

# Oracle® Solaris Studio 12.3 : DLight 教程

2011 年 12 月 , E26465-01

该 DLight 教程说明了如何使用 DLight 启动可执行文件，以及如何使用 DLight 监视图和工具来检查正在运行的程序中的分析数据。本教程还说明了如何连接到进程并检查该进程及其所有子进程和线程中的分析数据。本教程包括以下主题：

- 第 2 页中的“介绍”
- 第 3 页中的“设置 DLight 的 Oracle Solaris 特权”
- 第 4 页中的“下载 Oracle Solaris Studio 样例应用程序”
- 第 4 页中的“生成样例应用程序”
- 第 4 页中的“启动 DLight”
- 第 4 页中的“分析样例应用程序”
- 第 22 页中的“分析进程树”
- 第 28 页中的“有关更多信息”

## 介绍

DLight 是一种交互式图形观察工具，该工具使用 Oracle Solaris 动态跟踪 (DTrace) 技术来收集有关在 Oracle Solaris 上运行的程序的信息。使用 DLight，您可以通过 DTrace 分析程序，而无需了解如何使用 D 脚本语言。DLight 以同步方式运行多个 DTrace 脚本，同时以图形方式向您显示输出，从而可帮助您追踪到应用程序中运行时问题的根本原因。

DLight 的基本工作流程如下所示：

1. 创建目标来定义要分析的进程或可执行文件
2. 运行目标
3. 检查运行目标时打开的 DLight 工具中的数据
4. 与这些工具交互来查看代码中的问题点
5. 如有必要，停止目标

创建目标时，可以指定目标的类型以及应在其中运行目标的系统（本地或远程系统）。

DLight 目标类型包括：

- |                                    |                      |
|------------------------------------|----------------------|
| <b>Executable Target</b> （可执行目标）   | 尚未运行的程序              |
| <b>Process Target</b> （进程目标）       | 正在运行的进程              |
| <b>Process Tree Target</b> （进程树目标） | 正在运行的进程以及该进程产生的所有子进程 |

当您运行可执行目标或进程目标时，每个工具都会在 "DLight Run Monitor"（DLight 运行监视器）窗口中的一个图形中显示使用情况信息。

运行进程树目标时，工具显示在 "Process Tree Profiling"（进程树分析）窗口中，提供有关多个进程和线程活动的更多信息。

本文档稍后部分将介绍 "DLight Run Monitor"（DLight 运行监视器）窗口和 "Process Tree Profiling"（进程树分析）窗口中的工具。

在运行 DLight 的系统上，您必须具有与 DTrace 相关的特定 Oracle Solaris 特权，如下一部分中所述。

---

**提示** – 有关目标、远程主机上的分析进程以及更改工具配置的信息，请参见 "Help"（帮助）⇒ "Help Contents"（帮助内容）。

---

# 设置 DLight 的 Oracle Solaris 特权

DLight 要求为运行它的用户指定 Oracle Solaris 特权：`dtrace_proc`、`dtrace_user` 和 `dtrace_kernel`。这些特权控制用户对 DTrace 功能的访问。如果您的用户名未被指定这些特权，DLight 会提示您输入可设置特权的管理员帐户（例如 `root`）的口令。您第一次在 DLight 会话中运行可执行目标或连接到目标进程时，会出现此提示。

您提供正确的口令后，将使用该管理员帐户向在 DLight 会话过程中运行 DLight 的进程指定所需的特权。如果您在向您的用户名永久性指定了这些特权的系统上或者在您以管理员身份登录的系统上使用 DLight，则 DLight 不会提示输入管理员口令。

---

注—应谨慎使用这些特权，因为它们允许用户深入查看内核的运行。应当仅在用于软件开发的系统上（而不是在生产系统上）对用户启用这些特权。

---

要检查您的 Solaris 特权，请在命令提示符下键入以下内容：

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

如果您的帐户具有所需的特权，`ppriv` 命令应返回类似以下的内容：

```
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
```

以 **I**: 开头的行非常重要，因为它指定从 shell 启动的程序所继承的特权。如果您的帐户没有所需的可继承特权，并且您对系统没有管理员特权或者超级用户权限，您应该要求系统管理员向您的帐户添加 `dtrace_proc`、`dtrace_user` 和 `dtrace_kernel` 可继承特权。

如果您对系统具有管理员特权或者超级用户权限，则可以按如下所述向您的用户帐户授予所需的特权。

向用户帐户永久授予所需的 DTrace 特权：

1. 确保要修改其特权的用户帐户从系统中注销。
2. 以超级用户 (`root`) 或其他管理员用户身份登录。
3. 在命令提示符下键入以下内容，并用您要修改的用户帐户名称替换 *username*：

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

还可以向用户的正在运行的 shell 进程指定特权来临时向该用户授予这些增强特权。

向用户帐户临时授予所需的 DTrace 特权：

1. 在用户的命令提示符下键入以下内容来确定用户 shell 进程的进程 ID：

```
$ echo $$
```

2. 在另一终端中以超级用户 (`root`) 或其他管理员用户身份登录。
3. 键入以下内容，并用从 `echo` 命令返回的进程 ID 替换 *process-ID*。

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

在由 *process-ID* 指定的 shell 中键入的所有命令现在将继承所需的特权。用户应在此 shell 中启动 DLight。

# 下载 Oracle Solaris Studio 样例应用程序

本教程使用名为 ProfilingDemo 的样例程序。

可从 [Oracle Solaris Studio 12.3 样例应用程序 Web 页面 \(http://www.oracle.com/technetwork/server-storage/solarisstudio/downloads/solaris-studio-samples-1408618.html\)](http://www.oracle.com/technetwork/server-storage/solarisstudio/downloads/solaris-studio-samples-1408618.html) 上的样例应用程序 zip 文件中获取该样例程序的源代码。

如果您尚未下载样例应用程序 zip 文件，请下载该文件并将其解压缩到您选择的目录中。样例应用程序位于 SolarisStudioSampleApplications 目录的 DLight 子目录中。

## 生成样例应用程序

此过程假定您已经下载了样例应用程序 zip 文件并将其解压缩到了您选择的目录中。

1. 将 ProfilingDemo 目录复制到您自己的专用工作区，例如您的起始目录：

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

2. 生成您的程序副本：

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

程序 profilingdemo 是使用 -g 选项生成的，该选项会生成调试信息。如果编译时未使用此选项，DLight 可能会收集并显示该程序的运行时数据，但用于显示函数源代码的功能将不起作用。

## 启动 DLight

通过键入以下内容启动 DLight：

```
% dLight
```

如果您未将 Oracle Solaris Studio 目录添加到您的路径，可以通过键入以下内容启动 DLight：

```
% /studio-installation-directory/bin/dLight
```

缺省安装目录是 /opt/solarisstudio12.3/bin。

## 分析样例应用程序

DLight 可执行目标允许 DLight 启动并分析尚未运行的应用程序。在本部分中，您将创建并运行可执行目标，然后在样例应用程序运行时观察 DLight 显示的动态图形。

1. 在 DLight 中，单击 "New DLight Target"（新建 DLight 目标）按钮 。"Create New Executable Target"（新建可执行目标）对话框将打开，其中 "Executable Target"（可执行目标）选项卡处于选定状态。
2. 在 "Create New Executable Target"（新建可执行目标）对话框中：
  - a. 在 "Run"（运行）字段中键入 profilingdemo 可执行文件的路径名，或者单击 "Browse"（浏览）以导航到 profilingdemo 可执行文件并将其打开。
  - b. 单击 "Run"（运行）。

新的可执行目标将保存并列在 "DLight Targets"（DLight 目标）窗口中，然后 DLight 运行该目标。

- c. 如果您没有对 DTrace 的足够特权（如第 3 页中的“设置 DLight 的 Oracle Solaris 特权”中所述），系统将提示您输入您的超级用户口令。如果您用适当管理员名称替换 root 用户名，还可以使用非 root 管理员用户的凭证。
3. profilingdemo 应用程序开始执行，且 "DLight Run Monitor"（DLight 运行监视器）窗口显示动态图形，其中包含所收集的正在运行的应用程序分析数据。

---

提示 – 如果看不到所有图形，请关闭 "Host Info"（主机信息）窗口。还可以使用 "DLight Run Monitor"（DLight 运行监视器）窗口边上的滚动条来查看更多图形。

---

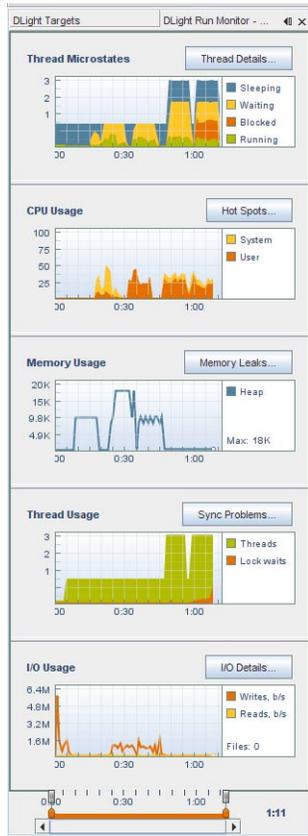
4. 在 "Output"（输出）窗口中，profilingdemo 程序会告知您正在执行的操作，这样您可以将输出与图形中显示的数据相匹配。每次程序请求该操作时请单击 "Output"（输出）窗口并按 Enter 键，直到完成执行。

## 使用滑块控件

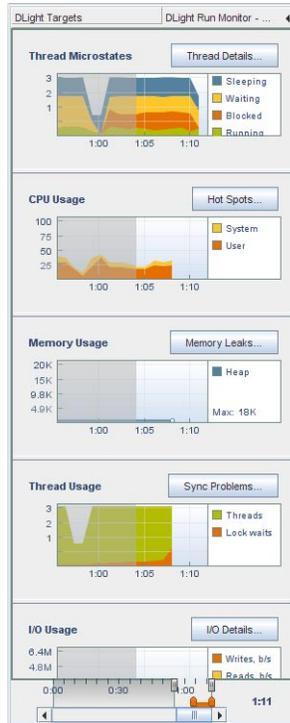
在 "DLight Run Monitor"（DLight 运行监视器）窗口的底部，可以看到用于控制图形视图的滑块："View"（视图）滑块、"Details"（详细信息）滑块和 "Time"（时间）滑块。



1. 将鼠标光标置于滑块端点的上方可获取有关每个滑块的工具提示信息。
2. 在 "Time"（时间）滑块上单击并按住鼠标光标，然后将滑块拖到左侧可看到运行的开始处。所有图形都会一起滑动，从而可以在给定时间看到每个方面（CPU、内存、线程、I/O）所发生的情况，并看到它们之间的关系。
3. 从左向右拖动 "Time"（时间）滑块可看到完整的运行。
4. 将鼠标光标移动到 "View"（视图）滑块，该控件覆盖了时间单元。使用 "View"（视图）滑块控件可以选择在图形中显示运行时的哪一部分。
5. 单击并拖动 "View"（视图）滑块的左手柄（起点），一直拖到运行的开始处 0:00。这些图形现在会立即显示整个运行。效果与尽可能缩小很相似。请注意，如果选择完整运行时，"Time"（时间）滑块将不起作用。您现在已经看到了所有数据，因此没有必要再进行滚动。



6. "View" (视图) 滑块可用于进行缩放。向右拖动"View" (视图) 滑块的起点。当拖动手柄时, 这些图形会放大, 以聚焦到靠近运行结束处的区域。请注意, 可以再次使用"Time" (时间) 滑块在运行时中来回滚动。
7. 将鼠标光标置于橙色"Details" (详细信息) 滑块的端点上方可获取关于如何使用该滑块的说明。使用"Details" (详细信息) 滑块控件可以选择运行时的某一部分进行详细检查。
8. 将"Details" (详细信息) 滑块的起点拖动到比"View" (视图) 滑块的起点稍晚的一个时间点。请注意, 在起点前面的区域中, 图形是灰显的, 从而为起点和终点之间的图形提供了一种突出显示效果。



