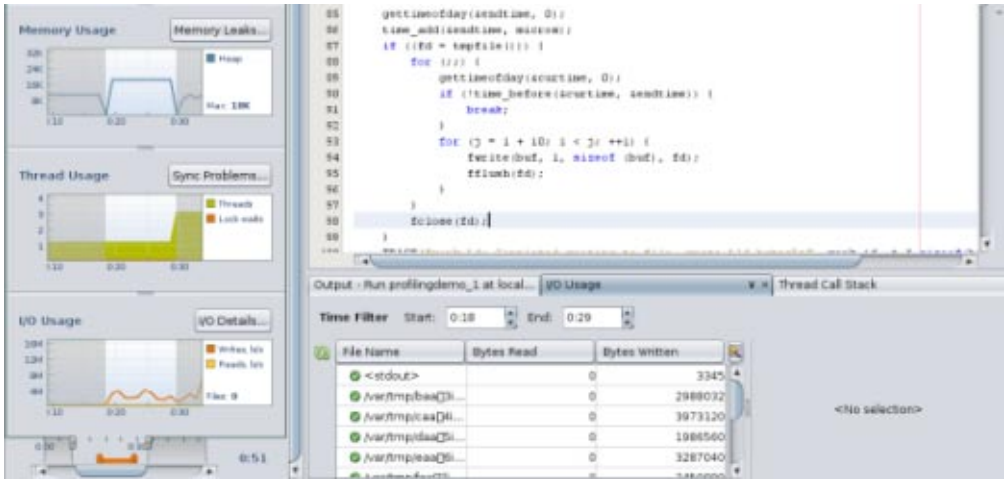
4. このプログラムでは Enter キーで何度か入力するだけであるため、「読み取りバイト数 (Bytes Read)」列はあまり役に立ちません。「読み取りバイト数 (Bytes Read)」列を閉じるには、列ヘッダーの右にある「表示列の変更 (Change Visible Columns)」ボタンをクリックします。「表示列の変更 (Change Visible Columns)」ダイアログボックスで、「読み取りバイト数 (Bytes Read)」を選択解除してから「了解 (OK)」をクリックします。
5. このプログラムでこれらのすべての一時ファイルがどのように使用されているか詳しく知りたいとします。「I/O 使用 (I/O Usage)」の詳細タブで `/var/tmp/baa[]...` ファイルをクリックし、そのファイルがどの関数によって開かれて閉じられたかを確認します。ファイルリストの右側のパネルに、関数が一覧表示されます。
6. この関数リストで `work_run_syscpu` 関数をダブルクリックすると、この関数のソースファイルが開きます。



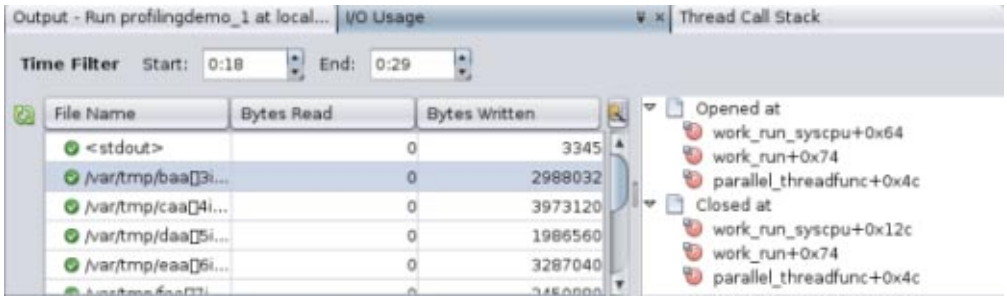
7. 「関数あたりの CPU 時間 (CPU Time Per Function)」タブと「スレッド同期の詳細 (Thread Synchronization Details)」タブで行ったようにように、「I/O 使用 (I/O Usage)」の詳細タブでも表示する期間を指定できます。「実行モニター」ウインドウの「I/O 使用 (I/O Usage)」グラフで、書き込み活動は実行の開始位置に非常に近い時点から始まっています。ここはプログラムの `SEQUENTIAL DEMO` 部分が一時ファイルに書き込みを始めるポイントです。
8. 「終了」フィールドに実行の `SEQUENTIAL DEMO` 部分が終了した時間 (次の図では 0:17) を入力して、この部分を強調表示します。「実行モニター」ウインドウと「I/O 使用」の詳細タブでデータがフィルタされた結果、`SEQUENTIAL DEMO` フェーズの入力と出力がフォーカスされます。



