Oracle Solaris Studio 12.2 DLight チュートリアル

2010 年 9 月

- 2 ページの「概要」
- 2 ページの「サンプルアプリケーションの構築」
- 3 ページの「DLight の起動」
- 3 ページの「サンプルアプリケーションの実行」
- 3 ページの「インジケータコントロールの使用」
- 6 ページの「スレッドマイクロステートの調査」
- 15 ページの「CPU 使用状況の調査」
- 18 ページの「メモリー使用状況の調査」
- 20 ページの「スレッド使用状況の調査」
- 23 ページの「I/O 使用状況の調査」

概要

DLight は、Oracle Solaris 動的トレース (DTrace) テクノロジに基づく、C/C++ 開発者向けの対話型グラフィカル可観測性ツールです。Solaris プラットフォームで DLight を使用すると、複数の DTrace スクリプトからのデータを同期の手法を用いて分析し、アプリケーションの実行時の問題を根本原因までトレースできます。

ユーザーは、DLight を実行するシステムの root 権限が必要です。DLight の起動時に root ユーザーとしてログインしていない場合、プログラムを最初に実行する時に root パスワードの入力を求められます。

DLight には C、C++、および Fortran プログラム向けの次の 5 つのプロファイルツールが含まれます。

- スレッドマイクロステート
- CPU使用状況
- メモリー使用状況
- スレッド使用状況
- I/O 使用状況

また DLight には、AMP (Apache、MySQL、PHP) スタックのプロセス用に次の 3 つのプロファイルツールが含まれます。

- Apache モニター
- MySQL モニター
- PHP モニター

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、各ツールによって「実行監視 (Run Monitor)」ウィンドウのグラフに使用状況情報が表示されます。各グラフには、クリックして詳細情報を表示するボタンがあります。ウィンドウ下部にあるインジケータコントロールでは、グラフの表示を制御できます。

サンプルアプリケーションの構築

インストールされた Oracle Solaris Studio 12.2 の `examples/dlight/ProfilingDemo` ディレクトリにある `ProfilingDemo` アプリケーションを使用します。

1. ProfilingDemo ディレクトリの内容を、ユーザー専用の作業領域にコピーします。

   ```
   cp -r installation_directory/examples/dlight/ProfilingDemo ~/ProfilingDemo
   ```

2. ユーザー独自のプログラムコピーを構築します。

   ```
   cd ~/ProfilingDemo
   make
   ```

   プログラムは、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(このオプションを使用せずにコンパイルを行うと、DLight によってプログラムの実行時のデータの収集と表示はされますが、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

DLight がまだ実行中でない場合は、次のように入力して起動します。

installation_directory`/bin/dlight`

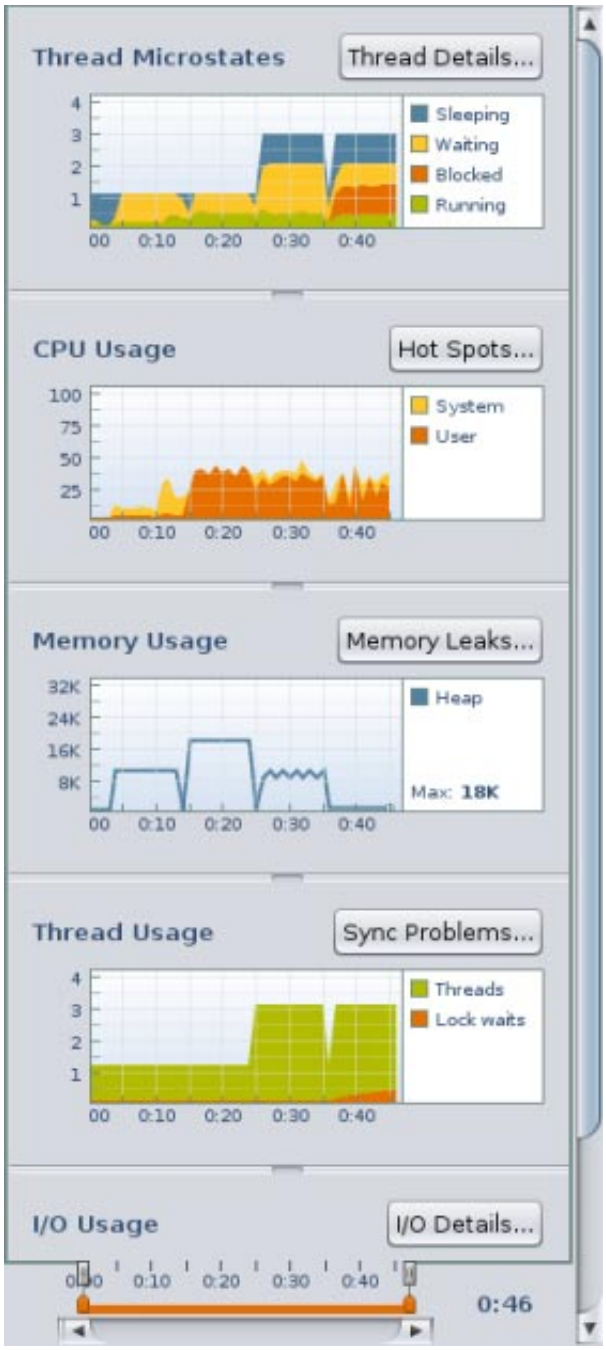
サンプルアプリケーションの実行

1. DLight で、「新規 DLight ターゲット (New DLight Target)」ボタン  をクリックします。ターゲットのダイアログボックスで、次の手順に従います。
   a. 「実行可能なターゲット (Executable Target)」タブをクリックします。
   b. 「実行 (Run)」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 (Browse)」をクリックし、ファイル選択ダイアログボックスを使用して `profilingdemo` 実行可能ファイルに移動します。
   c. サンプルアプリケーションは引数を使用しないため、「引数 (Arguments)」フィールドは空白のままにしておきます。
   d. 「トレース (Trace)」フィールドは実行可能ファイルの子プロセスをトレースする場合にのみ使用するため、このフィールドも空白にしておきます。
   e. 「実行 (Run)」をクリックします。
   f. DLight の起動時に root ユーザーとしてログインしなかった場合は、root パスワードの入力を求められます。
2. `ProfilingDemo` アプリケーションが実行を開始し、「実行監視 (Run Monitor)」ウィンドウが動的グラフの表示を開始します。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。
3. 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。

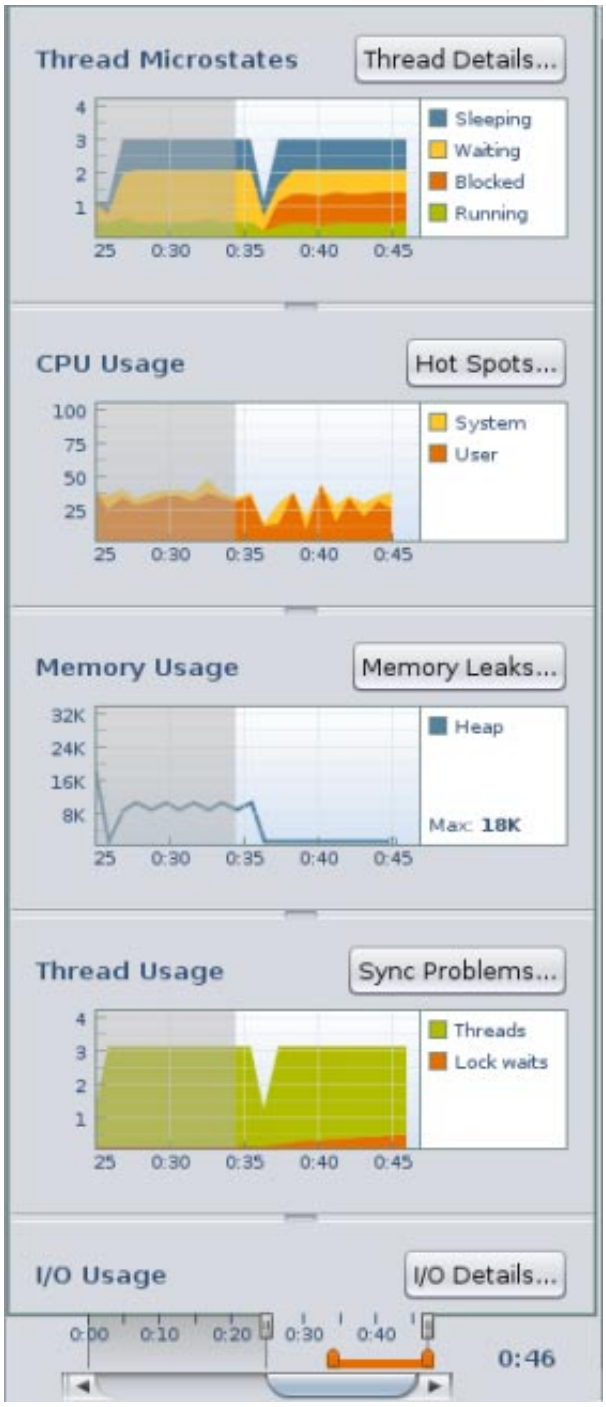
インジケータコントロールの使用

1. 「実行監視 (Run Monitor)」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。スライダに関する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、グラフに表示された実行時の特定の部分を選択できます。
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、実行の開始位置までのすべての過程をドラッグします。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. 次は詳細を見ていきましょう。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。
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。