- 单击图形的任意一个详细信息按钮 ("Thread Details" (线程详细信息)、"Hot Spots" (热点)、"Memory Leaks" (内存泄漏)、"Sync Problems" (同步问题) 或 "I/O Details" (I/O 详细信息)) 时, 详细信息选项卡中会显示突出显示区域的数据。换句话说, "Details" (详细信息) 滑块用于选择要显示在 "Details" (详细信息) 选项卡中的数据。
- 将 "View" (视图) 滑块的起点拖回到运行的开始处, 这样可以看到所有数据。

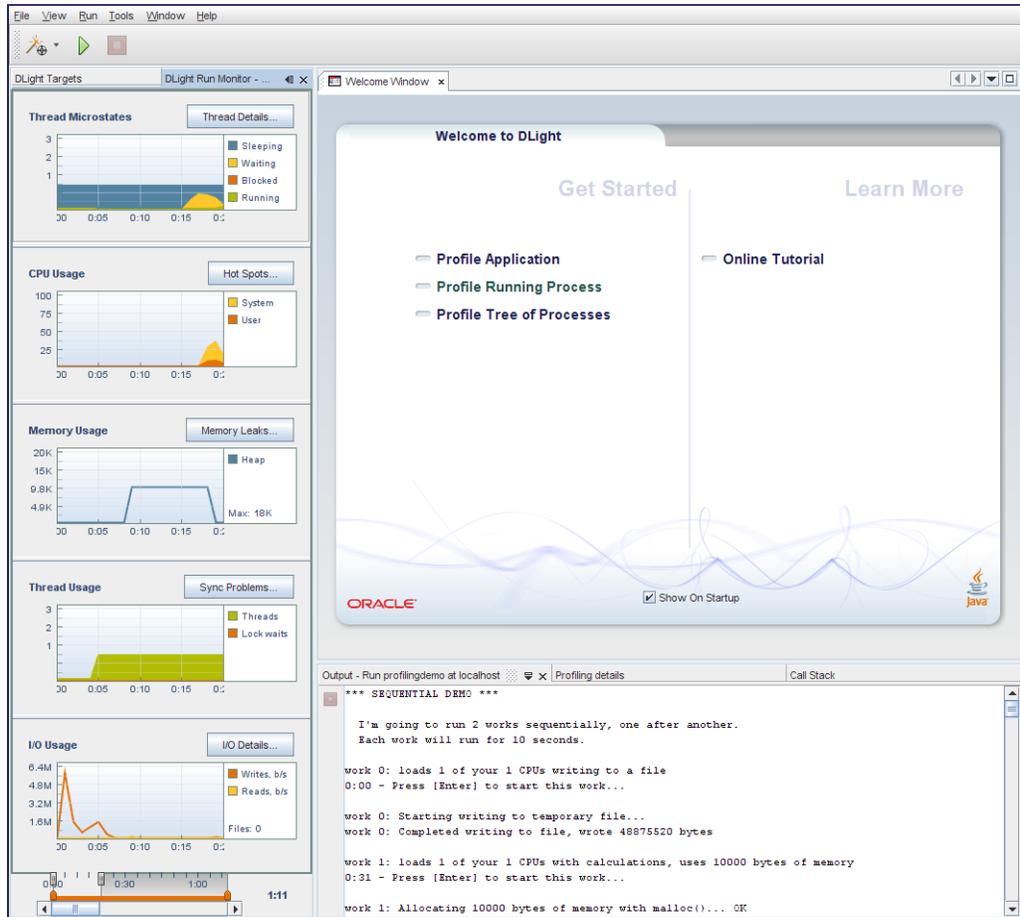
## 了解线程微状态

在程序运行期间, 随着程序线程进入各种执行状态, "Thread Microstates" (线程微状态) 图会显示程序线程的概览。Solaris 微状态计数功能使用 DTrace 工具来提供进入和退出十种不同的执行状态时有关每个线程的状态的细粒度信息:

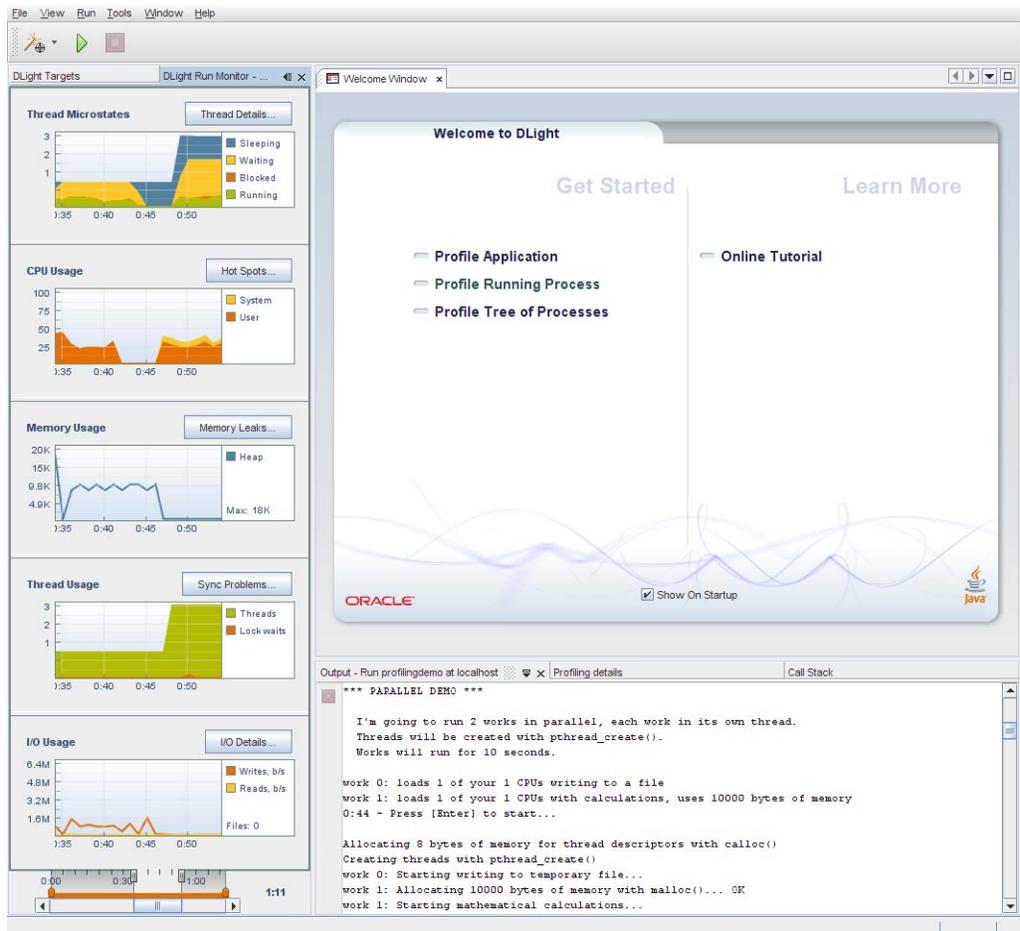
<b>User Running (用户运行)</b>	进程在用户模式下花费的时间百分比
<b>System Running (系统运行)</b>	进程在系统模式下花费的时间百分比
<b>Other Running (其他运行)</b>	进程处理系统陷阱等所花费的时间百分比
<b>Text page fault (文本缺页)</b>	进程处理文本缺页所花费的时间百分比
<b>Data page fault (数据缺页)</b>	进程处理数据缺页所花费的时间百分比
<b>Blocked (已阻塞)</b>	进程等待用户锁定所花费的时间百分比
<b>Sleeping (休眠)</b>	进程休眠所花费的时间百分比
<b>Waiting (等待)</b>	进程等待 CPU 所花费的时间百分比

"Thread Microstates" (线程微状态) 工具以图形方式显示在程序运行期间创建的所有线程的状态概括信息。仅显示四种状态: "Sleeping" (休眠)、"Waiting" (等待)、"Blocked" (已阻塞) 和 "Running" (运行)。这些状态提供了十个可能的微状态的简化或摘要视图, 并提供了程序中正在运行的所有线程的状态概述。例如, 在 "Running" (运行) 状态中花费的时间代表正在运行的状态的所有类型: 在用户模式下运行、在系统调用下运行、在缺页中运行以及在陷阱中运行。

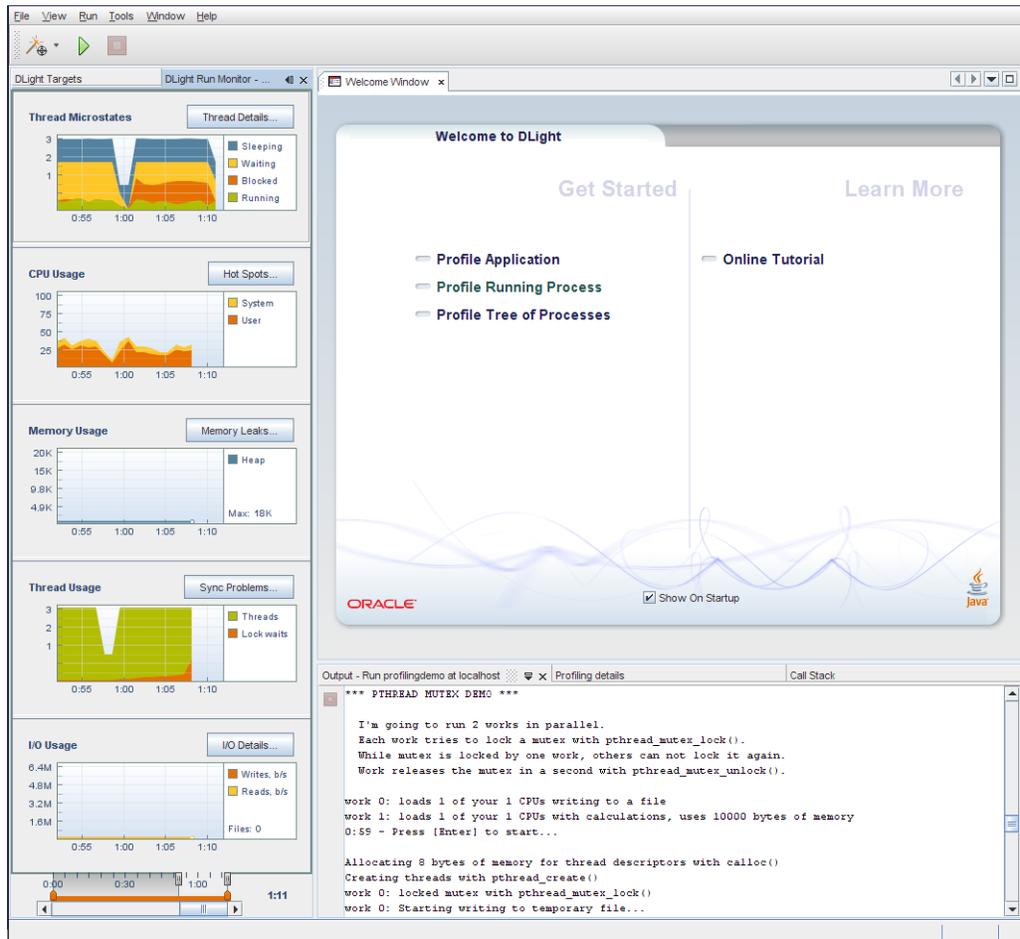
- 将 "View" (视图) 滑块的左手柄向左移动, 直到图形显示大约 20 秒的运行时, 如下所示。此图像显示了在单一线程中相继运行两个任务时 SEQUENTIAL DEMO 部分期间的程序运行的开始处。线程休眠所处的点与程序等待用户按 Enter 键所处的点相对应。