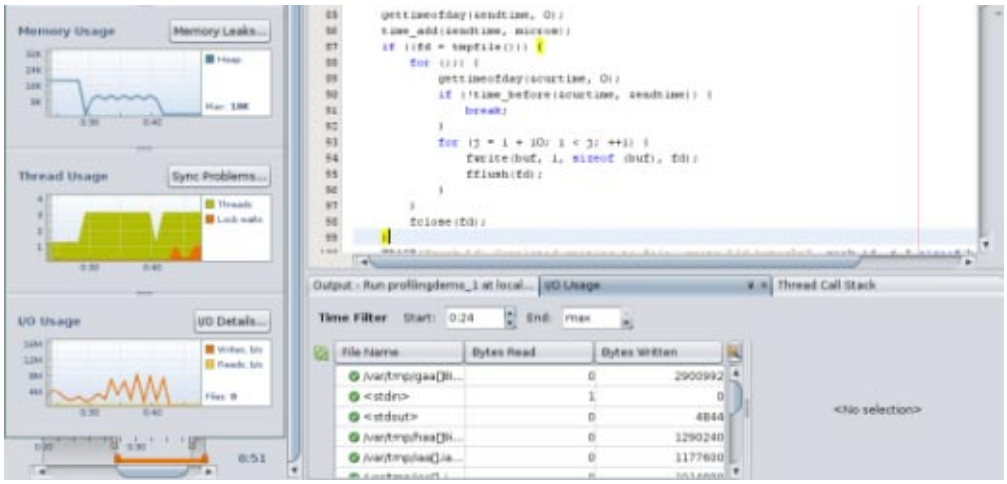
9. SEQUENTIAL DEMO の間、「I/O 使用 (I/O Usage)」の詳細タブには一時ファイルが 1 つ表示されていることを確認します。シングルスレッドが開き、ファイルへの書き込みを行い、ファイルを閉じます。ファイルをクリックしてこのファイルにアクセスした関数を確認し、関数をダブルクリックしてソースコードを表示します。
10. 「実行モニター」ウインドウで、書き込み活動が一時停止した後、ふたたび開始するところまで時間スライダを右に移動させます。書き込みの一時停止は、SEQUENTIAL DEMO の 2 番目のタスクの間に起こります。2 番目のタスクは計算タスクで、この間はディスクへの書き込みは行われません。再開した書き込み活動は、プログラムが `PARALLEL DEMO` 部分に入っていることを示します。この部分ではユーザーが Enter キーを押してタスクを開始し、別々のスレッドでディスクへの書き込みと計算が同時に起こります。
11. 「開始 (Start)」フィールドに PARALLEL DEMO の開始時間を入力し、「終了 (End)」フィールドに終了時間を入力して、PARALLEL DEMO 活動を強調表示します。このイメージに示されている実行では、開始時間は 0:18 で、終了時間は 0:29 です。



12. 実行の `PARALLEL DEMO` 部分の間、「I/O 使用 (I/O Usage)」の詳細タブには複数の一時ファイルが表示されていることを確認します。1 秒ごとに新しいファイルに切り替わるため、書き込みを行っているのは 1 つのスレッドだけです。計算タスクはディスクへの書き込みは行いません。
13. いずれかのファイルをクリックすると、`parallel_threadfunc` 関数がファイルを開いて閉じることが確認できます。