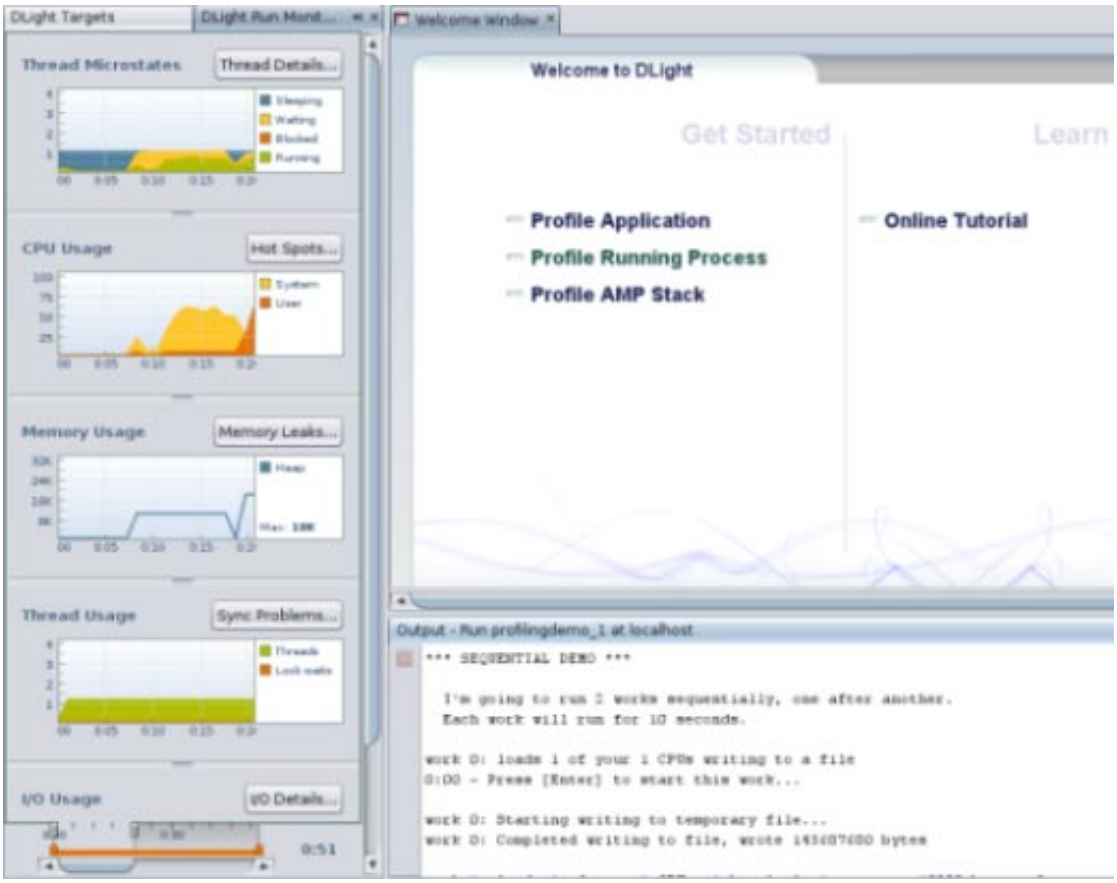
スレッドマイクロステートの調査

「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。

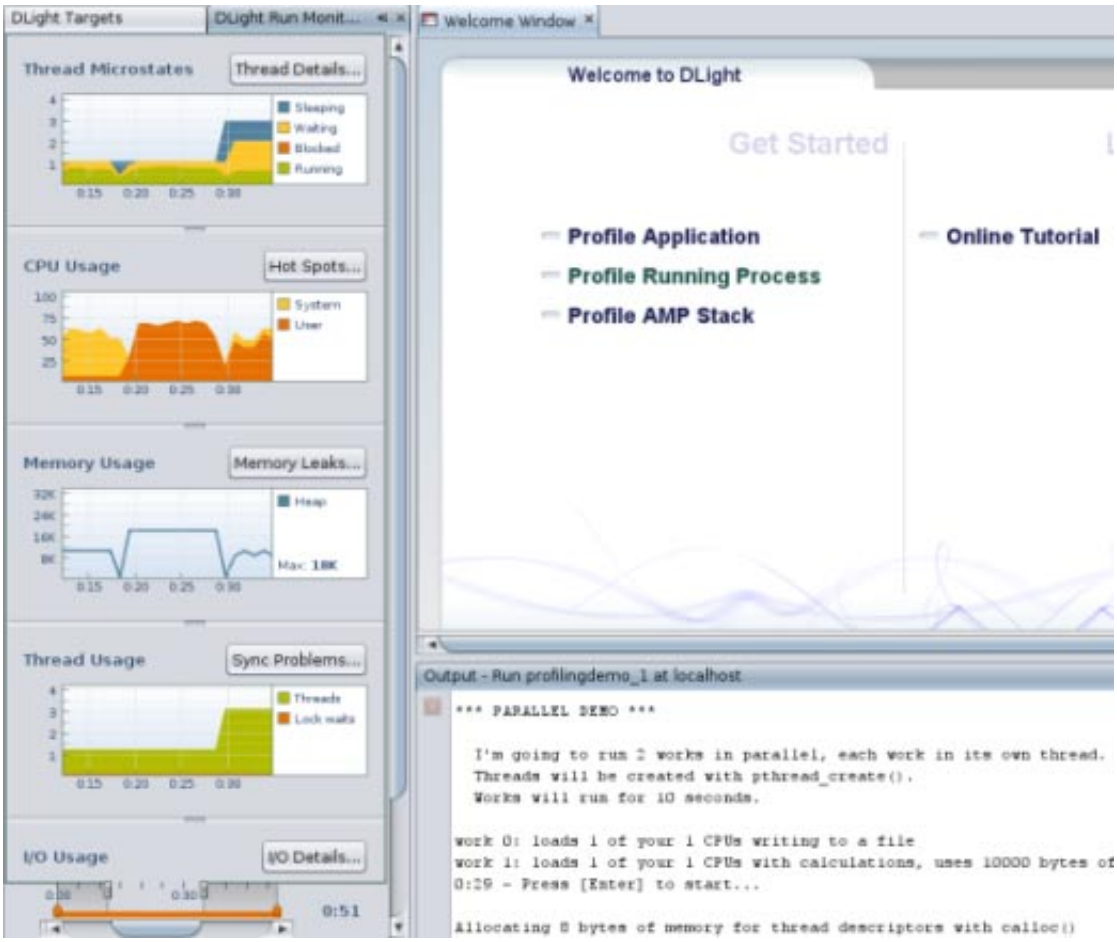
ユーザー実行中 (User Running)	ユーザーモードでプロセスが費やした時間の割合
システム実行中 (System Running)	システムモードでプロセスが費やした時間の割合
その他で実行中 (Other running)	システムトラップなどの処理でプロセスが費やした時間の割合
テキストページフォルト (Text page fault)	テキストページフォルトの処理でプロセスが費やした時間の割合
データページフォルト (Data page fault)	データページフォルトの処理でプロセスが費やした時間の割合
ブロック (Blocked)	ユーザーロックの待機にプロセスが費やした時間の割合
休眠中 (Sleeping)	休眠でプロセスが費やした時間の割合
待機中 (Waiting)	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。

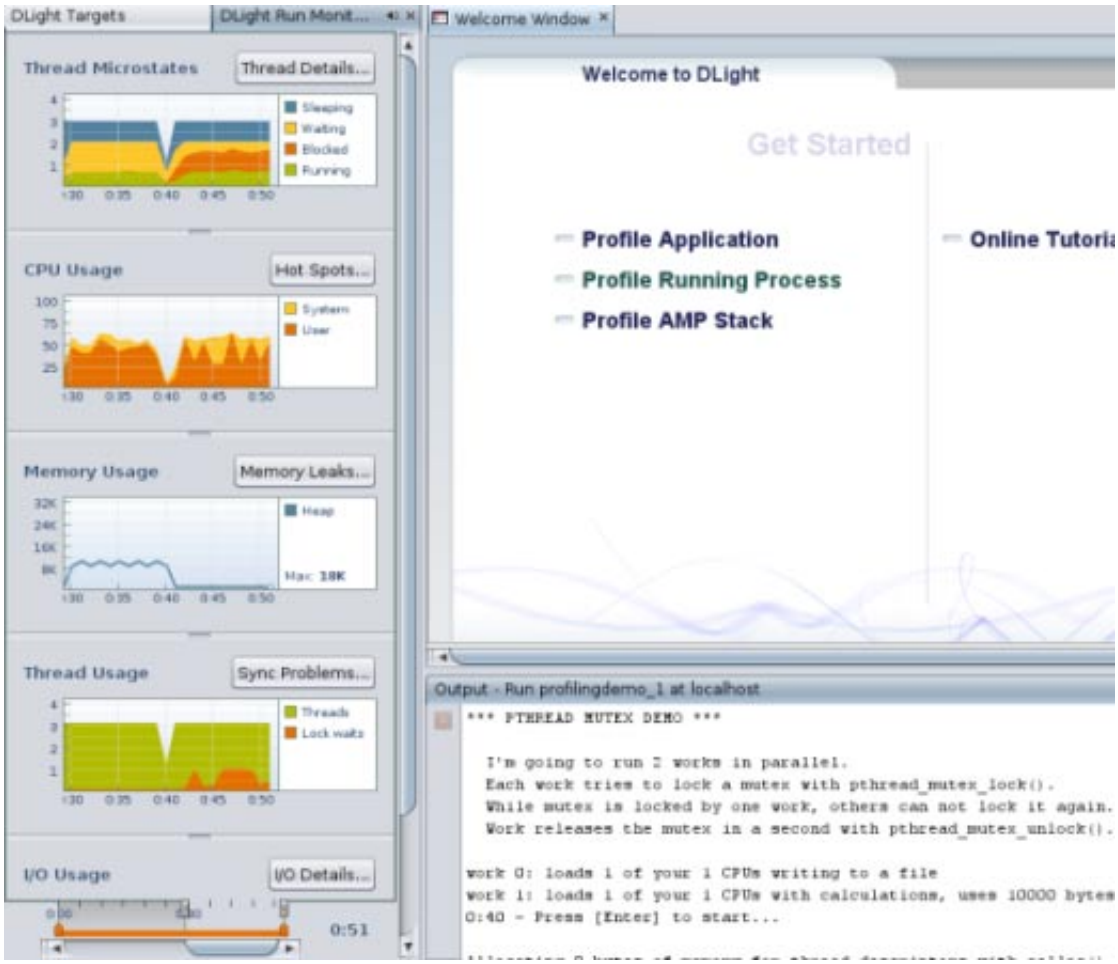
1. 表示スライダの左ハンドルを左に移動して、次の図のようにグラフが実行時の約 20 秒間を示すようにします。次のイメージではプログラムの実行開始位置、`SEQUENTIAL DEMO` 部分の期間が表示されています。この部分ではシングルスレッドで 2 つのタスクが交互に実行されます。スレッドが休眠中のポイントは、ユーザーが Enter キーをクリックするのをプログラムが待機している時点に相当します。