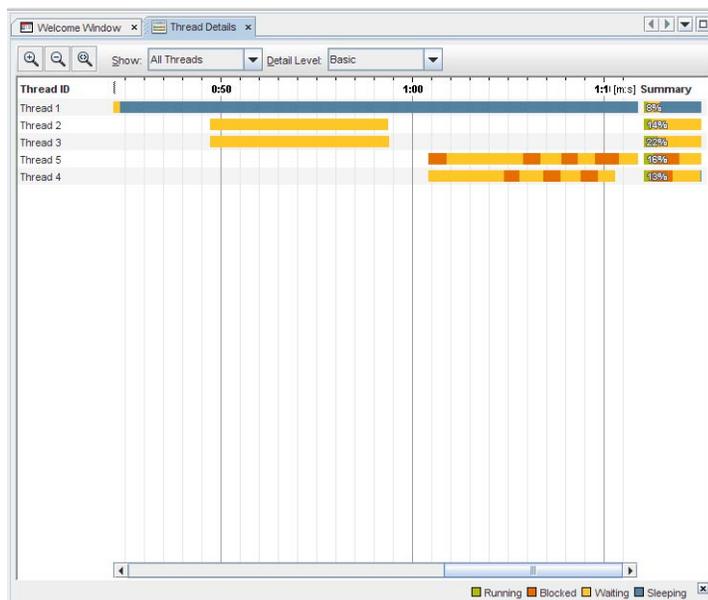
2. 单击 "Time" (时间) 滑块并一直向右拖动, 直到在 "Thread Microstates" (线程微状态) 工具中显示的线程数跳到三为止。
3. 当程序进入 PARALLEL DEMO 部分时, 线程数会跳到三。主线程会启动两个其他线程以并行方式运行两个任务, 每个任务都在各自的线程中运行。您可以看到在 "Waiting" (等待) 状态 (黄色) 和 "Sleeping" (休眠) 状态 (蓝色) 下花费了相当多的时间, 而在 "Running" (运行) 状态 (绿色) 下花费的时间则要少一些。在 PARALLEL DEMO 部分期间, 不会在 "Blocked" (已阻塞) 状态 (橙色) 下花费任何时间, 因为程序的这一部分不会执行任何线程同步策略, 例如会阻塞线程的互斥锁。



- 将 "Time" (时间) 滑块一直拖到运行时的结束处。请注意橙色的 "Blocked" (已阻塞) 微状态显示在程序进入 PTHREAD MUTEX DEMO 部分的点处, 在该部分中, 每个线程都使用独占锁来防止其他线程在某些点处受到干扰。每个线程仅在获取互斥锁之后才可以主动运行。如果某个线程拥有互斥锁, 则其他线程在尝试访问代码的锁定部分时会被阻止。当线程对同一数据进行重叠访问时, 使用互斥锁可防止线程出现数据争用情况。

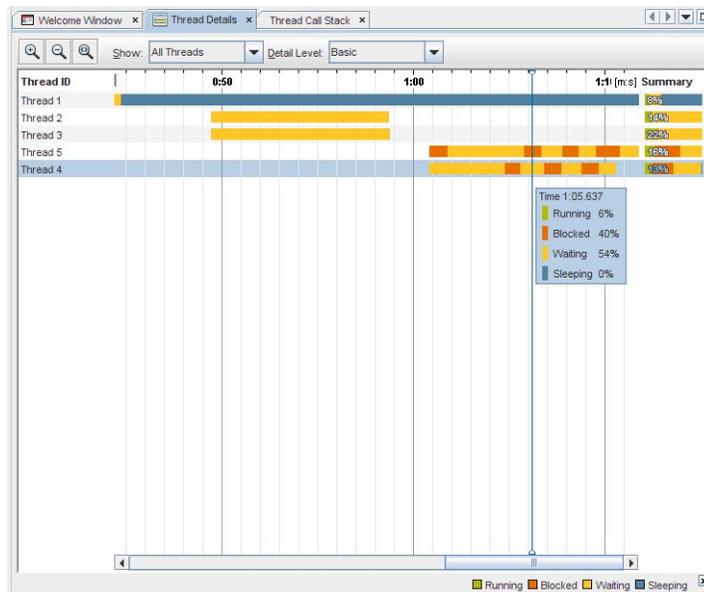


- 单击 "Thread Details" (线程详细信息) 按钮以显示有关线程微状态的详细信息。"Thread Details" (线程详细信息) 窗口将打开，显示在程序中运行的所有线程的图形时间线表示以及详细的状态信息。

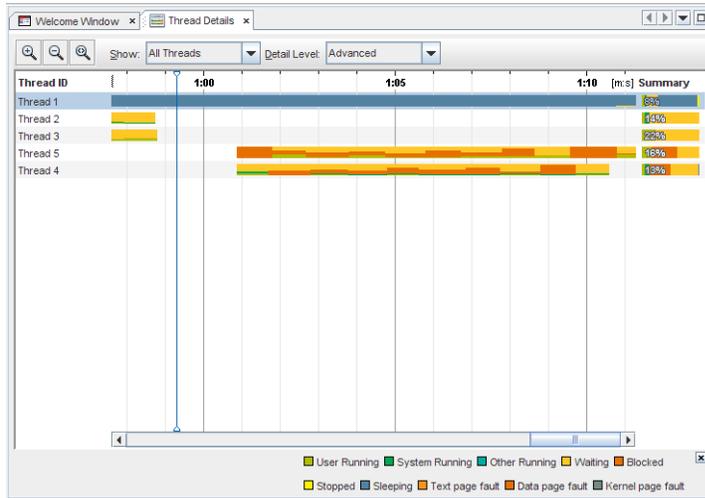


"Thread Details" (线程详细信息) 窗口显示在程序的完整运行时期每个线程的状态转换。

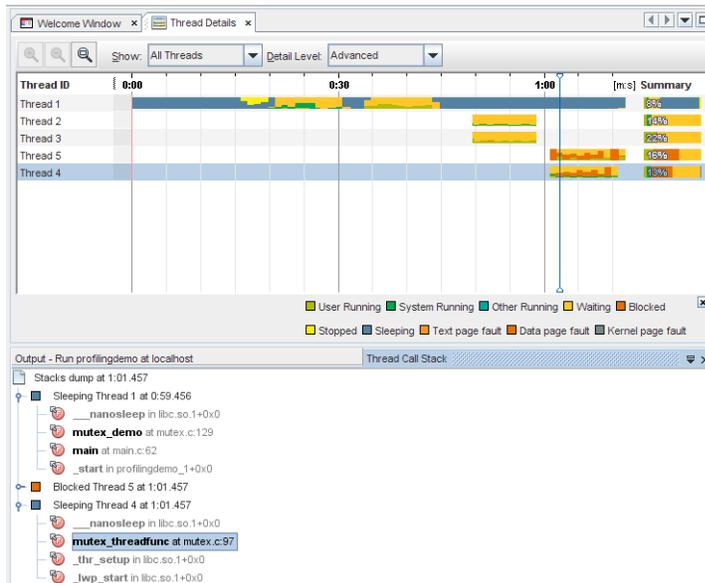
- 将光标放置到线程的其中一个颜色区域上。将出现一个弹出窗口，显示有关在该特定时刻该线程将要发生的变化详细信息。"Details" (详细信息) 包括数据的收集时间以及该时刻在每个线程状态下花费的时间百分比。将光标放置到窗口右侧的 "Summary" (摘要) 区域上方时，将出现一个弹出窗口，显示线程的完整运行在每个状态下的百分比。



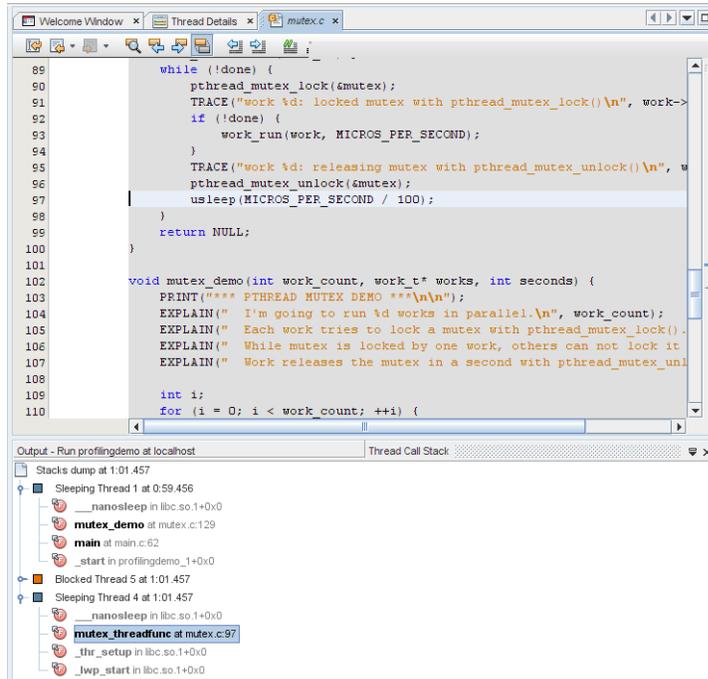
- 尝试使用窗口的控件来更改显示的内容：
  - 缺省情况下，该窗口会显示所有线程。单击 "Show" (显示) 下拉式列表右侧的向下箭头。您可以选择 "Live Threads only" (仅活动线程) 仅显示尚未终止的线程，或者选择 "Finished Threads only" (仅完成的线程) 仅显示在程序运行期间已终止的线程。
  - 缺省情况下，该窗口仅显示四种概括的执行状态。单击 "Detail Level" (详细信息级别) 下拉式列表右侧的向下箭头。您可以选择 "Moderate" (中等) 显示有关这些状态的更多详细信息，或者选择 "Advanced" (高级) 显示十种微状态。
  - 单击某个单独的线程，请注意该线程将突出显示。如果按住 Shift 键单击另一个线程，会选中某个范围的线程。要选择不相邻的多个线程，请在选择线程之前按 Ctrl 键。如果感兴趣的线程已突出显示，请右键单击并选择 "Show Only Selected Threads" (仅显示选定线程)。要再次看到所有线程，请从 "Show" (显示) 下拉式列表中选择 "All Threads" (所有线程)。
  - 右键单击任何线程，然后选择 "Thread Name" (线程名称) ⇒ "Thread Entry" (线程入口)，将显示的线程名称更改为线程开始执行时进入的函数。
- 通过单击 "Zoom In" (放大) 按钮 ，放大线程图形以便更仔细地查看。此图像显示了放大后的窗口，且 "Detail Level" (详细信息级别) 设置为 "Advanced" (高级)。



9. 单击 "Zoom Out" (缩小) 按钮  可返回到前一个缩放级别。
10. 单击 "Show Complete Run" (显示完整运行) 按钮  会立即在 "Thread Details" (线程详细信息) 窗口中显示完整运行。可以再次单击按钮  返回到线程详细信息的滚动视图。
11. 单击线程 4 上的第二个橙色矩形。"Thread Call Stack" (线程调用堆栈) 选项卡将打开, 显示线程在此刻的调用堆栈。您可以展开该堆栈中的节点查看该线程中发生的调用, 或者右键单击顶部节点并选择 "Expand All" (全部展开) 查看所有线程中的调用。



12. 双击 mutex\_threadfunc 函数可打开调用该函数的源文件。(您可以在未灰显的堆栈中显示任意函数的调用源文件。)



13. 单击"Thread Details" (线程详细信息) 选项卡返回到"Thread Details" (线程详细信息) 窗口。单击某个线程。您可以使用鼠标或键盘沿着线程的时间线导航。
  - 使用鼠标时，右键单击线程并使用"Navigate" (导航) 菜单向左或向右移动焦点，在时间线上确定一个点并更新"Thread Call Stack" (线程调用堆栈) 选项卡的内容，或者将焦点切换到"Thread Call Stack" (线程调用堆栈) 选项卡。
  - 要使用快捷键导航线程，请按下列键：
    - Ctrl + 向左方向键和 Ctrl + 向右方向键，用于在线程时间线上向左和向右滚动
    - Ctrl + 向下方向键，用于选择时间线上的某个点，此操作将更新"Thread Call Stack" (线程调用堆栈)
    - Alt + 向下方向键，用于在"Thread Call Stack" (线程调用堆栈) 窗口上定位输入焦点
  - 在"Thread Call Stack" (线程调用堆栈) 窗口中，可以使用方向键和 Enter 键打开与函数相关联的源文件