14. 詳細スライダの右ハンドルをクリックして、実行の終了ポイントまでドラッグします。次のイメージでは、プログラムが PTHREAD MUTEX DEMO 部分に入りユーザーが Enter キーを押すと、入出力動作がふたたび変化することを確認できます。開始時間を PARALLEL DEMO の終了時間、つまりこの場合は 0.24 に変更することによって、実行の PTHREAD MUTEX DEMO 部分のデータをフィルタ表示します。



15. 「I/O 使用 (I/O Usage)」の詳細タブには、実行の PTHREAD MUTEX DEMO 部分の間、複数のファイルが開かれていることが示されています。PARALLEL DEMO 部分の場合と同様に、1 つのスレッドがファイルへの書き込みを行い、1 秒ごとに書き込み先のファイルを切り替えます。ただし相互排他ロックが使用されているため、時には、書き込みを行っているスレッドが計算を行うスレッドによってブロックされ、ファイルへの書き込みを連続して行えなくなることがあります。

「ターゲットの実行」に対する「実行モニター」ウィンドウのツールのデモはこれで終了です。これらのツールは「プロセスターゲット」にも適用されます。

プロセスツリーのプロファイリング

「プロセスツリーターゲット」を実行すると、実行中のプロセスに DLight が接続されます。DLight は、指定されたプロセスおよびそのプロセスから生成またはフォークされたすべての子プロセスをプロファイルします。実行中のプロセスとその子プロセスのプロファイリングは、プロセス間だけでなくプロセスのスレッド間で発生する問題の検出に役立ちます。

プロセスが起動された時点からの活動を取り込むには、プログラムの実行に使用するシェルのプロファイリングを開始することで、シェル、プログラムのメインプロセス、およびそのすべての子プロセスの活動を記録できます。起動活動には関心がない場合は、プログラムのすでに実行されているプロセスをターゲットにすることもできます。

この節では、プロセスツリーターゲットの一般的な使用方法を示し、サンプルアプリケーションの使用は必要ありません。

1. ターミナルウィンドウで、次のコマンドを入力してシェルの PID を調べます。

   ```
   % echo $$
   ```

2. DLight で、「新規ターゲット」ボタンの横にある下矢印をクリックし、「プロセスツリーターゲット」を選択します。

   「プロセスツリーターゲット」タブが選択された状態で「新規ターゲットを作成」ダイアログボックスが開きます。タブには、システムで実行されているプロセスの一覧が表示されます。

3. 「プロセスツリーターゲット」タブの「フィルタ」フィールドに、`echo` コマンドから返された PID を入力して、プロセスの一覧でこの PID を見つけます。

   プロセスの一覧が自動的に再表示され、「フィルタ」フィールドに入力した文字列を含んでいるプロセスだけが表示されるはずです。

4. この PID で実行されているプロセスを選択し、「保存」をクリックします。

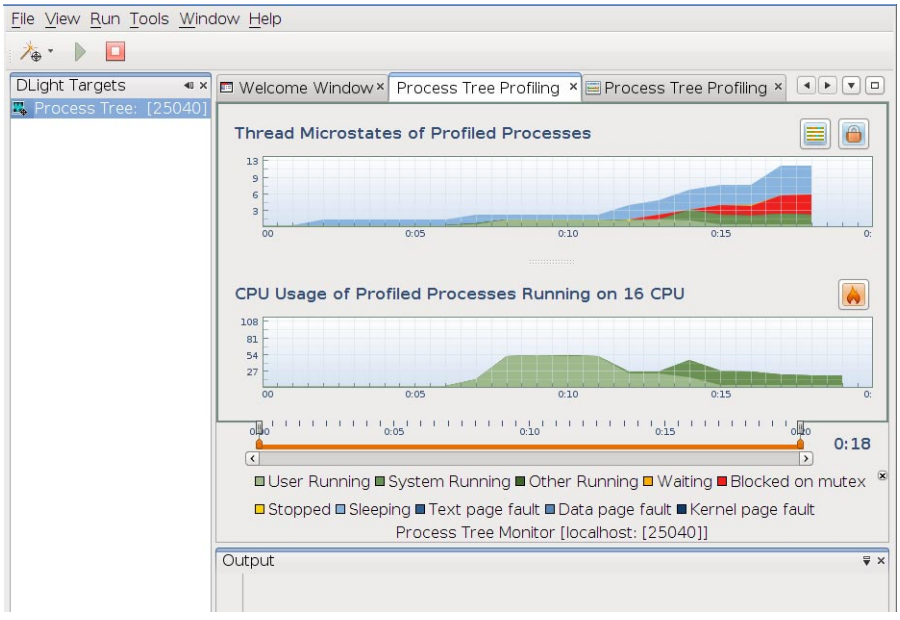
   ターゲットは「DLight ターゲット」タブに「プロセスツリー: [pid] [localhost]」というラベルで表示されます。ターゲットは、この PID で実行されているシェルからどのプログラムを実行する場合でも、この PID で実行されるように恒久的に保存されます。