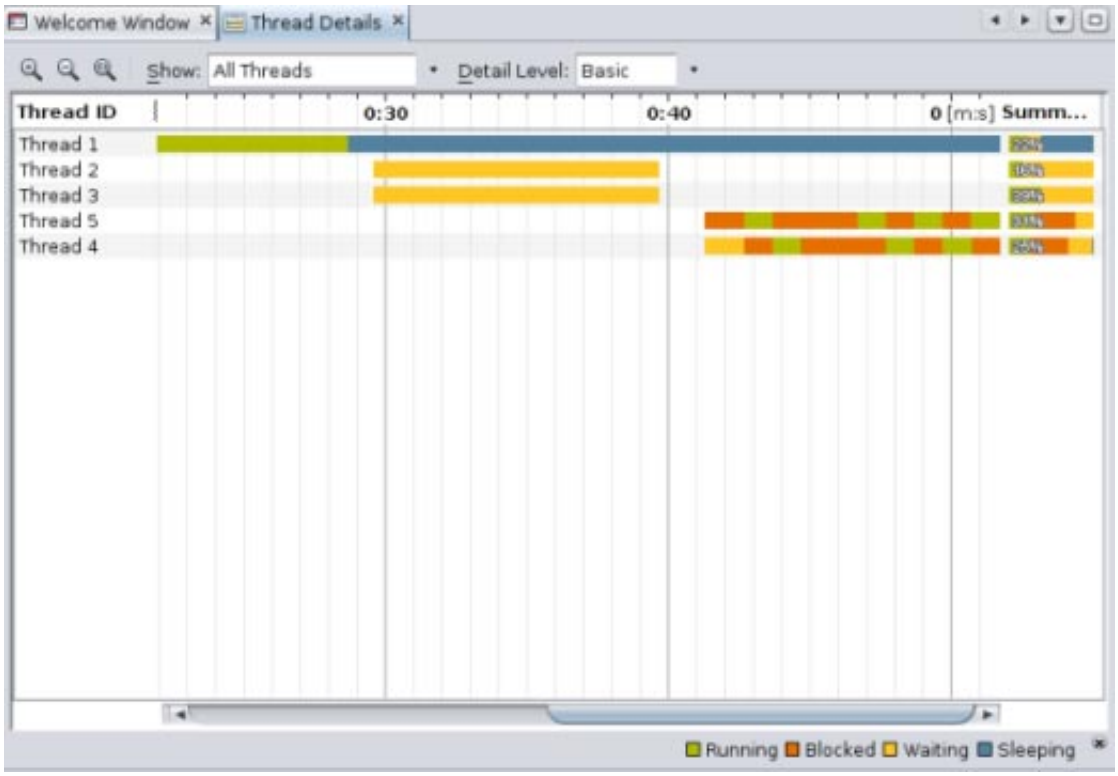
2. 時間スライダをクリックして右にドラッグし、スレッドマイクロステートツールに表示されるスレッド数が 3 に増加するところまで移動します。
3. プログラムが PARALLEL DEMO 部分に入るとスレッドの数は 3 つになっています。メインスレッドから 2 つの追加スレッドが起動し、それぞれのスレッドにある 2 つのタスクを並行して実行します。「待機中 (Waiting)」状態 (黄色) と「休眠中 (Sleeping)」状態 (青色) でかなりの時間が費やされ、「実行中 (Running)」状態 (緑色)にはそれほど時間が使われていないのがわかります。PARALLEL DEMO 部分に、「ブロック (Blocked)」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。

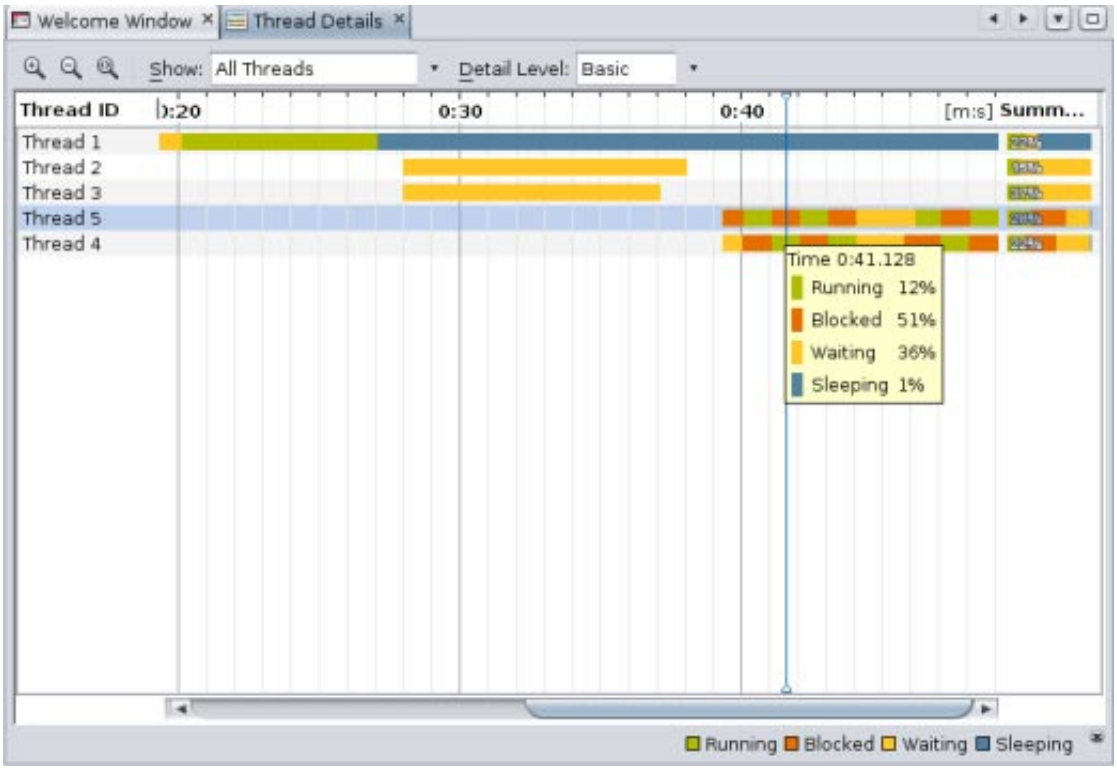


5. 「スレッドの詳細 (Thread Details)」ボタンをクリックすると、スレッドマイクロステートに関する詳細が表示されます。「スレッドの詳細 (Thread Details)」が開き、状態の詳細情報とともに、プログラムで実行されたすべてのスレッドがタイムラインでグラフィカルに表示されます。