## 了解 CPU 使用情况

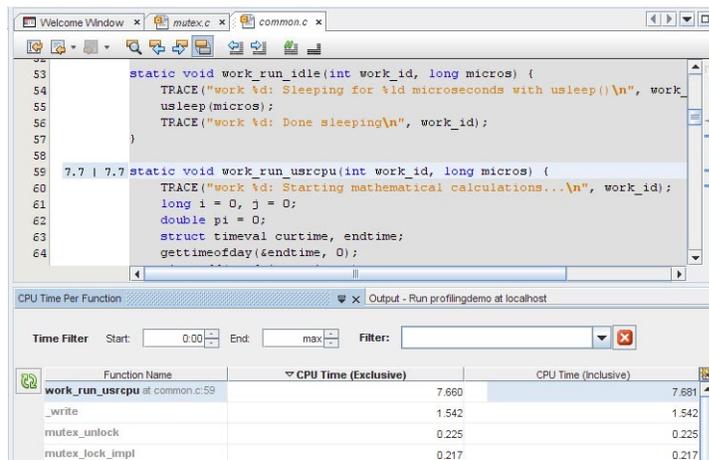
"CPU Usage" (CPU 使用情况) 图显示了应用程序在其运行期间所使用的 CPU 总时间的百分比。

1. 单击"Hot Spots" (热点) 按钮可显示有关 CPU 时间的详细信息。"CPU Time Per Function" (每个函数的 CPU 时间) 选项卡会打开，显示程序的各个函数以及每个函数所使用的 CPU 时间。这些函数按所使用的 CPU 时间的顺序列出，使用最长时间的函数列在第一个。如果程序仍在运行，最先显示的时间就是在单击按钮那一刻所消耗的时间量。

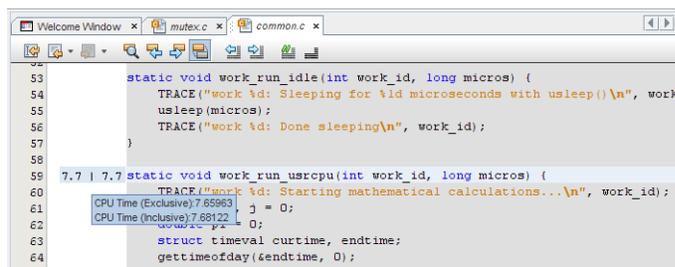
Function Name	CPU Time (Exclusive)	CPU Time (Inclusive)
work_run_usrcpu at common.c:59	7.660	7.681
_write	1.542	1.542
mutex_unlock	0.225	0.225
mutex_lock_impl	0.217	0.217
memcpy	0.053	0.053
gettimeofday	0.027	0.027
_fwrite_unlocked	0.023	0.072
malloc	0.019	0.019

2. 单击"Function Name" (函数名称) 列的标题可按字母顺序对函数进行排序。

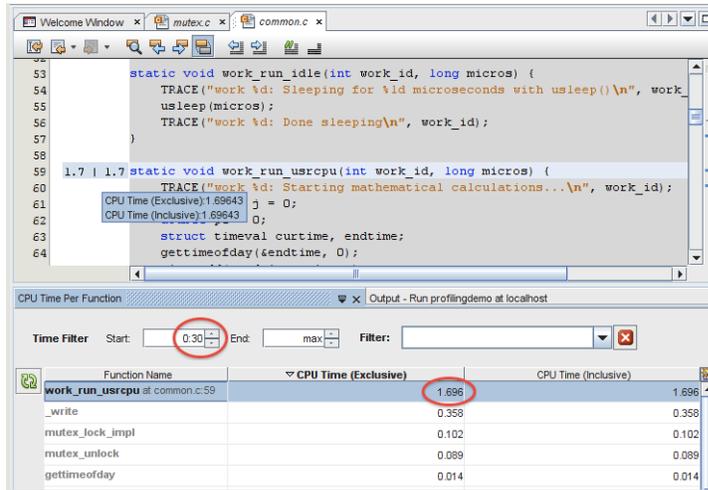
- 单击 "CPU Time (Exclusive)" (CPU 时间 (独占)) 列可按各个函数所使用时间的顺序对函数进行排序。
- 请注意两列 "CPU Time" (CPU 时间) 之间的差异。"CPU Time (Inclusive)" (CPU 时间 (包含)) 显示从进入函数的时间起直到函数退出的时间为止所花费的 CPU 总时间, 包括所列函数调用的所有其他函数的时间。"CPU Time (Exclusive)" (CPU 时间 (独占)) 仅显示特定函数所使用的时间, 而不包括它所调用的任何函数。
- 单击 "CPU Time (Inclusive)" (CPU 时间 (包含)) 列标题可将最消耗时间的函数放回到顶部。请注意, `work_run_usrcpu` 函数使用的 "CPU Time (Inclusive)" (CPU 时间 (包含)) 仅略多于 "CPU Time (Exclusive)" (CPU 时间 (独占)), 这意味着其 CPU 时间中实际上只有很少一部分是由它所调用的其他函数使用的, 而 `work_run_usrcpu` 函数本身使用了大部分时间。
- 某些函数以粗体显示。您可以显示这些函数的源文件。双击 `work_run_usrcpu` 函数。 `common.c` 文件将打开, 光标停留在第 59 行的 `work_run_usrcpu` 函数上。



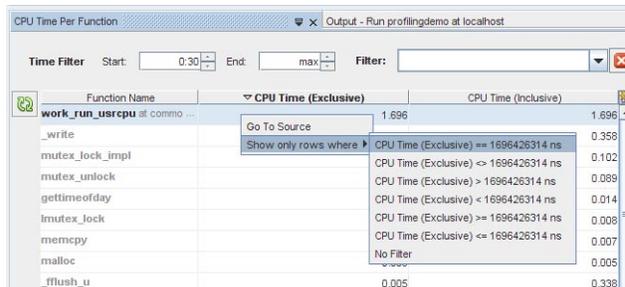
- 将鼠标光标放置到左边界中的数字上方。这些数字与 "CPU Time Per Function" (每个函数的 CPU 时间) 选项卡中所显示的函数的 CPU 独占时间和 CPU 包含时间的度量相同。这些度量为了节省空间而进行了四舍五入, 但将光标置于其上方时会显示未舍入的值。 `common.c` 源文件中也会显示 CPU 消耗行的度量 (例如在 `work_run_usrcpu` 函数内执行计算的 for 循环)。



- 通过键入时间并按 Enter 键, 或者使用箭头滚动选择秒数, 将 "CPU Time Per Function" (每个函数的 CPU 时间) 选项卡中的时间过滤器开始时间更改为 0:30。"DLight Run Monitor" (DLight 运行监视器) 窗口中的图形将随之变化, 如同移动 "Details" (详细信息) 滑块上的手柄时一样。如果拖动手柄, 将更新 "CPU Time Per Function" (每个函数的 CPU 时间) 选项卡中的 "Time Filter" (时间过滤器) 设置以进行匹配。更为重要的是, 为该选项卡中的函数显示的数据会进行更新以反映过滤器, 因此仅会显示在该时间段内使用的 CPU 时间。



- 您还可以过滤符合某个特定度量的数据。右键单击 `work_run_usrcpu` 的 "CPU Time (Exclusive)" (CPU 时间 (独占)) 度量。选择 "Show only rows where" (仅选择符合以下条件的行) `>` "CPU Time (Exclusive) == the metric shown for work\_run\_usrcpu" (CPU 时间 (独占) == `work_run_usrcpu` 显示的度量)。所有其他行都会过滤掉，仅显示 CPU 独占时间等于该度量的行。

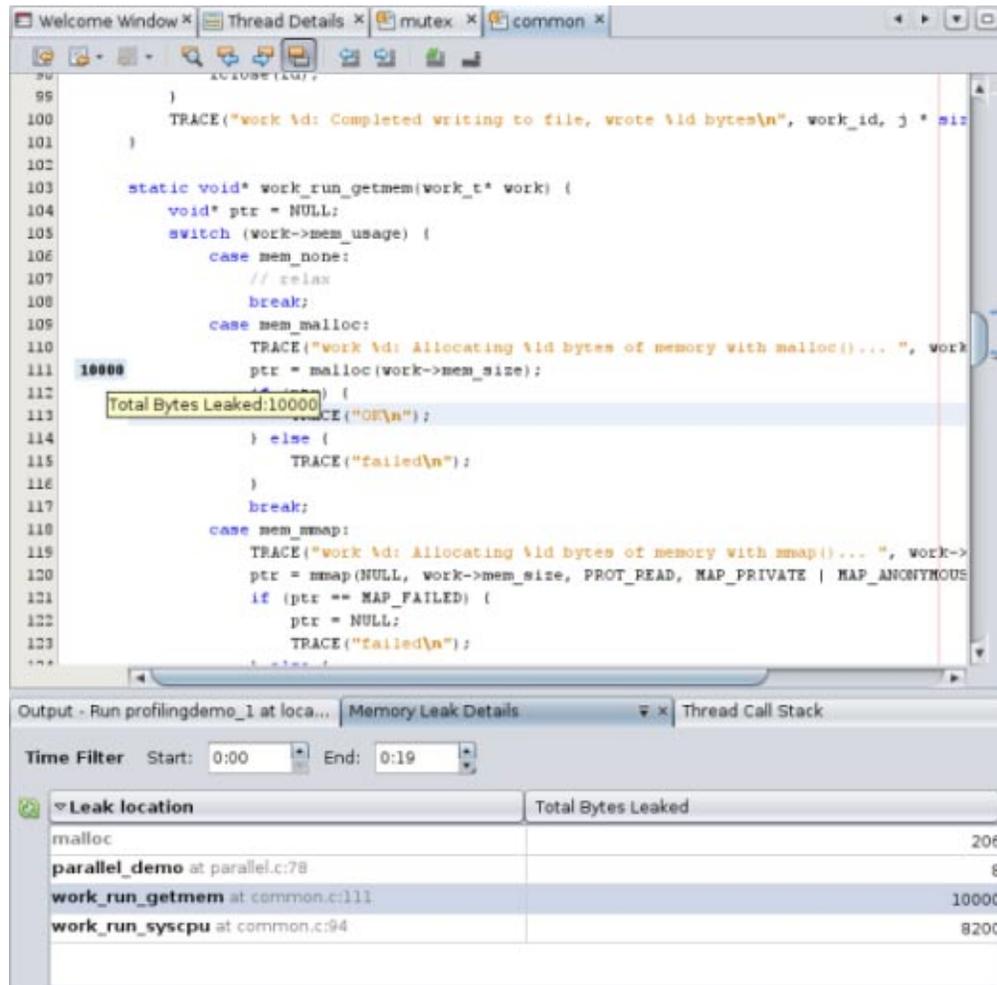


## 了解内存使用情况

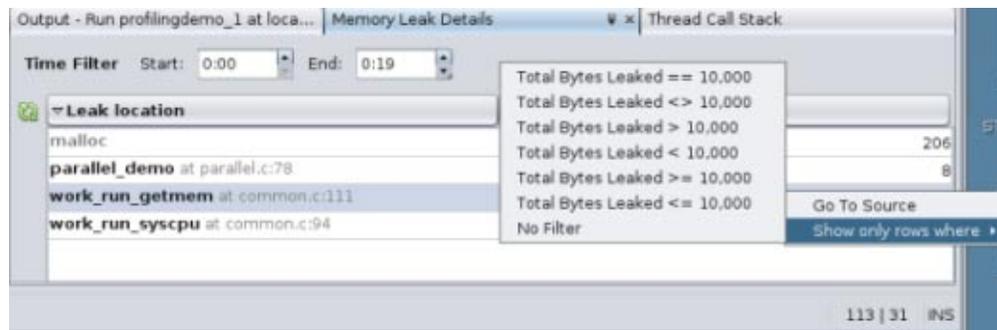
"Memory Usage" (内存使用情况) 工具显示程序的内存堆是如何随其运行时改变的。您可以使用该工具来识别内存泄漏，内存泄漏是指程序中不再需要的内存无法释放的点。内存泄漏可能会导致程序中的内存消耗增加。如果某个存在内存泄漏的程序运行足够长的时间，它最终可能会用尽可用内存。