5. プロセスツリーターゲットを選択し、「実行」ボタンをクリックします  。

   DLight で「プロセスツリーのプロファイリング」タブが開き、「プロファイルされたプロセスのスレッドマイクロステート」および「実行中のプロファイルされたプロセスの CPU 使用率」というラベルのグラフが表示されます。最初は、シェルだけがプロファイルされ、多くの活動は発生していません。

6. ターゲットのシェルが実行されているターミナルウィンドウで、プロファイルするプログラムを起動します。

   次の図は、多くの子プロセスを起動する `locker` というプログラムに関して生成されたデータを示す「プロセスツリーのプロファイリング」タブを示しています。



これらのグラフを併せて使用すると、アプリケーションの複数のプロセスおよび複数のスレッドがどのように連携しているかを判断できます。スレッドがブロックされている箇所と CPU 使用率への影響を調べ、それらが発生しているコード行まで問題を絞り込むことができます。

7. 「プロセスツリーのプロファイリング」タブの上部にある「プロファイルされたプロセスのスレッドマイクロステート」グラフを調べます。

   「プロファイルされたプロセスのスレッドマイクロステート」グラフでは、垂直軸は実行中のスレッドの数を示し、水平軸はプロセスの実行中に経過した時計時間を示します。ただし、このグラフは、DLight がプロセスツリーターゲットでプロファイルした全プロセスのすべてのスレッドのマイクロステートを集約して表示します。このグラフは、プロセスの動作を広域的に確認するために使用できます。

8. 「プロファイルされたプロセスの CPU 使用率」グラフを調べます。

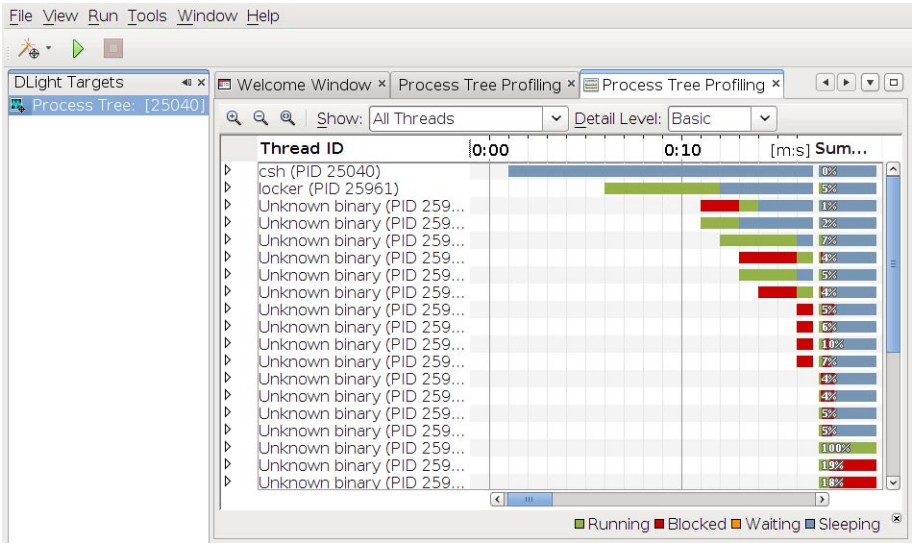
   「プロファイルされたプロセスの CPU 使用率」グラフでは、垂直軸は CPU 時間のパーセントを示し、水平軸はプロセスの実行中に経過した時計時間を示します。ただし、表示されるデータは、システム上のすべての CPU にわたってプロファイルされた全プロセスのすべてのスレッドの CPU 使用率を集約したものです。CPU の数は、グラフのタイトルのほか、DLight ウィンドウの左側の「ホスト情報」ウィンドウにも表示されます。