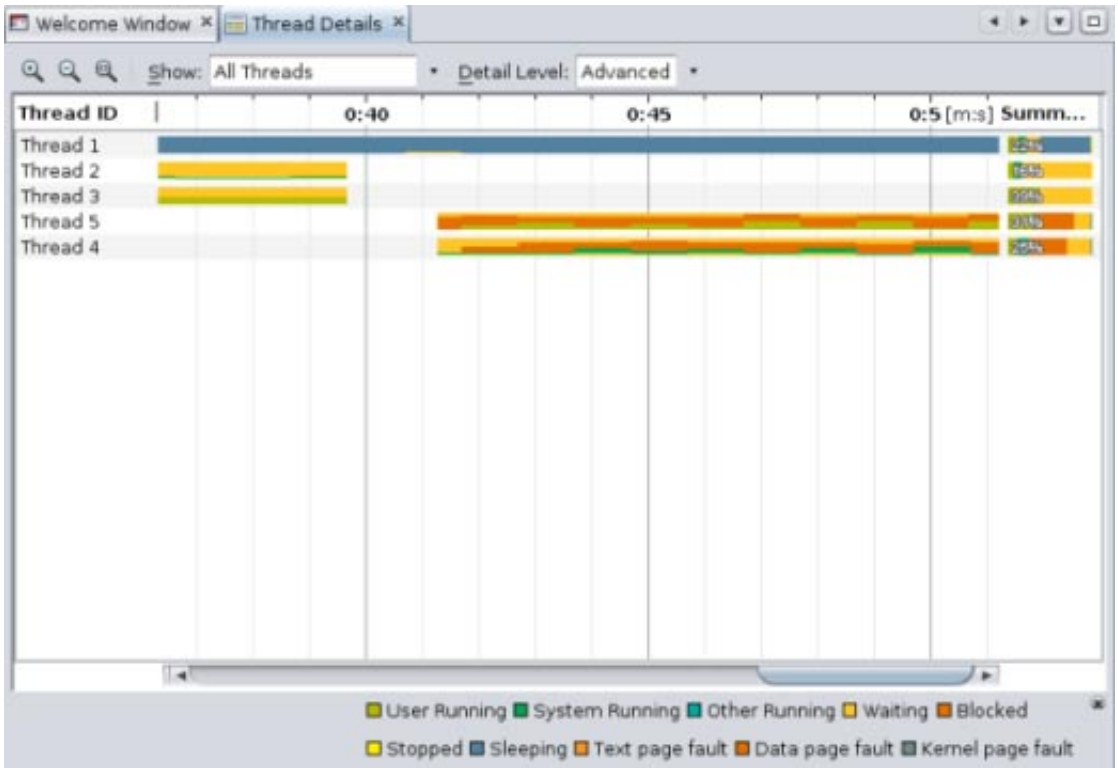


「スレッドの詳細 (Thread Details)」ウィンドウには、プログラムの実行時間全体での各スレッドに関する状態の推移が表示されます。

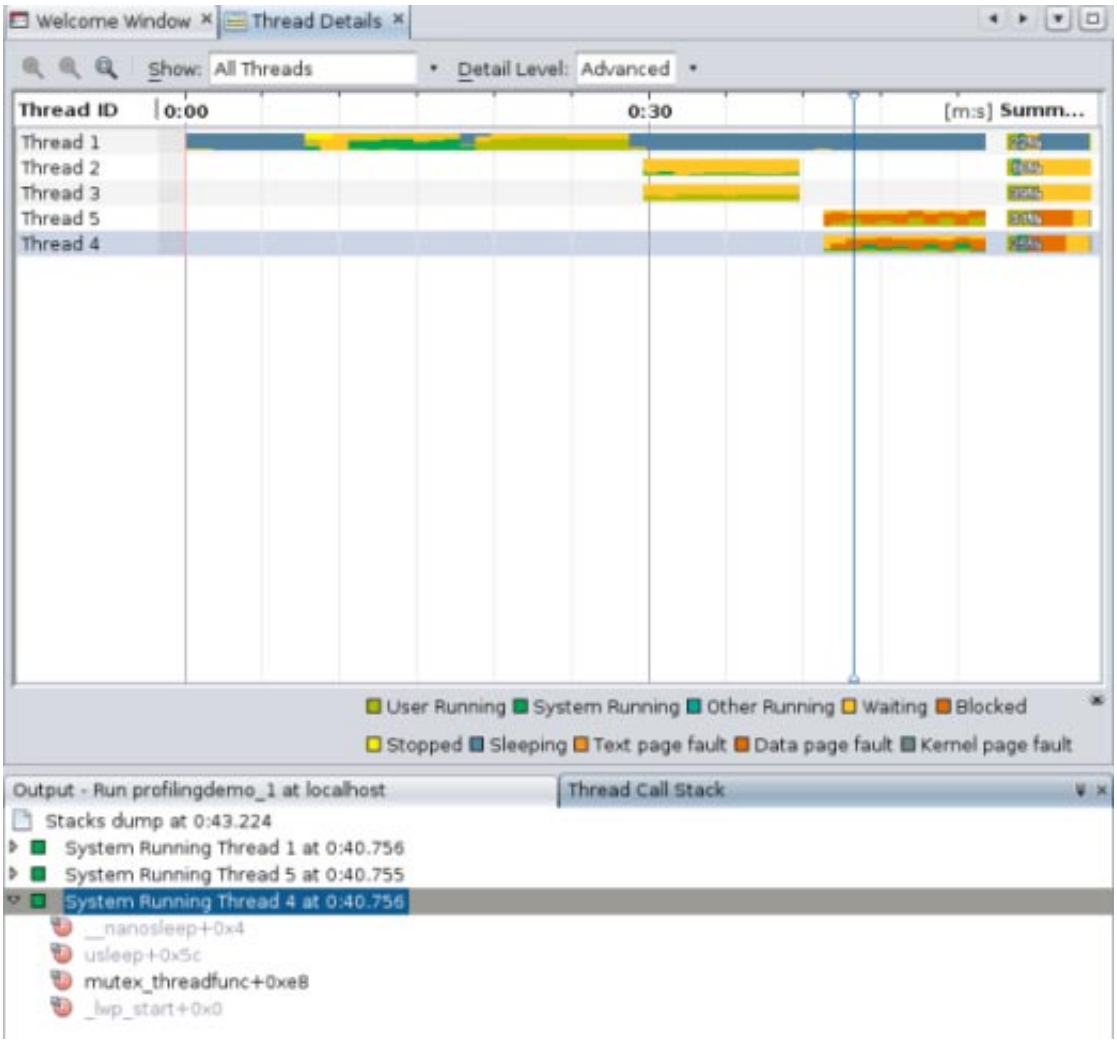
6. カーソルを、スレッドの色が付いた領域のいずれかに置きます。その特定の時点でそのスレッドで行われていることについての詳細を示すポップアップウィンドウが表示されます。詳細には、データが収集された時間と、その時点で各スレッド状態で費やされた時間の割合が含まれます。カーソルをウィンドウ右側にある「概要 (Summary)」領域に置くと、そのスレッドの実行全体における各状態の割合が、ポップアップウィンドウに表示されます。



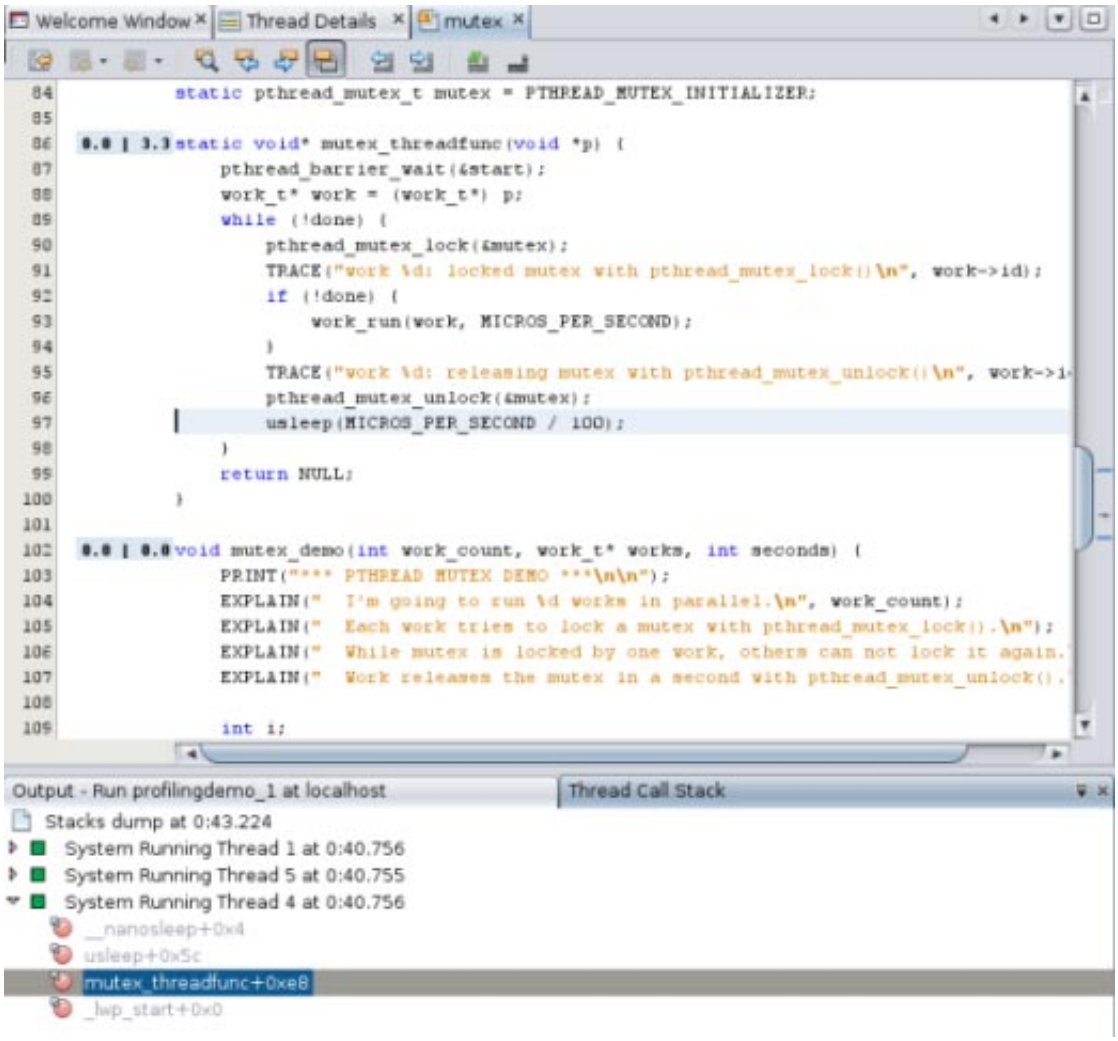
7. 表示内容を変更するには、ウィンドウのコントロールを使用してみてください。
   - デフォルトではウィンドウにはすべてのスレッドが表示されます。「表示 (Show)」ドロップダウンリストの右側にある下矢印をクリックします。終了していないスレッドのみを表示する「ライブスレッドのみ (Live Threads only)」を選択するか、プログラム実行中に終了したスレッドのみを表示する「終了スレッドのみ (Finished Threads only)」を選択できます。
   - デフォルトでは、ウィンドウには 4 つの汎用実行状態のみが表示されます。「詳細レベル (Detail Level)」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。
8. もっと詳しく見るには、拡大ボタン  をクリックしてスレッドのグラフを拡大します。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)。

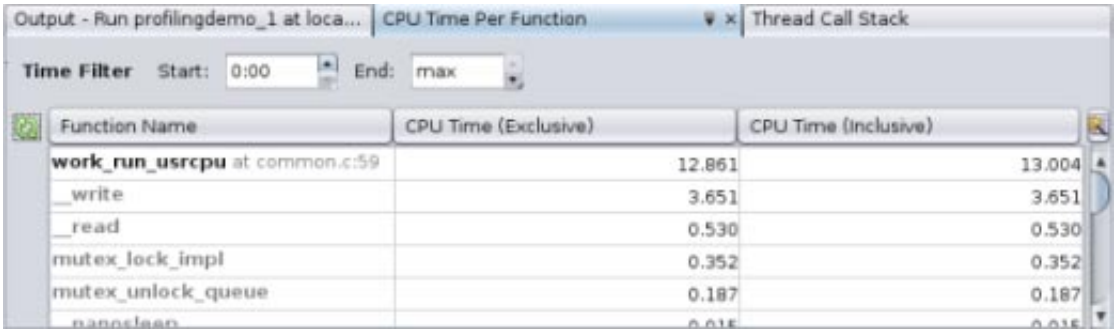


13. 「スレッドの詳細 (Thread Details)」タブをクリックすると、「スレッドの詳細 (Thread Details)」ウィンドウに戻ります。スレッドをクリックします。マウスまたはキーボードを使用して、スレッドのタイムラインに沿って移動できます。
    - マウスを使用する場合は、スレッドを右クリックし、「ナビゲート (Navigate)」メニューを使用してフォーカスを左または右に移動し、タイムライン上のポイントを設定し、「スレッド呼び出しスタック (Thread Call stack)」タブの内容を更新するか、「スレッド呼び出しスタック (Thread Call stack)」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。