- 在 "DLight Run Monitor" (DLight 运行监视器) 中向左和向右滑动 "Time" (时间) 滑块可查看内存堆是如何随时间而增加和减少的。在程序的运行中有四个使用量高峰。前两个发生在 SEQUENTIAL DEMO 期间，第三个发生在 PARALLEL DEMO 期间，第四个发生在 PTHREAD MUTEX DEMO 期间。
- 单击 "Memory Leaks" (内存泄漏) 按钮可显示 "Memory Leak Details" (内存泄漏详细信息) 选项卡，该选项卡显示有关哪些函数表现出内存泄漏的详细信息。该选项卡中仅会列出产生内存泄漏的函数。如果单击该按钮时程序正在运行，显示的泄漏位置就是单击按钮那一刻存在的位置。随着时间的推移，可能会出现更多泄漏，因此请单击 "Refresh" (刷新) 按钮  更新列表。如果直到运行结束仍未检测到任何内存泄漏，"Memory Leak Details" (内存泄漏详细信息) 选项卡会指示未找到内存泄漏。
- 您可以通过更改 "Memory Leak Details" (内存泄漏详细信息) 选项卡中的开始时间和结束时间或者通过使用 "DLight Run Monitor" (DLight 运行监视器) 窗口中的 "Details" (详细信息) 滑块来过滤数据。
- 在此运行中，ProfilingDemo 程序显示了一个与 `work_run_getmem` 函数相关联的内存泄漏。双击 `work_run_getmem` 函数，`common.c` 文件将打开，且光标位于函数中发生内存泄漏的行上。

- 在左边界中会显示内存泄漏度量。如同"CPU Usage"（CPU使用情况）度量一样，将鼠标移动到内存泄漏度量上也会显示详细信息。



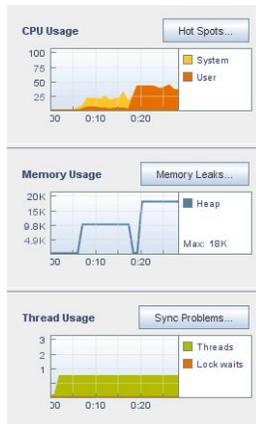
- 与"CPU Time Per Function"（每个函数的CPU时间）选项卡一样，您可以在"Memory Leak Details"（内存泄漏详细信息）选项卡中右键单击度量，然后选择一个用于过滤数据的标准。



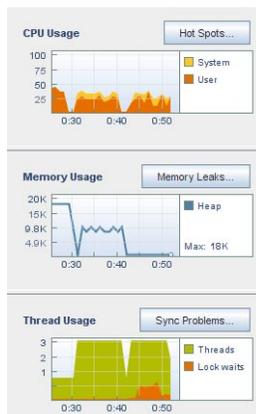
## 了解线程使用情况

"Thread Usage"（线程使用情况）工具显示程序所使用的线程数量，以及线程为了继续其任务必须等待获取锁定的所有时刻。此数据对于多线程应用程序很有用，这些应用程序为了避免昂贵的等待时间必须执行线程同步。

1. 将 "Time" (时间) 滑块滑动到运行的开始处, 并注意 (如同在 "Thread Microstates" (线程微状态) 图中一样), 在程序的 SEQUENTIAL DEMO 部分期间线程数量为一, 但随着程序进入 PARALLEL DEMO 部分, 线程数量会增加到三。
2. 移动 "View" (视图) 滑块的端点手柄, 这样您便可以看到从运行的开始处到另外两个线程启动之前的数据。
3. 查看同一时间段的 "CPU Usage" (CPU 使用情况) 图和 "Memory Usage" (内存使用情况) 图, 并注意是单个线程正在执行某个使用 CPU 时间和内存的活动。此时间段与程序的 SEQUENTIAL DEMO 部分相对应, 在该部分中主线程写入到文件, 然后按顺序执行某些计算。在程序等待用户按 Enter 键时, CPU 和内存使用量都会降低, 而线程数仍然是一个。

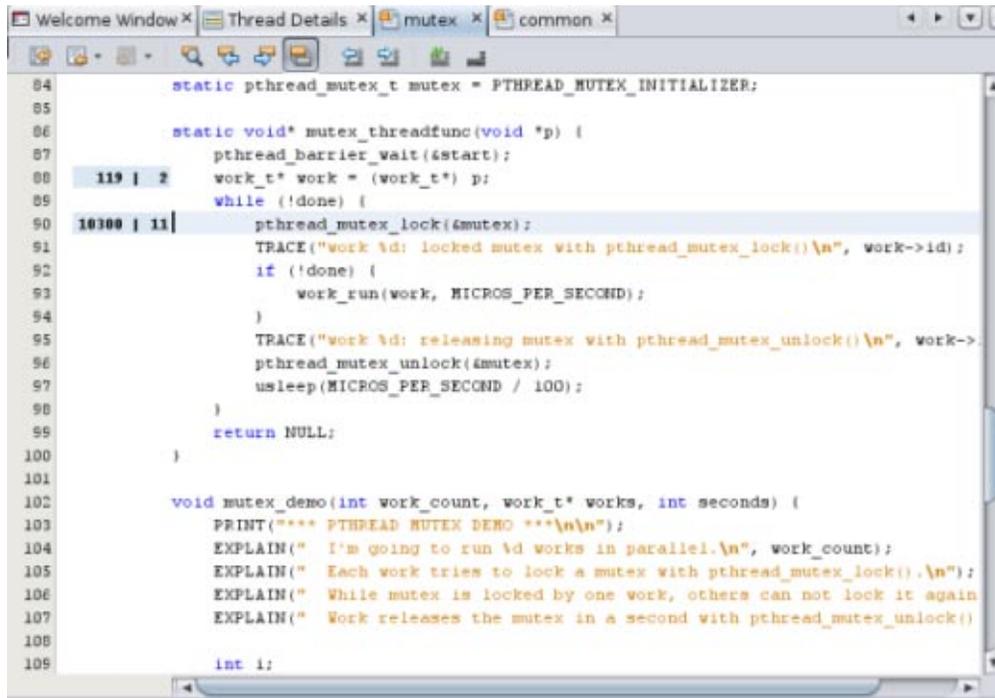


4. 将 "Time" (时间) 滑块向右滑动, 这样您便可以看到线程数增加到三时的两个点。线程数的第一次增加与程序运行的 PARALLEL DEMO 部分相对应, 其中主线程启动两个其他线程来并行执行写入文件和执行计算的工作。请注意, 内存使用量和 CPU 使用量在此部分会有点高, 但是完成两个任务的时间要比在 SEQUENTIAL DEMO 部分少得多。



5. 请注意在 PARALLEL DEMO 线程完成之后线程数又恢复为一个, 而主线程会等待用户按 Enter 键。
6. 在程序的 PTHREAD MUTEX DEMO 部分运行时, 线程计数会再次增加到三个。请注意, 在线程计数增加到三个之后不久, 会出现一个锁定等待, 以橙色显示。PTHREAD MUTEX DEMO 使用互斥锁来防止多个线程对某些函数进行重叠访问, 这导致线程需要等待获取锁定。
7. 单击 "Sync Problems" (同步问题) 按钮可显示有关线程锁定的详细信息。"Thread Synchronization Details" (线程同步详细信息) 选项卡将打开, 并列出了需要等待获取互斥锁的函数。同时还显示了函数等待所花费的毫秒数以及函数必须等待某个锁定的次数的度量。

8. 如果在程序正在运行时单击了 "Sync Problems" (同步问题) 按钮, 可能需要单击 "Refresh" (刷新) 按钮  使用最新的线程锁定更新显示。
9. 单击 "Wait Time" (等待时间) 列的标题可按等待所花费的时间的顺序对函数进行排序。
10. 单击 "Lock Waits" (锁定等待) 列的标题可按线程在函数中等待的次数对函数进行排序。
11. 双击 `mutex_threadfunc` 函数, 该函数的锁定等待次数最多。 `mutex.c` 源文件将打开, 且光标位于调用 `pthread_mutex_lock` 函数的行上。此函数负责在读取或写入内存位置之前锁定该内存位置, 而且必须等待, 直到没有任何其他线程在该内存位置上有锁定为止。



```

84     static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
85
86     static void* mutex_threadfunc(void *p) {
87         pthread_barrier_wait(&start);
88         work_t* work = (work_t*) p;
89         while (!done) {
90             pthread_mutex_lock(&mutex);
91             TRACE("work %d: locked mutex with pthread_mutex_lock()\n", work->id);
92             if (!done) {
93                 work_run(work, MICROS_PER_SECOND);
94             }
95             TRACE("work %d: releasing mutex with pthread_mutex_unlock()\n", work->
96                 pthread_mutex_unlock(&mutex);
97                 usleep(MICROS_PER_SECOND / 100);
98             }
99             return NULL;
100         }
101     }
102
103     void mutex_demo(int work_count, work_t* works, int seconds) {
104         PRINT("**** PTHREAD MUTEX DEMO ****\n\n");
105         EXPLAIN(" I'm going to run %d works in parallel.\n", work_count);
106         EXPLAIN(" Each work tries to lock a mutex with pthread_mutex_lock().\n");
107         EXPLAIN(" While mutex is locked by one work, others can not lock it again\n");
108         EXPLAIN(" Work releases the mutex in a second with pthread_mutex_unlock()\n");
109
110         int i;

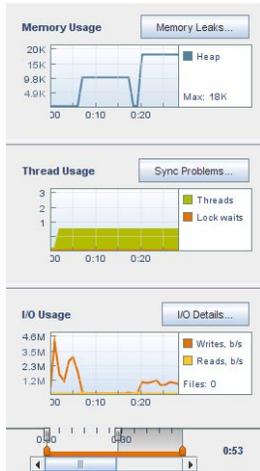
```

12. 在源文件的左列边界中显示了 "Wait Time" (等待时间) 和 "Lock Waits" (锁定等待) 的度量。将鼠标光标放置到这些度量的上方可查看详细信息, 这些详细信息与 "Thread Synchronization Details" (线程同步详细信息) 选项卡中显示的内容相符。
13. 右键单击这些度量, 并取消选定 "Show Profiler Metrics" (显示分析器度量)。这些度量将不会再显示在任何配置工具的程序编辑器中。
14. 选择 "View" (视图) ⇒ "Show Profiler Metrics" (显示分析器度量) 可再次显示这些度量。

## 了解 I/O 使用情况

"I/O Usage" (I/O 使用情况) 工具显示运行期间程序的读取和写入活动的概述。

1. 下面的图像显示了在 SEQUENTIAL DEMO 部分期间 (在此期间两个任务在单一线程中相继运行) 运行开始处的 I/O 使用情况。在前几秒钟, 程序启动, 然后等待用户按 Enter 键。用户按 Enter 键时, 程序会将字符写入到某个临时文件中。此活动在该图形中反映为显示写入的字节数的橙色线。