   スレッドマイクロステートグラフとともに CPU 使用率グラフを使用すると、ロックの問題がある場所を特定できます。

9. 「スレッドマップ」ボタン  をクリックして、プロセスとスレッドのマイクロステートに関する詳細情報を示す新しい「プロセスツリーのプロファイリング」タブを開きます。

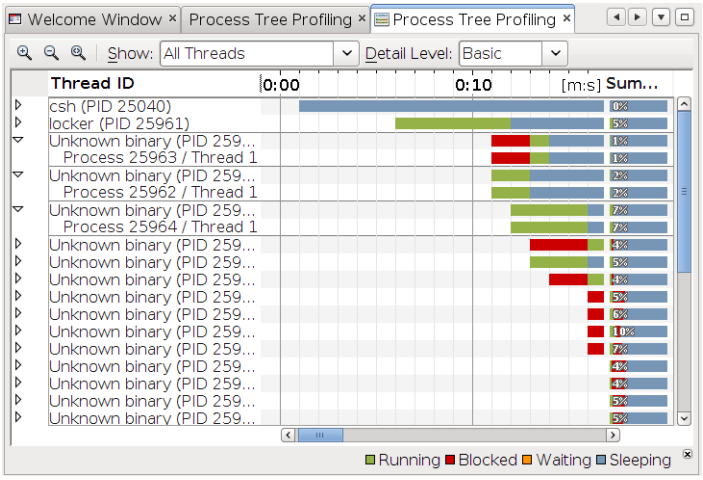
   「プロセスツリーマイクロステートの詳細」ウィンドウは、DLight がターゲットを監視している期間中、ターゲットプロセスとその子プロセスの各スレッドについて、マイクロステートの遷移をタイムライン形式で表示します。

   最初にウィンドウが開いたときは、プロファイルされた各プロセスが表示されます。ターゲットプロセスは一番上の行で、そのプロセスで実行されている全スレッドのマイクロステートの集約が表示されます。子プロセスのタイムラインも、実行するスレッドのマイクロステートを表示します。



   上の図では、DLight ターゲットで指定された `csh` シェルプロセスが一番上の行に表示されており、プログラムの実行中は休眠しています。2 番目の行は `locker` プログラムのメインプロセスで、残りの「不明なバイナリ」プロセスは `locker` プログラムによって元のプロセスのフォークとして起動されたものです。これは、DLight がプロセス名を判定できないことがあるという既知の問題です。

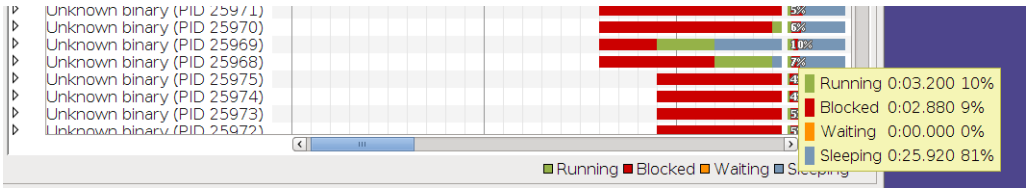
10. プロセスのタイムラインのハンドルをクリックして、そのプロセスによって作成された各スレッドのタイムラインを表示します。



このマニュアルで示されているようにプロセスがシングルスレッドの場合は、プロセスのタイムラインとスレッドのタイムラインは同じに見えます。ただし、後述のとおり、スレッドのタイムラインをクリックすると詳細な情報が表示されます。プロセスがマルチスレッドの場合は、プロセスを展開すると複数のスレッドが表示されます。