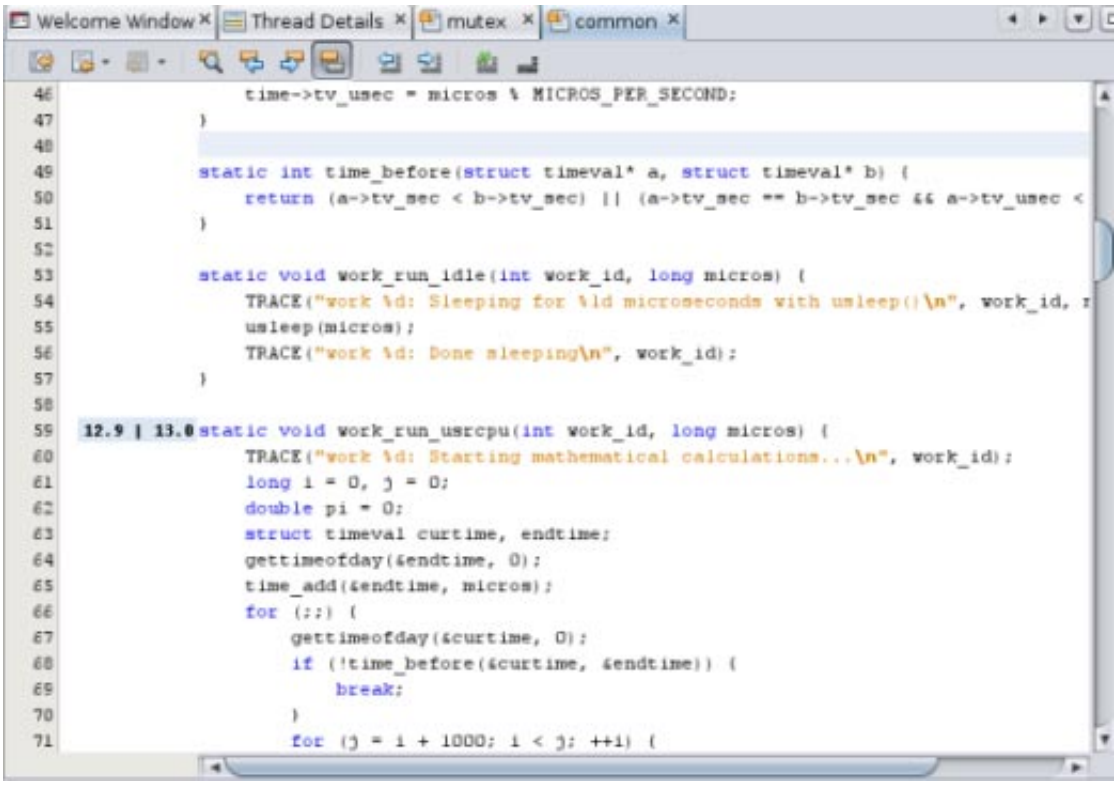
CPU 使用状況の調査

「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。

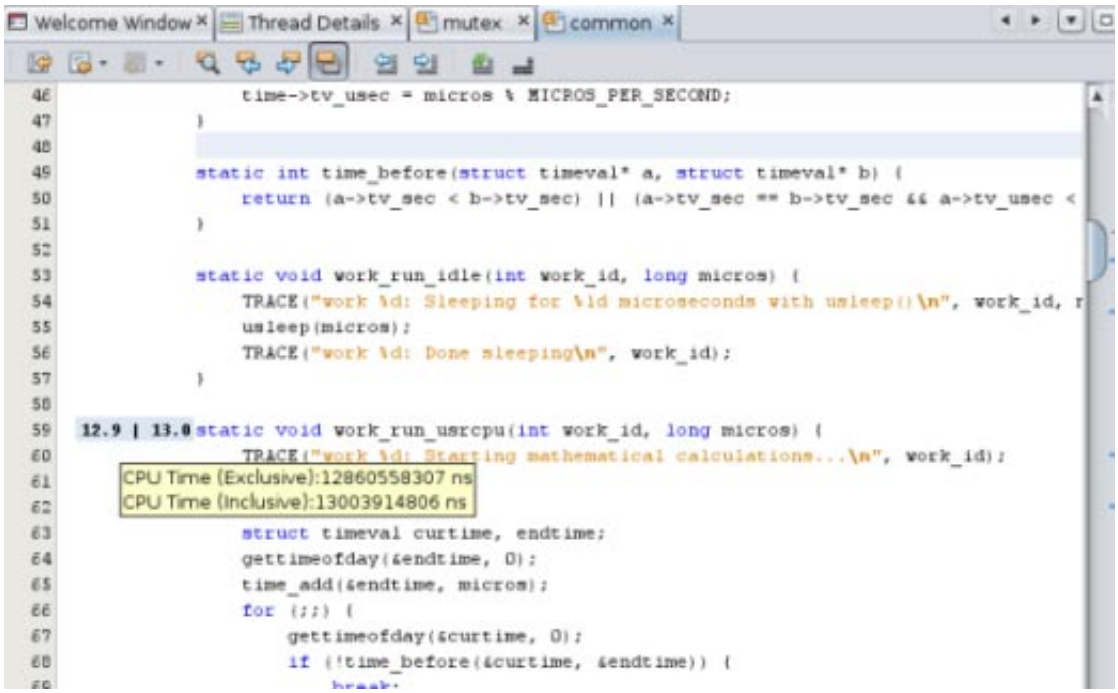
1. 「ホットスポット (Hot Spots)」ボタンをクリックすると、CPU 時間に関する詳細が表示されます。「関数あたりの CPU 時間 (CPU Time Per Function)」タブが開き、プログラムの関数が、各関数によって使用された CPU の時間とともに表示されます。関数は使用された CPU 時間の順に一覧表示されるため、使用時間がもっとも長い関数が最初に表示されます。プログラムがまだ実行中の場合は、初期状態で表示される時間は、ボタンをクリックした時点で費やした時間です。



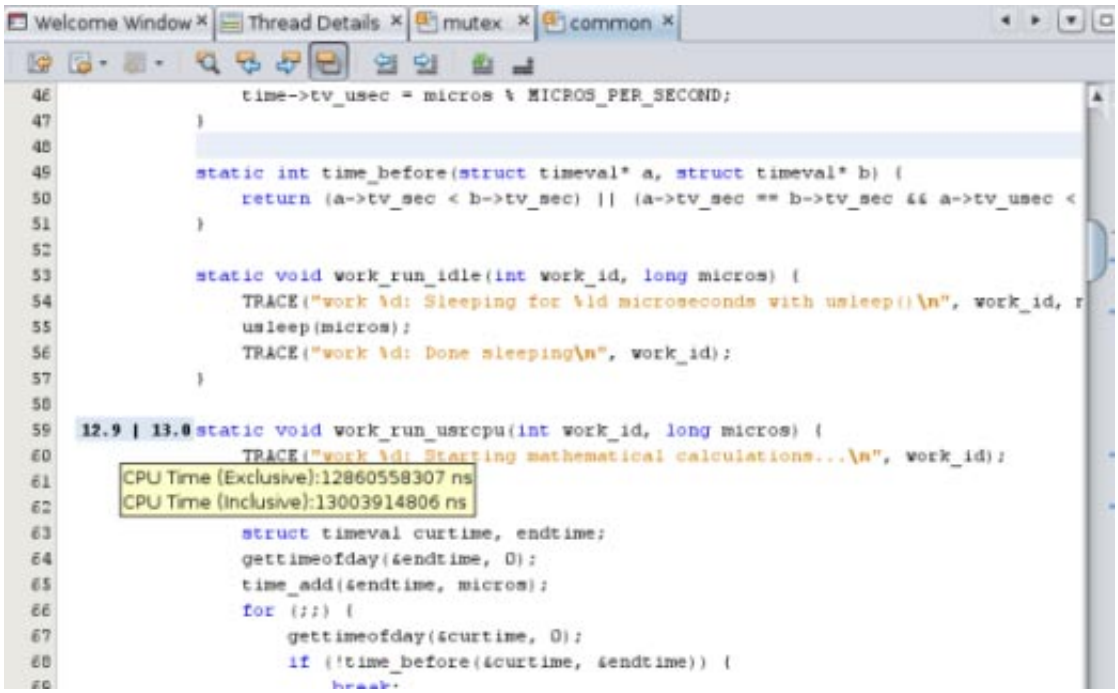
2. 「関数名 (Function Name)」列のヘッダーをクリックすると、関数がアルファベット順にソートされます。
3. 「CPU 時間 (排他的) (CPU Time (Exclusive))」列のヘッダーをクリックすると、個々の関数によって使用された時間の順に関数がソートされます。
4. CPU 時間の 2 つの列の違いを確認します。「CPU 時間 (包括的) (CPU Time (Inclusive))」には、関数の実行開始から終了までの間に使用された CPU 時間の合計が表示されます。また、一覧表示された関数によって呼び出されたその他すべての関数の時間も含まれます。「CPU 時間 (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。この行の左マージンに、いくつかの数字が表示されています。



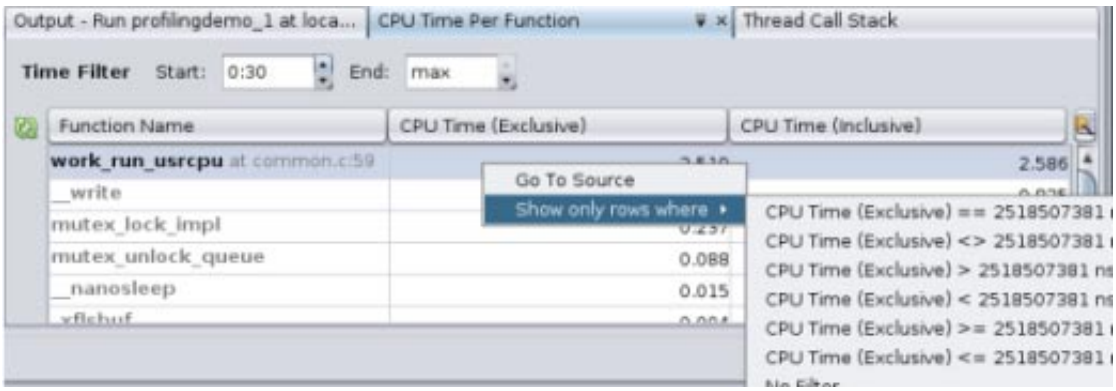
7. 左マージンにある数字の上にマウスカーソルを合わせます。これらの数字は、「関数あたりの CPU 時間 (CPU Time Per Function)」タブに表示される関数の包括的および排他的 CPU 時間と同じメトリックです。表示領域を少なくするためにメトリックは丸めて表示されますが、マウスをその上に合わせると、丸められていない値が表示されます。`work_run_usrcpu` 関数内で計算を行う for ループなど、CPU を消費する行のメトリックは、`common.c` ソースファイルにも表示されます。



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。「実行監視 (Run Monitor)」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。