2. 请注意以下事项：

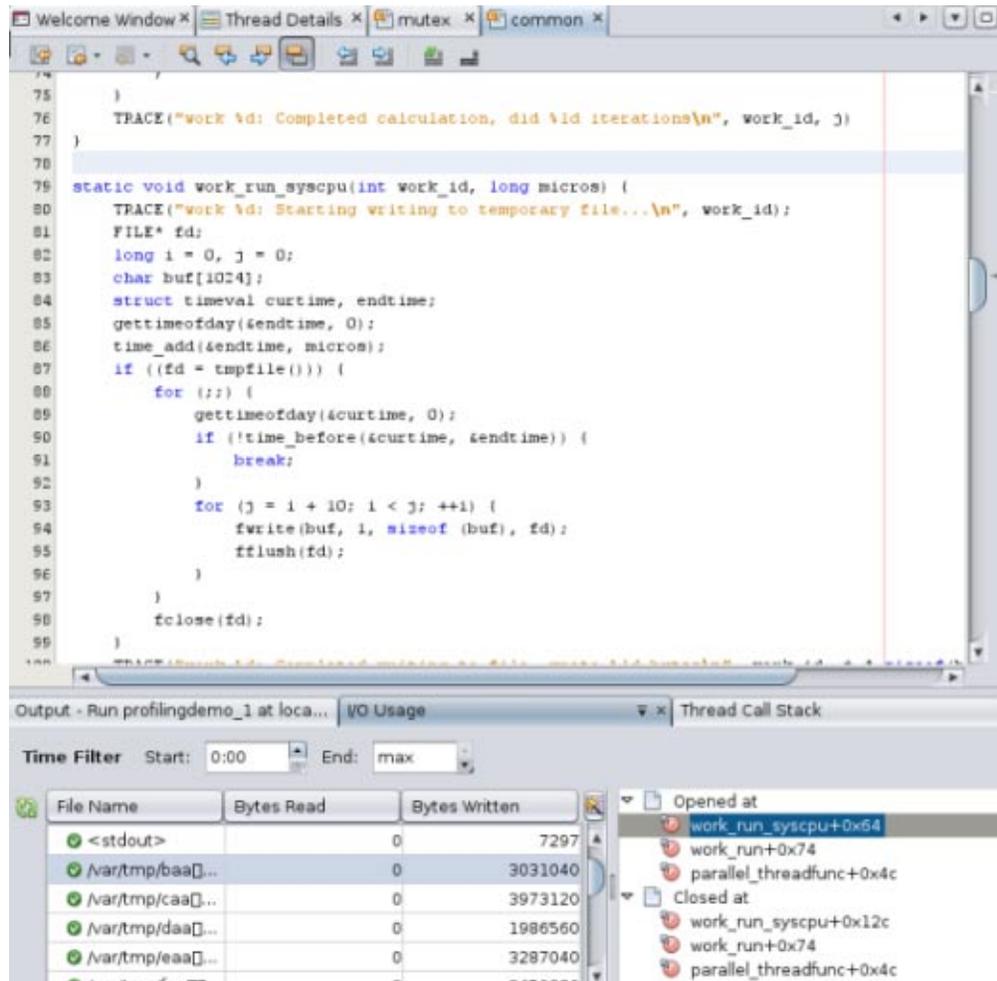
- 橙色线显示了最后一秒写入的字节数。"Profiling Demo"（分析演示）程序自身会在 "Output"（输出）窗口中报告：在 SEQUENTIAL DEMO 的这次运行期间，共写入 7998464 个字节（大约 76.3M）。如果将橙色线上显示的所有数据点加起来，总数会接近 76.3M。
- 在此阶段，"CPU Usage"（CPU 使用情况）工具中显示的系统时间值很高，因为程序利用系统调用来生成数据并将其写入到磁盘。
- "Memory Usage"（内存使用情况）工具显示了一个已分配的稳定的 8K 字节内存堆。

3. 单击 "I/O Details"（I/O 详细信息）按钮。"I/O Usage"（I/O 使用情况）详细信息选项卡将打开，并显示标准输入、标准输出以及程序读取自和写入到的临时文件。具有复选标记的文件已经关闭。标有黄色图标文件仍处于打开状态，以供执行读取和写入操作。

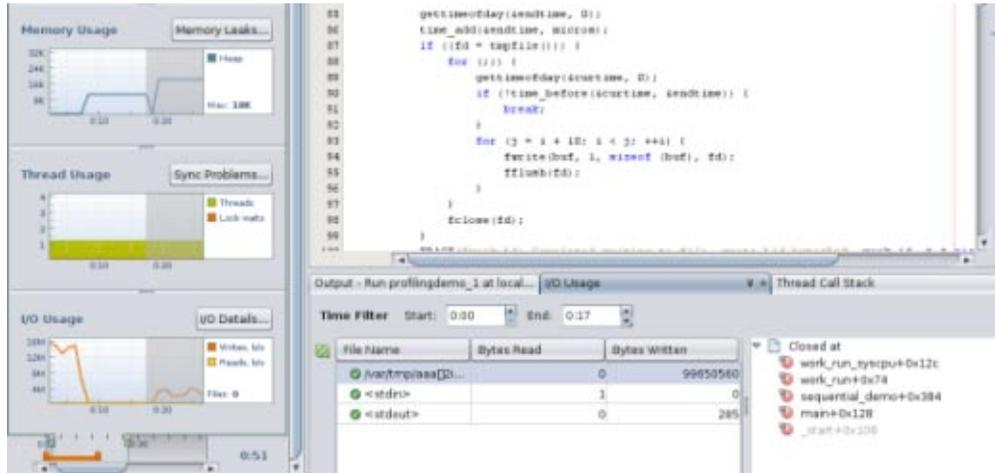
The screenshot shows the 'I/O Usage' window with a table of file names, bytes read, and bytes written. The table has columns for 'File Name', 'Bytes Read', and 'Bytes Written'. The data is as follows:

File Name	Bytes Read	Bytes Written
/var/tmp/aaa[...]	0	99650560
<stdin>	3	0
<stdout>	0	7297
/var/tmp/baa[...]	0	3031040
/var/tmp/caa[...]	0	3973120

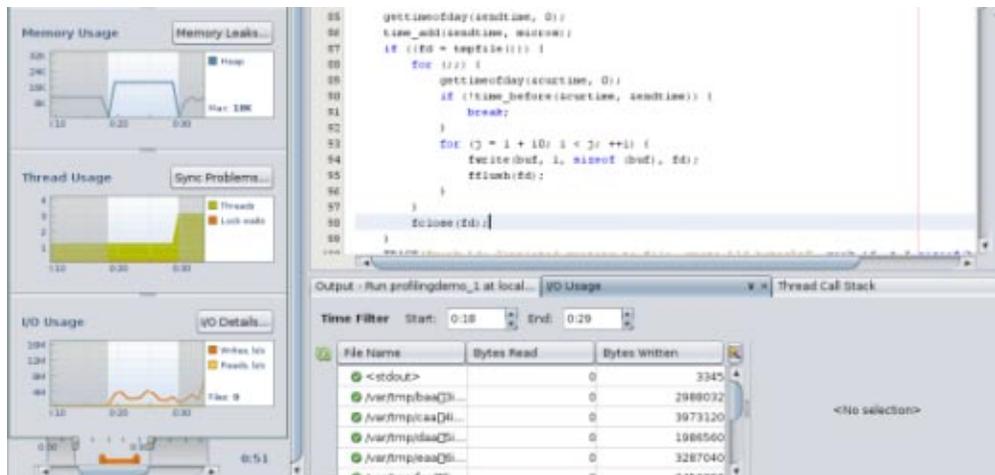
4. 请注意，此程序只需要偶尔按 Enter 键进行输入，因此 "Bytes Read"（读取的字节数）列不是很有用。您可以通过单击列标题右侧的 "Change Visible Columns"（更改可视列）按钮关闭 "Bytes Read"（读取的字节数）列。在 "Change Visible Columns"（更改可视列）对话框中，单击该复选框以取消选定 "Bytes Read"（读取的字节数），然后单击 "OK"（确定）。
5. 假定您需要有关此程序如何使用所有这些临时文件的更多详细信息。单击 "I/O Usage"（I/O 使用情况）详细信息选项卡中的 /var/tmp/baa[...] 文件可查看打开和关闭该文件的函数。会在面板中文件列表的右侧列出这些函数。
6. 双击函数列表中的 work\_run\_syscpu 函数可打开该函数的源文件。



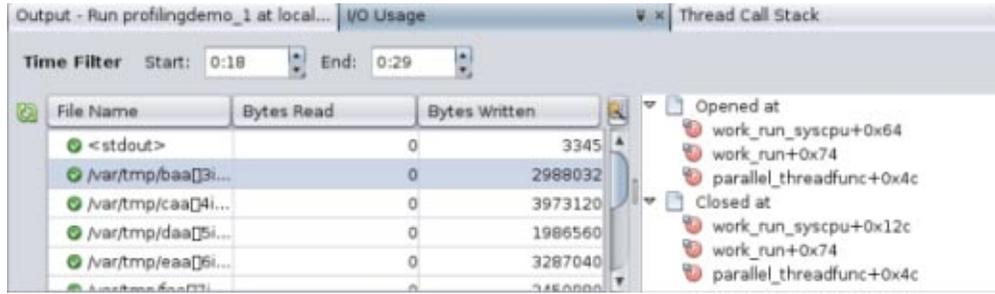
- 与 "CPU Time Per Function" (每个函数的 CPU 时间) 选项卡和 "Thread Synchronization Details" (线程同步详细信息) 选项卡中一样, 在 "I/O Usage" (I/O 使用情况) 详细信息选项卡中, 您可以指定一个查看时间间隔。在 "DLight Run Monitor" (DLight 运行监视器) 窗口中的 "I/O Usage" (I/O 使用情况) 图中, 写入活动在离运行的开始处 (此时程序的 SEQUENTIAL DEMO 部分开始写入到某个临时文件) 非常近的位置启动。
- 通过在 "End" (结束) 字段中键入运行结束的时间 (在下图中为 0:17), 分离运行的 SEQUENTIAL DEMO 部分。数据是在 "DLight Run Monitor" (DLight 运行监视器) 窗口和 "I/O Usage" (I/O 使用情况) 详细信息选项卡中过滤的, 因此您可以将焦点置于 SEQUENTIAL DEMO 阶段的输入和输出上。



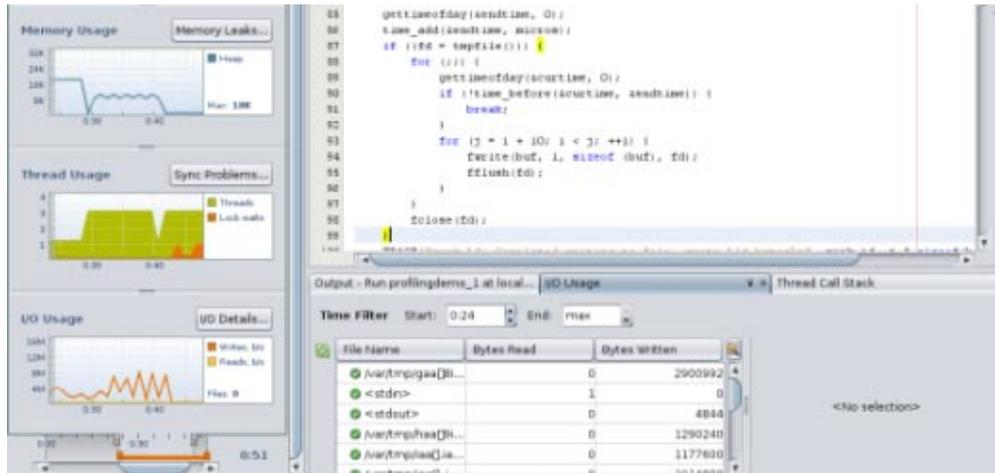
9. 请注意，在顺序演示期间，“I/O Usage”（I/O 使用情况）详细信息选项卡中会显示一个临时文件。一个单一线程将打开该文件，写入到该文件，然后关闭该文件。单击该文件可查看访问该文件的函数，然后双击某个函数可查看其源代码。
10. 在“DLight Run Monitor”（DLight 运行监视器）窗口中，向右拖动“Time”（时间）滑块，直到看到写入活动在暂停之后再次开始。写入中的暂停发生在顺序演示的第二个任务期间，该任务是一个计算任务，在该任务期间不会发生任何磁盘写入。再次开始的写入活动显示程序正在进入 PARALLEL DEMO 部分，在该部分中用户按 Enter 键以启动任务，而且在单独的线程中同时向磁盘和计算写入。
11. 通过键入并行演示活动在“Start”（开始）字段中开始的时间和在“End”（结束）字段中结束的时间，可分离并行演示活动。对于此图像中显示的运行，开始时间是 0:18，结束时间是 0:29。