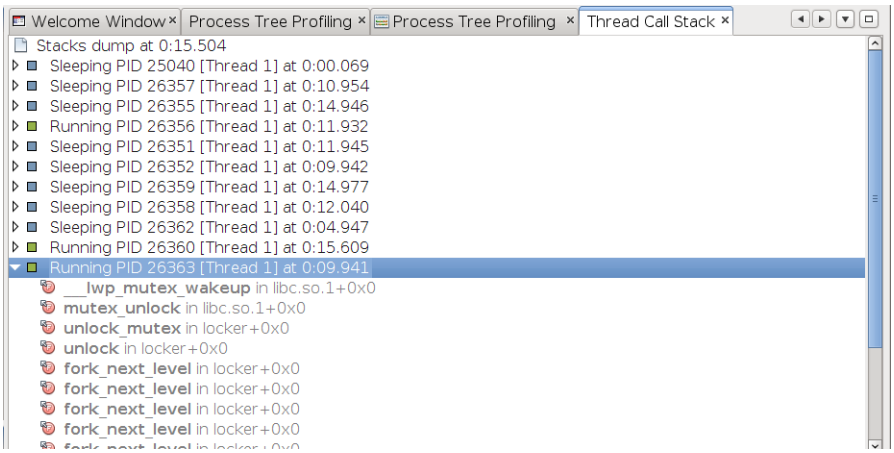
11. 右側にある「概要」領域の上にマウスカーソルを置きます。

    「概要」領域には、スレッドまたはプロセスの実行全体にわたってマイクロステートで費やされた時間の割合が表示されます。「概要」の上にマウスカーソルを置くと、色の意味を示す凡例と、プロセスまたはスレッドの時間の割合が表示されます。プロセスおよびスレッドのタイムライン自体の上にマウスカーソルを置くと、同様のポップアップが表示されます。



上の図では、マウスカーソルはスレッドの「概要」領域に置かれています。

12. プロセスのタイムラインのハンドルをクリックしてそのスレッドのタイムラインを表示し、スレッドのタイムライン内のいずれかの点をクリックすると、「スレッド呼び出しスタック」ウィンドウが開き、その時点で実行されていた呼び出しスタックが表示されます。