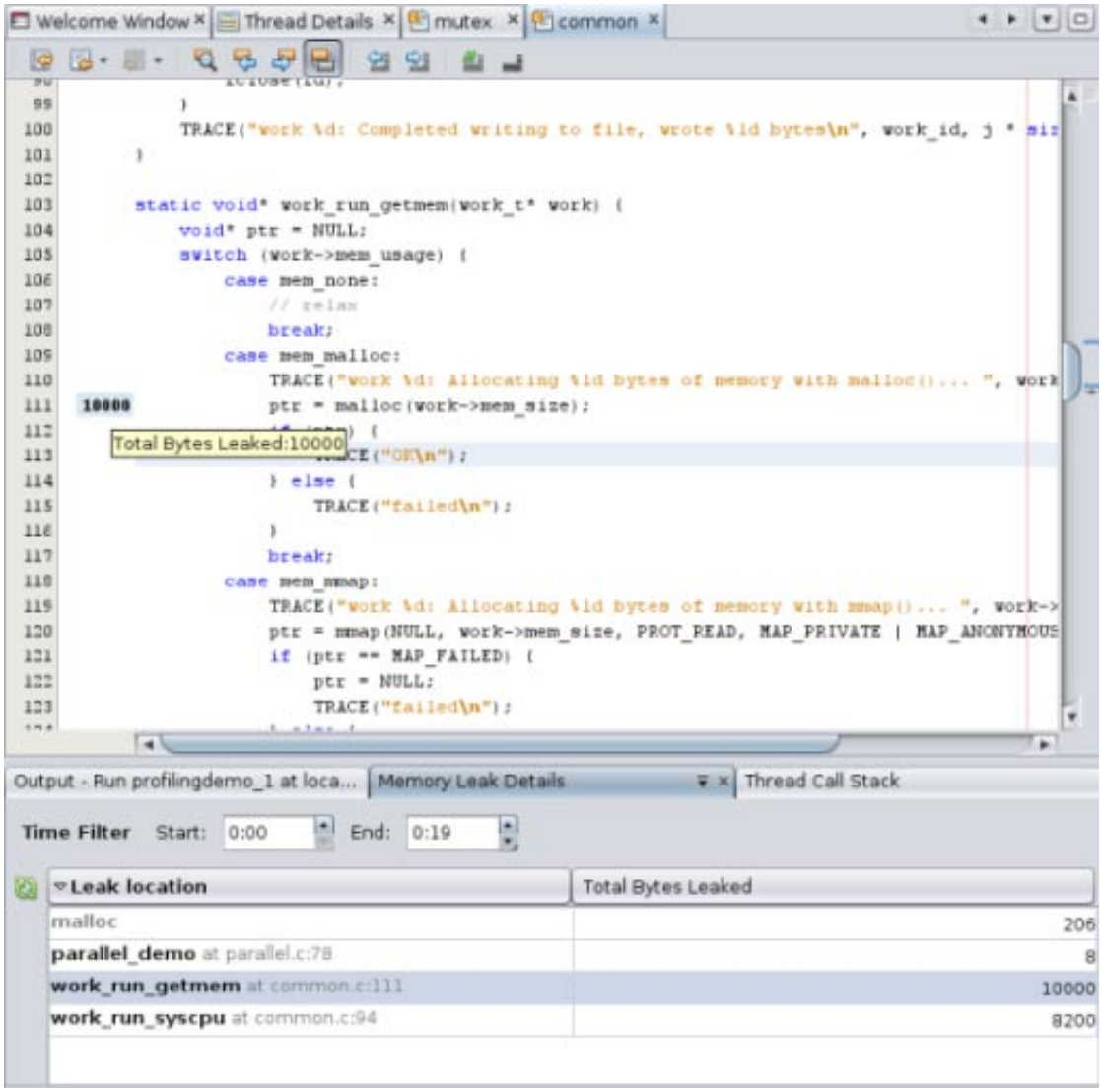


メモリー使用状況の調査

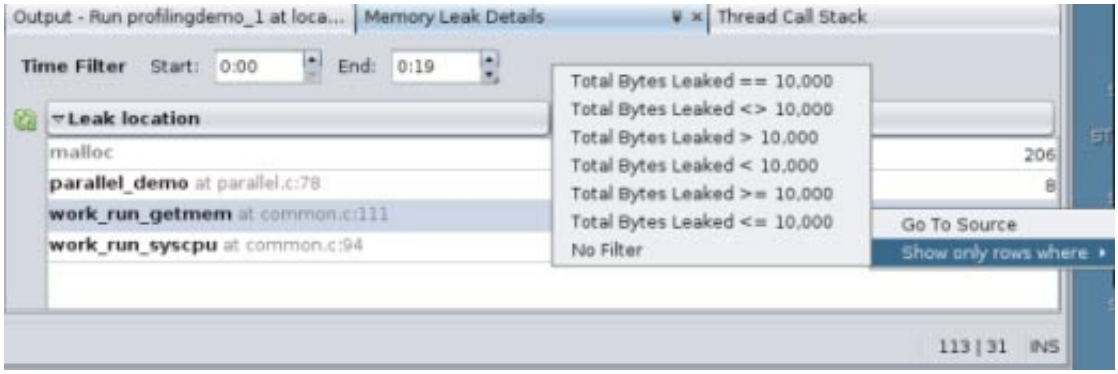
「メモリー使用 (Memory Usage)」ツールには、プログラム実行時のメモリーヒープの経時変化が表示されます。これは、プログラム内で不要になったメモリーの開放に失敗した場所をポイントする、メモリーリークの特定に使用できます。メモリーリークは、プログラムのメモリー消費が増加する原因となります。メモリーリークが発生しているプログラムの実行時間が長くなると、結果的に使用できるメモリーが不足する場合があります。

1. 「実行監視 (Run Monitor)」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. 「実行監視 (Run Monitor)」ウィンドウで「開始 (Start)」時間および「終了 (End)」時間を変更するか、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。



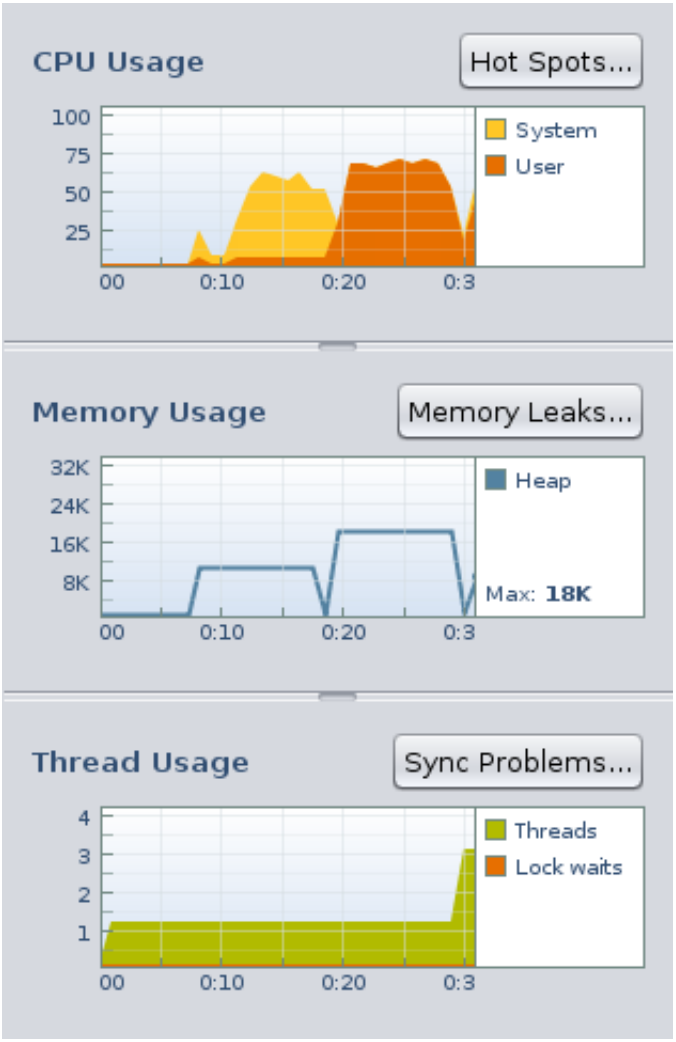
6. 「関数あたりの CPU 時間 (CPU Time Per Function)」タブと同じように、「メモリーリーク詳細 (Memory Leak Detail)」タブでメトリックを右クリックしてデータをフィルタする基準を選択します。