12. 请注意，几个临时文件会在运行的 PARALLEL DEMO 部分期间显示在“I/O Usage”（I/O 使用情况）详细信息选项卡中。只有一个线程会写入到这些文件，因为每一秒钟它都会切换到一个新文件。正在计算的任务不会写入到磁盘。
13. 单击其中一个文件，会看到 parallel\_threadfunc 函数正在打开和关闭文件。



- 单击 "Details" (详细信息) 滑块的右手柄并将其拖动到运行的结束处。在下图中, 您可以看到, 当程序进入 PTHREAD MUTEX DEMO 部分且用户按 Enter 键之后, I/O 活动会再次发生改变。通过将并行演示的开始时间更改为结束时间 (在此示例中为 0:24), 为运行的此部分过滤数据。



- "I/O Usage" (I/O 使用情况) 详细信息选项卡显示, 在运行的 PTHREAD MUTEX DEMO 部分期间有多个文件处于打开状态。一个线程正在写入到某个文件, 且每一秒钟它都会切换它写入到的文件, 如同在 PARALLEL DEMO 部分中一样。但是, 由于使用了互斥锁, 有时写入线程会被计算线程阻止, 无法继续写入文件。

这样就结束了可执行目标的 "DLight Run Monitor" (DLight 运行监视器) 窗口工具的演示。这些工具也适用于进程目标。

## 分析进程树

运行进程树目标时, 将 DLight 连接到正在运行的进程。DLight 会分析指定的进程及其产生或派生的所有子进程。通过分析运行的进程及其所有子进程, 可以帮助您检测在进程之间以及某个进程的线程间发生的问题。

如果您希望捕获进程启动时的活动, 则可以启动对用于执行程序 shell 进行分析, 这样将会记录该 shell 和程序的主进程及其所有子进程的活动。如果您对启动活动不感兴趣, 还可以针对程序的已经运行的进程进行分析。

本部分概要说明了如何使用进程树目标, 不需要使用样例应用程序。

- 在终端窗口中, 键入以下命令来确定 shell 的 PID。
 

```
% echo $$
```
- 在 DLight 中, 单击 "New DLight Target" (新建 DLight 目标) 按钮旁边的向下箭头并选择 "Process Tree Target" (进程树目标)。

"Create New Process Tree Target" (创建新进程树目标) 对话框将打开, 其中 "Process Tree Target" (进程树目标) 选项卡处于选定状态。该选项卡显示系统上运行的进程列表。

- 在 "Process Tree Target" (进程树目标) 选项卡的 "Filter" (过滤器) 字段中, 键入 echo 命令返回的 PID 以在进程列表中查找该 PID。

进程列表应自动刷新, 以仅显示包含您在 "Filter" (过滤器) 字段中键入的字符串的进程。

- 选择使用该 PID 运行的进程并单击 "Save" (保存)。

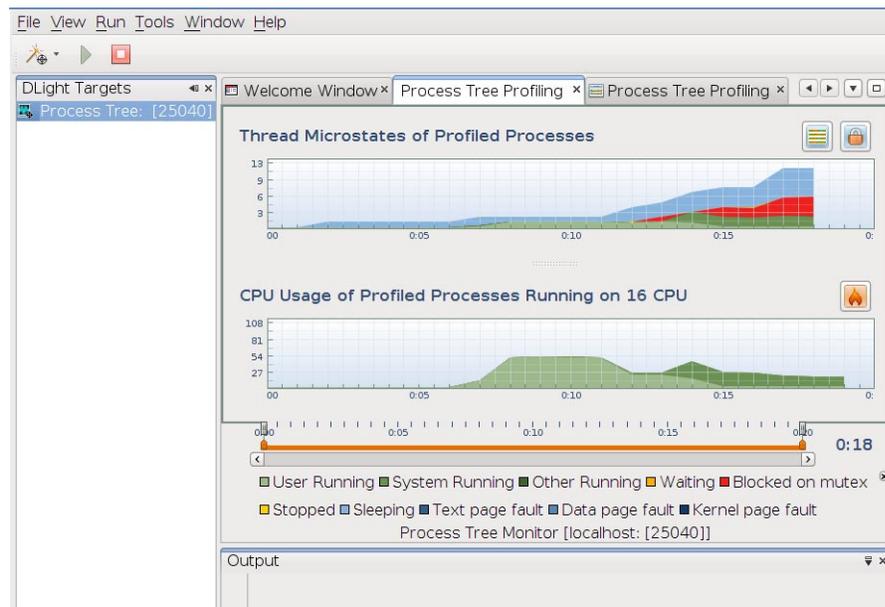
目标将列在 "DLight Targets" (DLight 目标) 选项卡中, 并具有标签 "Process Tree: [pid] [localhost]" (进程树: [pid] [localhost])。目标将永久保存为在此 PID 上运行, 而不管您从在此 PID 下运行的 shell 运行什么程序。

- 选择进程树目标并单击 "Run" (运行) 按钮: 。

"Process Tree Profiling" (进程树分析) 选项卡将在 DLight 中打开, 并显示标签为 "Thread Microstates of Profiled Processes" (已分析进程的线程微观状态) 和 "CPU Usage of Profiled Processes" (已分析进程的 CPU 使用情况) 的图形。最初, 仅分析 shell, 没有发生许多活动。

- 在运行目标 shell 的终端窗口中, 启动要分析的程序。

下图显示了 "Process Tree Profiling" (进程树分析) 选项卡, 其中包含正在名为 locker 的程序中生成的数据, 该程序启动了许多子进程。



这些图形一起可用来确定您应用程序的多个进程和多个线程是如何共同工作的。您可以看到线程被阻塞的点以及对 CPU 使用情况的影响, 并将问题范围缩小到发生问题的代码行。

- 检查 "Process Tree Profiling" (进程树分析) 选项卡顶部的 "Thread Microstates of Profiled Processes" (已分析进程的线程微观状态) 图形。

在 "Thread Microstates of Profiled Processes" (已分析进程的线程微观状态) 图形中, 垂直轴显示正在运行的线程数, 水平轴显示进程运行时经过的挂钟时间。但是, 该图形显示的是 DLight 使用进程树目标分析的所有进程的所有线程的微观状态综合信息。可以使用该图形大概了解进程的行为。

- 检查 "CPU Usage of Profiled Processes" (已分析进程的 CPU 使用情况) 图形。

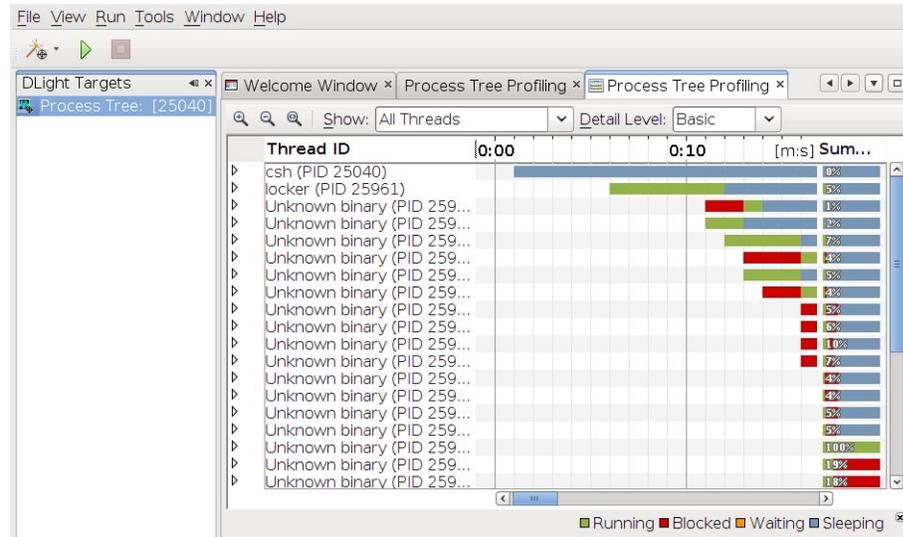
"CPU Usage of Profiled Processes" (已分析进程的 CPU 使用情况) 图形的垂直轴显示 CPU 时间百分比, 水平轴显示进程运行时经过的挂钟时间。但是, 显示的数据是在系统上所有 CPU 中分析的所有进程中所有线程的 CPU 使用情况综合信息。在图形的标题中以及在 DLight 窗口左侧的 "Host Info" (主机信息) 窗口中注明了 CPU 数量。

通过将 "Thread Microstates" (线程微状态) 图形与 "CPU Usage" (CPU 使用情况) 图形一起使用, 可以确定存在锁定问题的位置。

- 单击 "Thread Map" (线程图) 按钮  来打开新的 "Process Tree Profiling" (进程树分析) 选项卡, 以显示有关进程和线程微状态的详细信息。

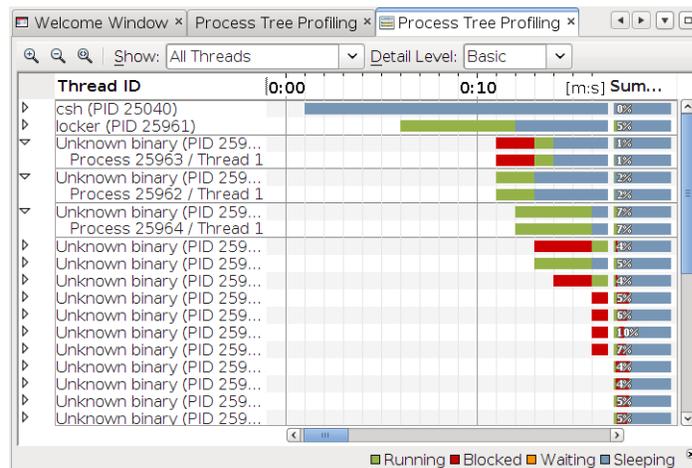
"Process Tree Microstate Details" (进程树微观状态详细信息) 窗口以时间线形式显示 DLight 监视目标期间, 目标进程及其子进程的每个线程的微状态转换。

最初打开该窗口时, 显示每个已分析的进程。目标进程是最顶部的行, 它显示该进程中执行的所有线程的微状态综合信息。子进程时间线还显示它们执行的线程的微状态。



在上图中, 最顶部的行显示在 DLight 目标中指定的 csh shell 进程, 程序运行时该进程处于休眠状态。第二行是 locker 程序的主进程, 其余的 "Unknown binary" (未知二进制文件) 进程是由 locker 程序作为原始进程的派生进程而启动的。DLight 有时无法确定进程名称是一个已知问题。

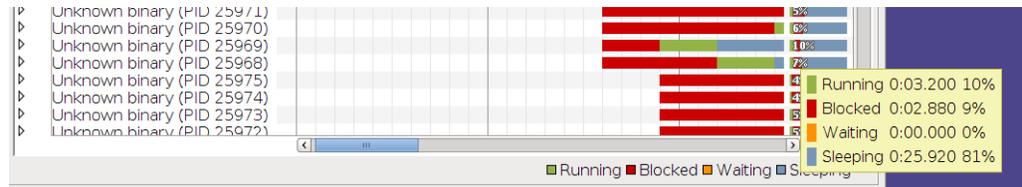
- 单击进程时间线的句柄来显示该进程创建的每个线程的时间线。



如果进程是单线程的, 如本文档中所示, 则进程时间线与线程时间线看起来一样。但是, 当您单击线程时间线时, 它们会显示更多信息, 如后面所述。如果进程是多线程的, 则展开该进程将显示多个线程。