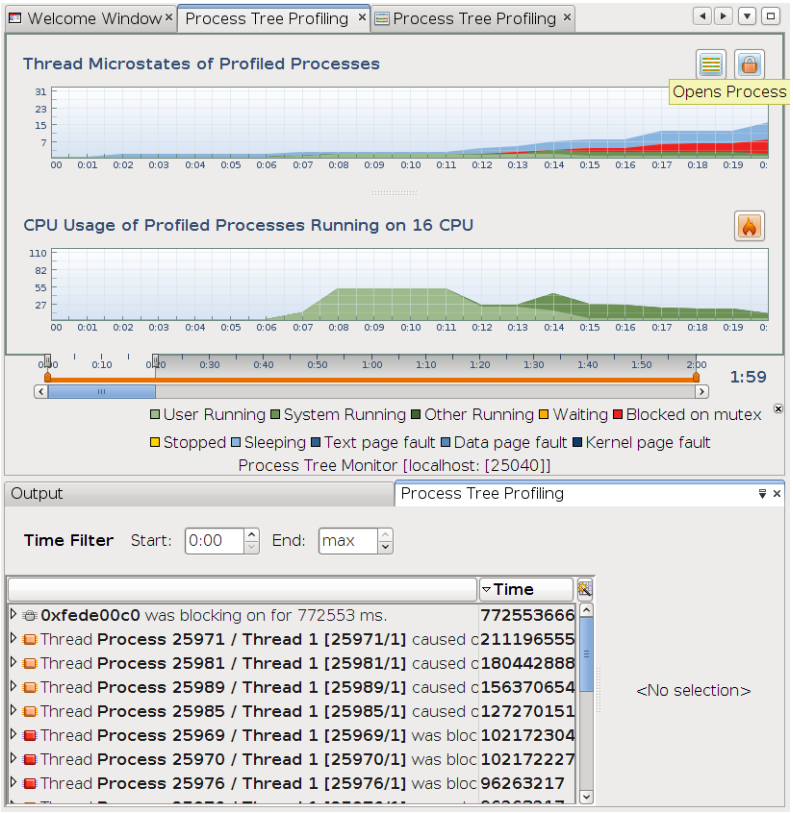


呼び出しスタックでは、プログラムのソースコードが表示できる場合、強調表示された関数はユーザーのソースに出現する関数を示しています。強調表示された関数をダブルクリックすると、DLight の新しいウィンドウにソースを表示できます。実行可能ファイルがデバッグ情報付きで (-g コンパイラオプションを使用して) 構築されている場合は、DLight でソースコードのナビゲーションも可能です。

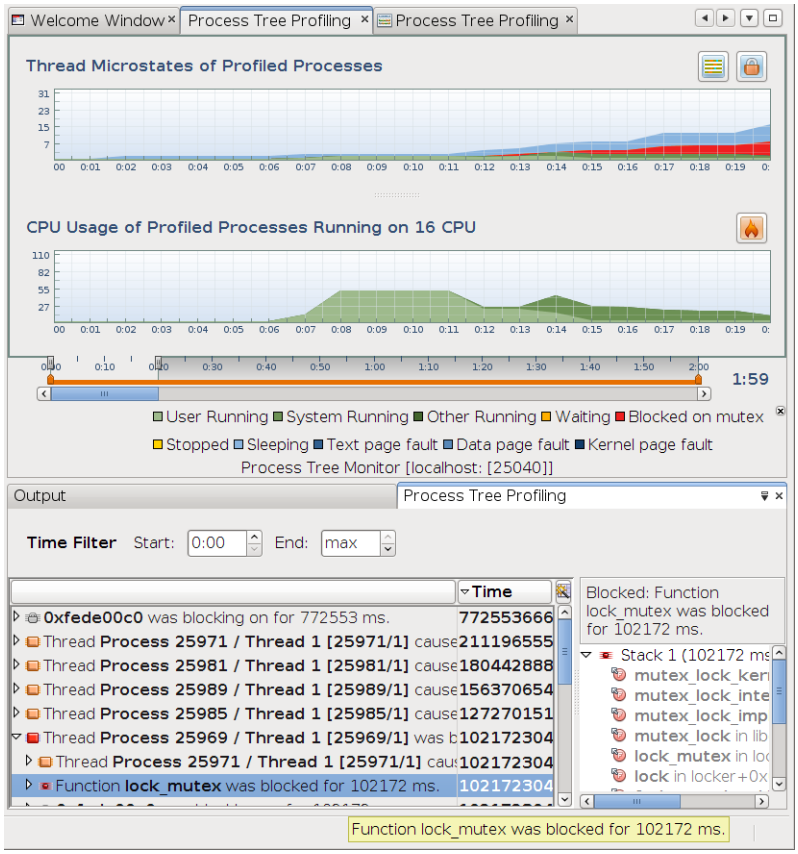
マイクロステートのタイムラインの表示に関する詳細は、DLight のメニューバーから「ヘルプ」⇒「ヘルプの目次」を選択し、DLight のヘルプを参照してください。

13. メインの「プロセスツリーのプロファイリング」タブをクリックして、「プロファイルされたプロセスのスレッドマイクロステート」および「プロファイルされたプロセスの CPU 使用率」のグラフに戻ります。
14. 「ロック状態」ボタン  をクリックして、プロセスとスレッドのマイクロステートに関する詳細情報を示す新しいタブを開きます。

    新しい「プロセスツリーのプロファイリング」タブが開き、ブロックされているスレッドのブロックされた時間、ロックを保持しているスレッド、相互排他ロック、スレッドがブロックされたコード行など、プロセスおよびその子に関するロック統計が表示されます。