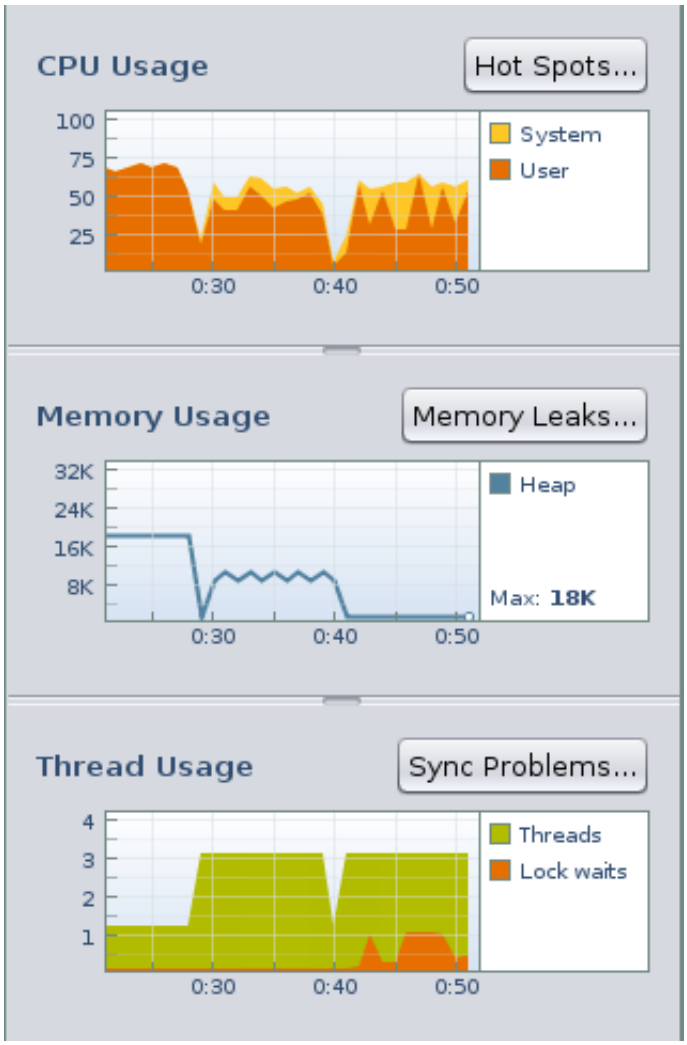
スレッド使用状況の調査

「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。

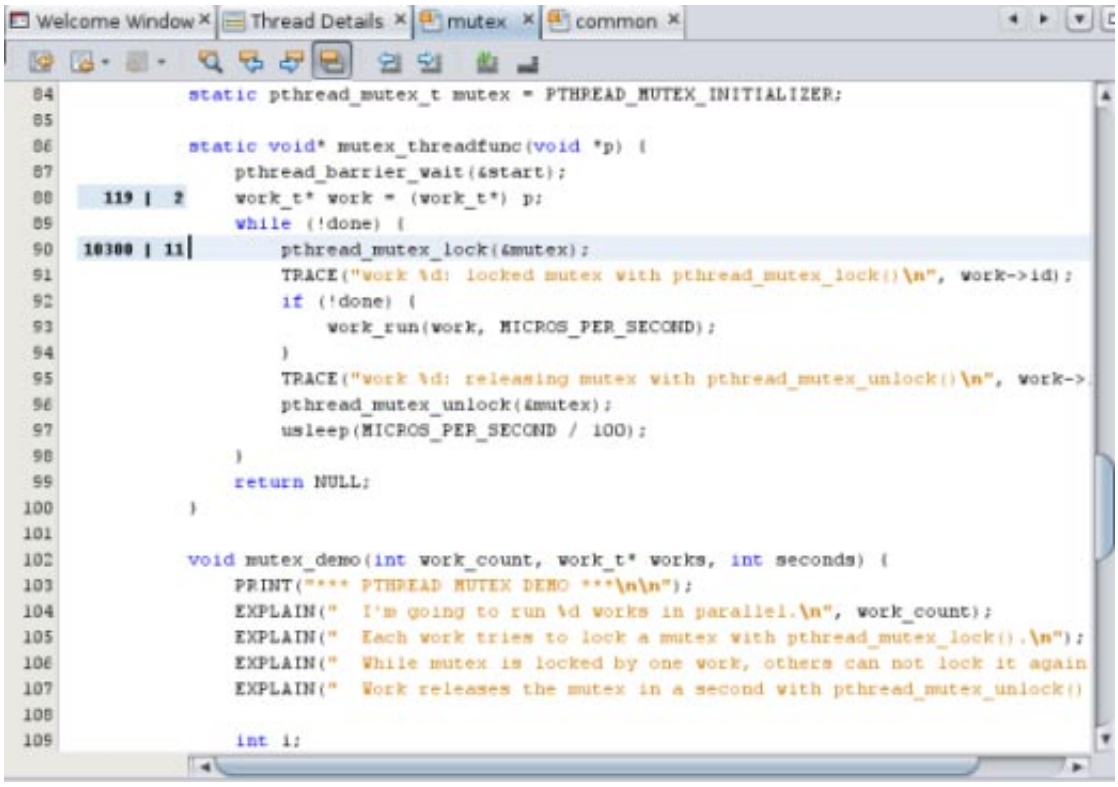


4. 時間スライダを右にスライドさせ、スレッドが 3 に増加する 2 か所のポイントを確認できます。スレッドの 1 回目の増加は、プログラムの実行の `PARALLEL DEMO` 部分に対応します。ここで、メインスレッドがファイルへの書き込みと計算の実行の作業を並行して行うため、2 つの追加のスレッドを開始します。この部分でメモリー使用量と CPU 使用量が少し増加しますが、`SEQUENTIAL DEMO` 部分の場合よりもずっと短い時間で 2 つのタスクが完了します。



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、相互排他ロックを使用します。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. 「ロック待機数 (Lock Waits)」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。

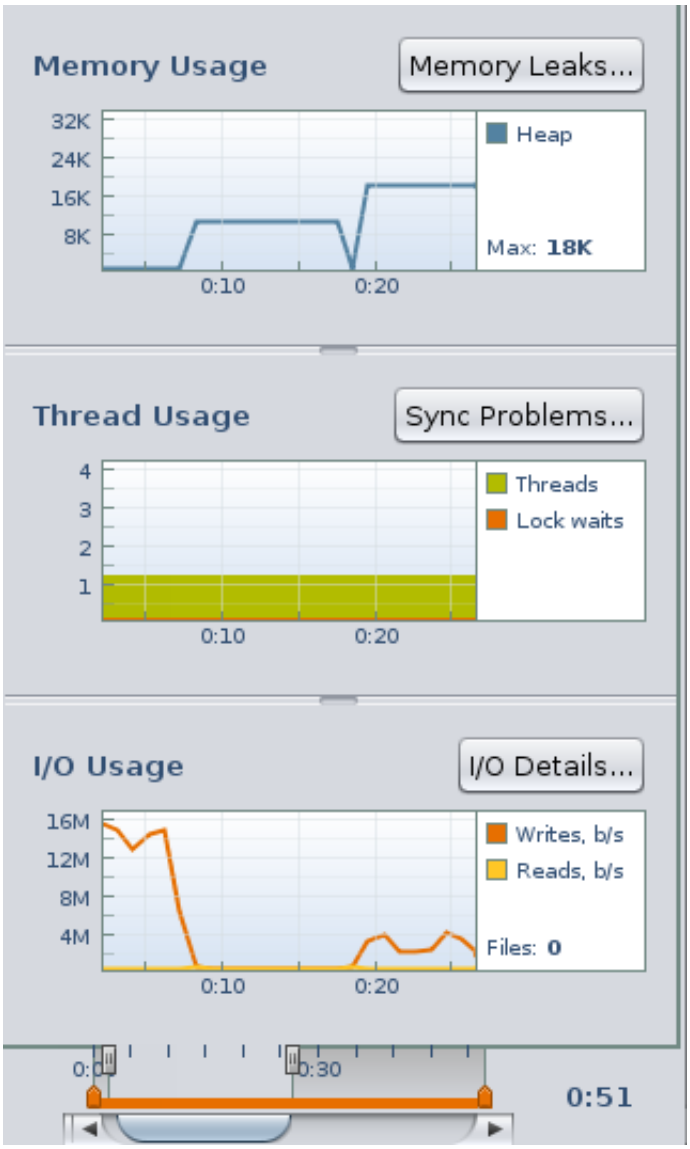


12. 待ち時間およびロック待機数のメトリックは、ソースファイルの左側の列のマージンに表示されます。メトリックの上にマウスカーソルを合わせると詳細が表示されます。この詳細は「スレッド同期の詳細 (Thread Synchronization Details)」タブの表示内容と一致します。
13. メトリックを右クリックして「プロファイラメトリックを表示 (Show Profiler Metrics)」を選択解除します。これでメトリックはどのプロファイルツールのソースエディタにも表示されなくなります。
14. 「表示 (View)」 > 「プロファイラメトリックを表示 (Show Profiler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

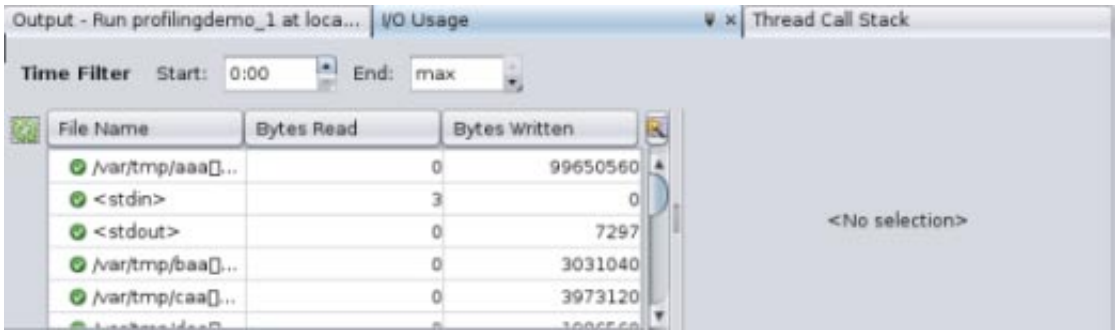
I/O 使用状況の調査

「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。

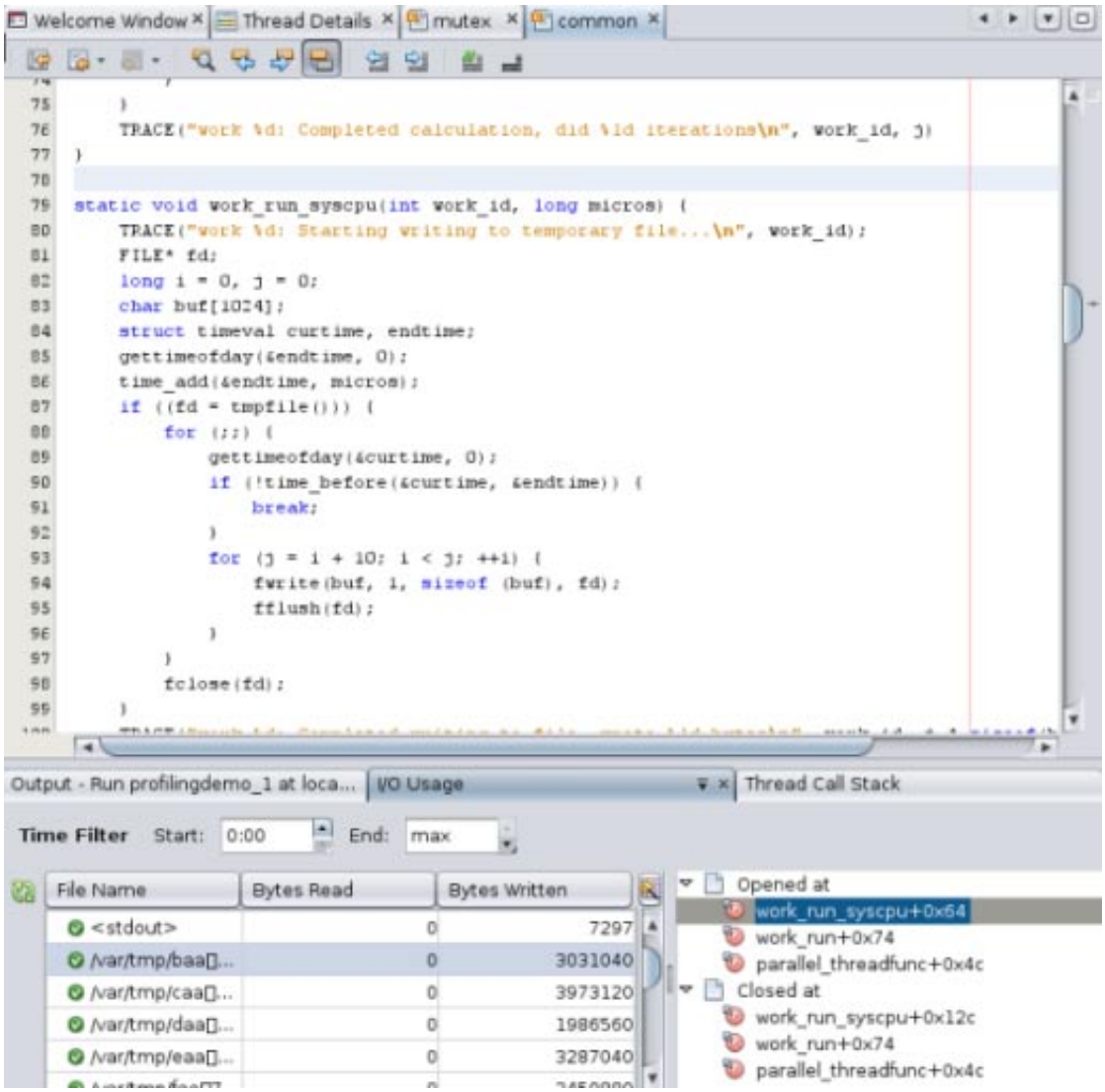
1. 次のイメージには、シングルスレッドで 2 つのタスクが交互に実行される SEQUENTIAL DEMO 部分における、実行の開始位置での I/O 使用状況が表示されています。最初の数秒間でプログラムが開始し、その後ユーザーが Enter キーを押すまで待機していました。ユーザーが Enter キーを押すと、プログラムによって一時ファイルに文字が書き込まれました。この活動は、書き込まれたバイト数を示すオレンジ色の線でグラフに反映されます。