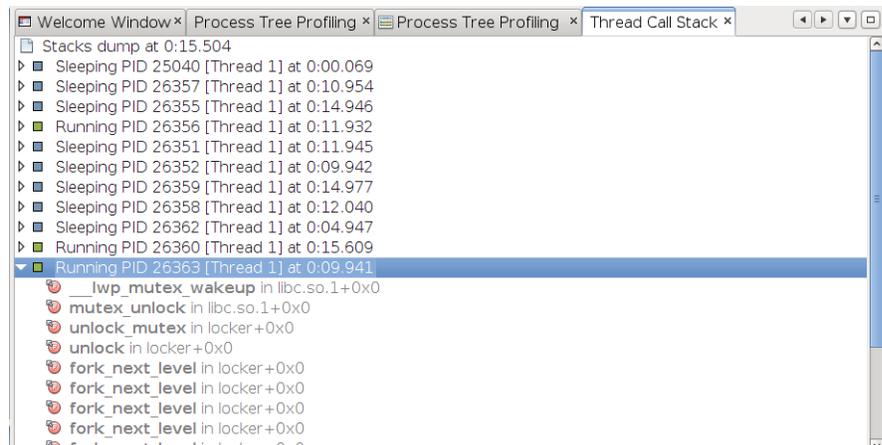
11. 将鼠标光标放置到右侧的"Summary"（摘要）区域上方。

"Summary"（摘要）区域显示线程或进程的整个运行在微状态中花费的时间百分比。将鼠标光标放置到"Summary"（摘要）的上方时，会显示图例来指示颜色的含义以及进程或线程的时间百分比。将鼠标光标放置到进程和线程时间线自身上方时，会显示类似的弹出式窗口。



在上图中，鼠标光标被放置在线程的"Summary"（摘要）区域上方。

12. 单击进程时间线的句柄来显示其线程时间线，然后单击线程时间线中的某个点，"Thread Call Stack"（线程调用堆栈）窗口将打开来显示在该时刻执行的调用堆栈。

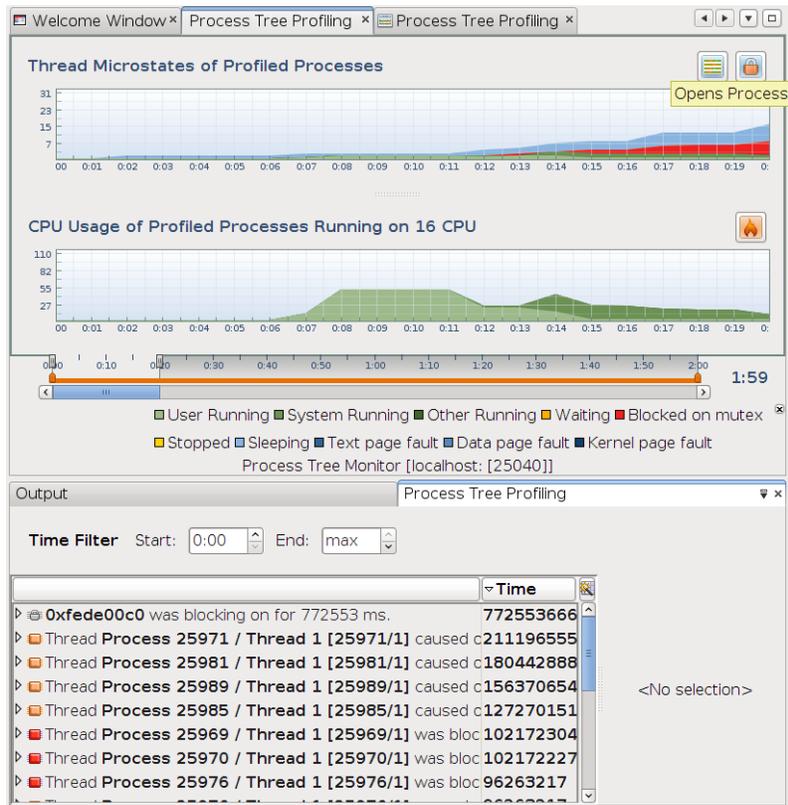


在调用堆栈中，如果程序的源代码可用，则突出显示的函数表示您的源代码中出现的函数。双击突出显示的函数可以在 DLight 中的新窗口中查看源代码。如果可执行文件是使用调试信息生成的（使用 -g 编译器选项），则 DLight 还可以提供源代码导航。

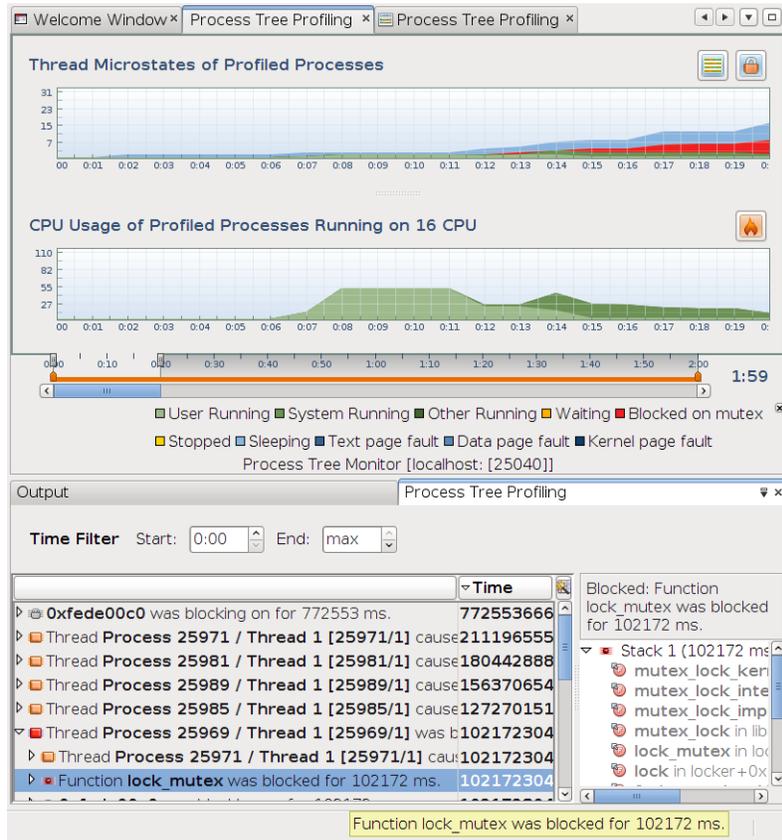
有关查看微状态时间线的更多信息，请通过从 DLight 菜单栏中选择"Help"（帮助）⇒"Help Contents"（帮助内容）来查看 DLight 帮助。

13. 单击主"Process Tree Profiling"（进程树分析）选项卡以返回"Thread Microstates of Profiled Processes"（已分析进程的线程微观状态）图形和"CPU Usage of Profiled Processes"（已分析进程的 CPU 使用情况）图形。
14. 单击"Lock Stat"（锁定 Stat）按钮  来打开新的选项卡，以显示有关进程和线程微状态的详细信息。

一个新的"Process Tree Profiling"（进程树分析）选项卡会打开并显示进程及其子进程的锁定统计信息，其中包括被阻塞的线程的阻塞时间量、持有锁的线程、互斥锁以及线程被阻塞的代码行。



15. 单击左侧的句柄以显示线程、函数和正在阻塞或被阻塞的代码行之间的关系。
16. 单击列表中的项以在侧窗口中打开线程调用堆栈。



如果阻塞影响多个调用堆栈，则每个堆栈都列在此窗口中。针对列表中的项显示的阻塞时间是对该项的所有调用的累积时间。在调用堆栈窗口中，可以查看每个调用堆栈使用了多少阻塞时间。最消耗时间的调用堆栈将第一个列出。如果一个代码行在代码中是在不同路径中执行的，则您可以看到多个调用堆栈，即使该代码行使用了所有阻塞时间也是如此。

17. 请注意，调用堆栈中的某些函数以粗体显示。单击粗体函数名称可以在 DLight 中打开所列行号处的源文件。

由于 DLight 可以访问源代码，因此函数将以粗体列出并显示源文件名和行号。

18. 单击主 "Process Tree Profiling"（进程树分析）选项卡以返回 "Thread Microstates of Profiled Processes"（已分析进程的线程微观状态）图形和 "CPU Usage of Profiled Processes"（已分析进程的 CPU 使用情况）图形。

19. 单击 "CPU Usage of Profiled Processes"（已分析进程的 CPU 使用情况）面板中的 "Hot Spots"（热

点）按钮  来打开新的选项卡，以帮助确定程序的进程树中占用大量 CPU 的区域（热点）。

Function Name	CPU Time, sec
_write in Unknown binary	131.949
<b>work_run_usrcpu at utils.c:25</b>	<b>125.981</b>
0xfece767 in Unknown binary	12.354
postfork1_child_sigev_mq in Unknown binary	7.886
fwrite in Unknown binary	6.098
getxfdats in Unknown binary	3.942
strcmp in Unknown binary	3.229
memcpy in Unknown binary	2.774
_forkx in locker	2.024
close in Unknown binary	1.370
getegid in csh	0.591

该选项卡显示程序中的函数及其独占 CPU 时间度量。这是仅由特定函数所使用的时间，而不包括它所调用的任何函数所使用的 CPU 时间。

请注意，列表中的某些函数以粗体显示。

20. 双击粗体函数名称以打开所列行号处的源文件。

如果源文件可用于 DLight，将打开一个新窗口来显示源文件。

## 有关更多信息

有关使用 DLight 图形工具来确定应用程序性能问题的更多信息，请从 DLight 菜单栏中选择 "\Help\" (帮助) ⇒ "\Help Contents\" (帮助内容)。

帮助包括对本文中未涵盖主题的说明，其中包括：

- 对远处主机进行分析，以便您可以在本地运行 DLight 并分析在远处系统上运行的应用程序。
- 指定工具配置来创建 DLight 工具的子集，以便仅查看您感兴趣的工具。
- 每个工具的详细用法信息，用于说明可以使用每个工具执行的任务。

版权所有 ©2011 本软件和相关文档是根据许可证协议提供的，该许可证协议中规定了关于使用和公开本软件和相关文档的各种限制，并受知识产权法的保护。除非在许可证协议中明确许可或适用法律明确授权，否则不得以任何形式、任何方式使用、拷贝、复制、翻译、广播、修改、授权、传播、分发、展示、执行、发布或显示本软件和相关文档的任何部分。除非法律要求实现互操作，否则严禁对本软件进行逆向工程设计、反汇编或反编译。

此文档所含信息可能随时被修改，恕不另行通知，我们不保证该信息没有错误。如果贵方发现任何问题，请书面通知我们。

如果将本软件或相关文档交付给美国政府，或者交付给以美国政府名义获得许可证的任何机构，必须符合以下规定：

**U.S. GOVERNMENT END USERS:**

Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

本软件或硬件是为了在各种信息管理应用领域内的一般使用而开发的。它不应被应用于任何存在危险或潜在危险的应用领域，也不是为此而开发的，其中包括可能会产生人身伤害的应用领域。如果在危险应用领域内使用本软件或硬件，贵方应负责采取所有适当的防范措施，包括备份、冗余和其它确保安全使用本软件或硬件的措施。对于因在危险应用领域内使用本软件或硬件所造成的一切损失或损害，Oracle Corporation 及其附属公司概不负责。

Oracle 和 Java 是 Oracle 和/或其附属公司的注册商标。其他名称可能是各自所有者的商标。

Intel 和 Intel Xeon 是 Intel Corporation 的商标或注册商标。所有 SPARC 商标均是 SPARC International, Inc 的商标或注册商标，并应按照许可证的规定使用。AMD、Opteron、AMD 徽标以及 AMD Opteron 徽标是 Advanced Micro Devices 的商标或注册商标。UNIX 是 The Open Group 的注册商标。

本软件或硬件以及文档可能提供了访问第三方内容、产品和服务的方式或有关这些内容、产品和服务的信息。对于第三方内容、产品和服务，Oracle Corporation 及其附属公司明确表示不承担任何种类的担保，亦不对其承担任何责任。对于因访问或使用第三方内容、产品或服务所造成的任何损失、成本或损害，Oracle Corporation 及其附属公司概不负责。

E26465

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