15. 左側のハンドルをクリックして、ブロックしているまたはブロックされているスレッド、関数、およびコード行の間の関係を表示します。
16. リスト内の項目をクリックすると、横のウィンドウにスレッド呼び出しスタックが開きます。



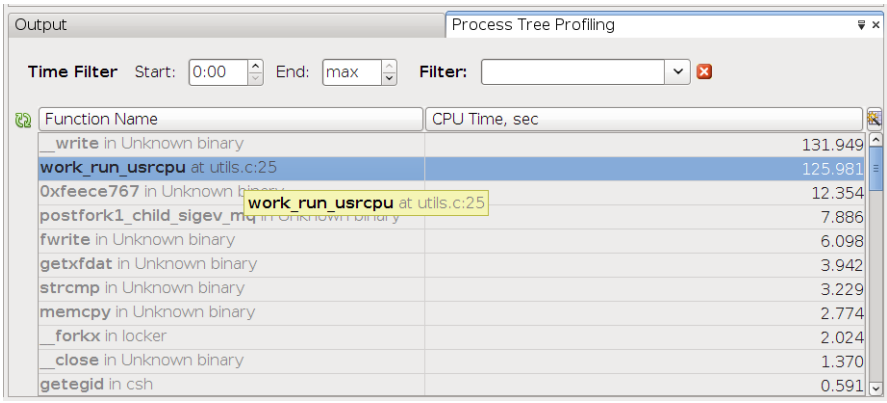
ブロックによって複数の呼び出しスタックが影響を受けた場合は、このウィンドウに各スタックが表示されます。リスト内の項目に示されているブロックされた時間は、その項目に対するすべての呼び出しを集約したものです。呼び出しスタックウィンドウでは、各呼び出しスタックでブロック時間がいくら使用されたかを確認できます。もっとも多くの時間を消費する呼び出しスタックがリストの最初に表示されます。すべてのブロック時間がコードの 1 行だけに起因する場合でも、その行がコードのさまざまなパスで実行されると、複数の呼び出しスタックが表示されることがあります。

17. 呼び出しスタック内の関数のいくつかは太字で表示されています。太字の関数名をクリックすると、示されている行番号の位置で、DLight にソースファイルが開きます。

    DLight がソースコードにアクセスできるため、関数が太字で表示され、ソースファイル名と行番号が表示されています。

18. メインの「プロセスツリーのプロファイリング」タブをクリックして、「プロファイルされたプロセスのスレッドマイクロステート」および「プロファイルされたプロセスの CPU 使用率」のグラフに戻ります。

19. 「ホットスポット」ボタン  を「プロファイルされたプロセスの CPU 使用率」パネルでクリックすると、プログラムのプロセスツリー内で CPU 負荷の大きい領域 (ホットスポット) を特定するのに役立つ新しいタブが開きます。



このタブには、プログラム内の関数が、その排他的 CPU 時間メトリックとともに表示されます。これは、特定の関数のみに使用された時間であり、そこから呼び出された関数に使用された CPU 時間は含まれません。

リスト内の関数のいくつかは太字で表示されています。

20. 太字の関数名をダブルクリックすると、示されている行番号の位置でソースファイルが開きます。

    DLight がソースファイルを表示できる場合は、ソースを表示する新しいウィンドウが開きます。

詳細情報

DLight のグラフィカルツールを使用してアプリケーションのパフォーマンスの問題を特定する方法の詳細については、DLight のメニューバーから「ヘルプ」⇒「ヘルプの目次」を選択してください。

ヘルプでは、このマニュアルでは扱われていない次のようなトピックが説明されています。

- DLight をローカルで実行して、リモートシステムで実行されているアプリケーションをプロファイルできるようにする、リモートホストのプロファイリング。
- 関心のあるツールだけが表示されるように DLight ツールのサブセットを作成するための、ツール構成の指定。
- 各ツールで実行できるタスクを示す、各タスクの詳細な使用方法。

E26464

Oracle Corporation 500 Oracle Parkway, Redwood City, CA 94065 U.S.A.

ORACLE