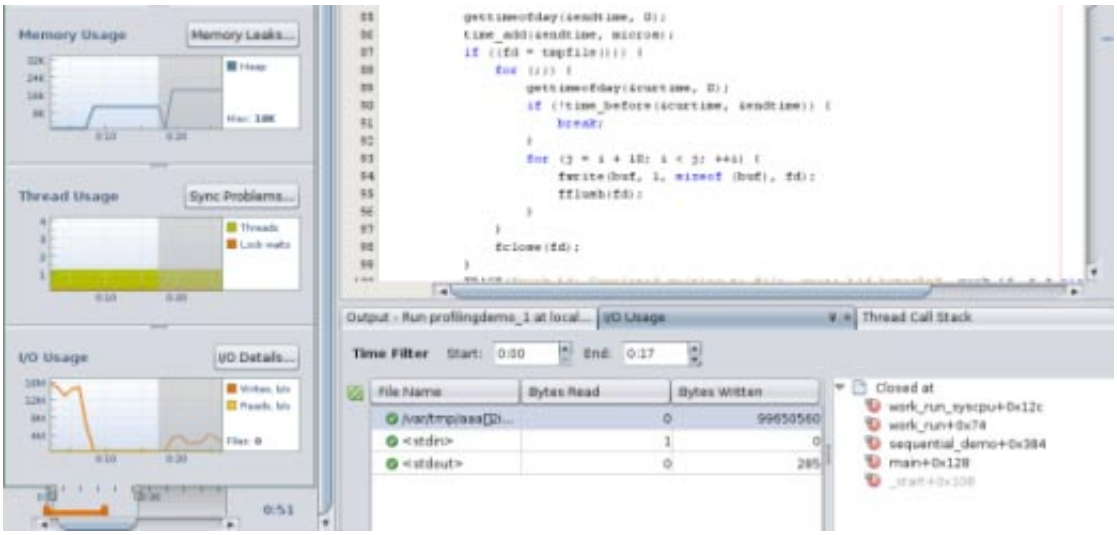
2. 次を確認します。
   - オレンジ色の線は、最近の 1 秒間に書き込まれたバイトの数を示しています。ProfilingDemo プログラム自体は、この `SEQUENTIAL DEMO` の実行中に合計 79984640 バイト、つまり約 76.3M バイトの書き込みを行ったことを「出力 (Output)」ウィンドウに報告します。オレンジ色の線で示されたデータのポイントをすべて合計すると、その値は 76.3M バイト近くになります。
   - プログラムはシステムコールを利用してデータを生成しディスクに書き込むため、このフェーズの間は「CPU 使用 (CPU Usage)」ツールに示さ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。



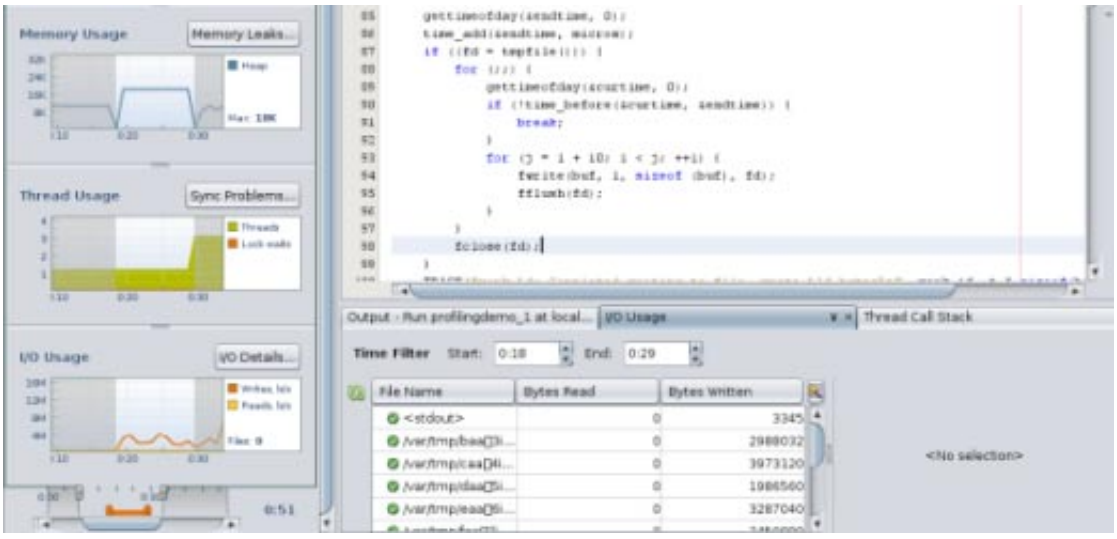
4. このプログラムでは Enter キーで何度か入力するだけであるため、「読み取りバイト数 (Bytes Read)」列はあまり役に立ちません。「読み取りバイト数 (Bytes Read)」列を閉じるには、列ヘッダーの右にある「表示列の変更 (Change Visible Columns)」ボタンをクリックします。「表示列の変更 (Change Visible Columns)」ダイアログボックスで、「読み取りバイト数 (Bytes Read)」を選択解除してから「了解 (OK)」をクリックします。
5. このプログラムでこれらのすべての一時ファイルがどのように使用されているか詳しく知りたいとします。「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。



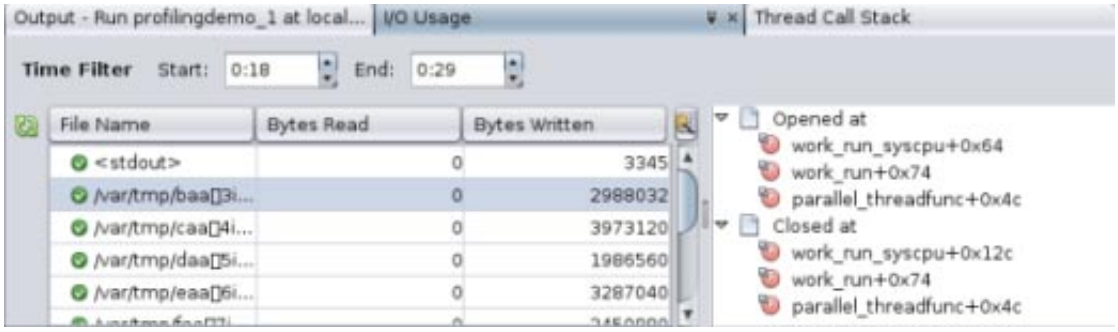
7. 「関数あたりの CPU 時間 (CPU Time Per Function)」タブと「スレッド同期の詳細 (Thread Synchronization Details)」タブで行ったようにように、「I/O 使用 (I/O Usage)」の詳細タブでも表示する期間を指定できます。「実行監視 (Run Monitor)」ウインドウの「I/O 使用 (I/O Usage)」グラフで、書き込み活動は実行の開始位置に非常に近い時点から始まっています。ここはプログラムの `SEQUENTIAL DEMO` 部分が一時ファイルに書き込みを始めるポイントです。
8. 「終了 (End)」フィールドに実行の `SEQUENTIAL DEMO` 部分が終了した時間を入力して、この部分を強調表示します。「実行監視 (Run Monitor)」ウインドウと「I/O 使用 (I/O Usage)」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。



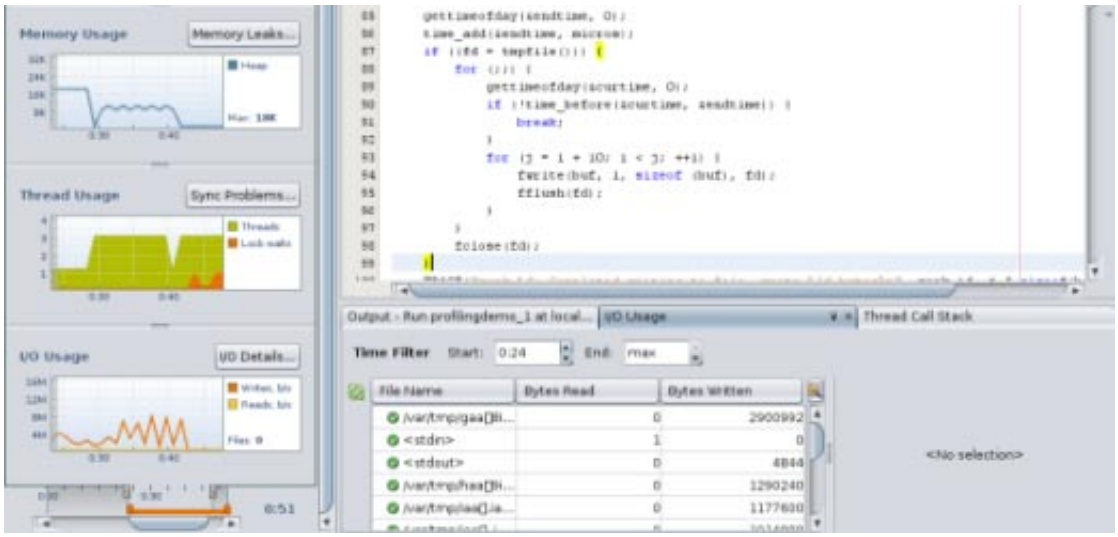
9. SEQUENTIAL DEMO の間、「I/O 使用 (I/O Usage)」の詳細タブには一時ファイルが 1 つ表示されていることを確認します。シングルスレッドが開き、ファイルへの書き込みを行い、ファイルを閉じます。ファイルをクリックしてこのファイルにアクセスした関数を確認し、関数をダブルクリックしてソースコードを表示します。
10. 「実行監視 (Run Monitor)」ウインドウで、書き込み活動が一時停止した後、ふたたび開始するところまで時間スライダを右に移動させます。書き込みの一時停止は、SEQUENTIAL DEMO の 2 番目のタスクの間に起こります。2 番目のタスクは計算タスクで、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。

821-2502

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