Task isolation prctl interface

Certain types of applications benefit from running uninterrupted by background OS activities. Realtime systems and high-bandwidth networking applications with user-space drivers can fall into the category.

To create an OS noise free environment for the application, this interface allows userspace to inform the kernel the start and end of the latency sensitive application section (with configurable system behaviour for that section).

Note: the prctl interface is independent of nohz_full=.

The prctl options are:

- PR_ISOL_FEAT_GET: Retrieve supported features.
- PR_ISOL_CFG_GET: Retrieve task isolation configuration.
- PR_ISOL_CFG_SET: Set task isolation configuration.
- PR_ISOL_ACTIVATE_GET: Retrieve task isolation activation state.
- PR_ISOL_ACTIVATE_SET: Set task isolation activation state.

Summary of terms:

- feature:

  A distinct attribute or aspect of task isolation. Examples of features could be logging, new operating modes (eg: syscalls disallowed), userspace notifications, etc. The only feature currently available is quiescing.

- configuration:

  A specific choice from a given set of possible choices that dictate how the particular feature in question should behave.

- activation state:

  The activation state (whether activate/inactive) of the task isolation features (features must be configured before being activated).

Inheritance of the isolation parameters and state, across fork(2) and clone(2), can be changed via PR_ISOL_CFG_GET/PR_ISOL_CFG_SET.

At a high-level, task isolation is divided in two steps:

1. Configuration.
2. Activation.

Section "Userspace support" describes how to use task isolation.

In terms of the interface, the sequence of steps to activate task isolation are:

1. Retrieve supported task isolation features (PR_ISOL_FEAT_GET).
2. Configure task isolation features (PR_ISOL_CFG_GET/PR_ISOL_CFG_SET).
3. Activate or deactivate task isolation features (PR_ISOL_ACTIVATE_GET/PR_ISOL_ACTIVATE_SET).

This interface is based on ideas and code from the task isolation patchset from Alex Belits: https://lwn.net/Articles/816298/

Feature description

- `ISOL_F_QUIESCE`

This feature allows quiescing select kernel activities on return from system calls.

Interface description

PR_ISOL_FEAT:

Returns the supported features and feature capabilities, as a bitmask:

```
prctl(PR_ISOL_FEAT, feat, arg3, arg4, arg5);
```

The 'feat' argument specifies whether to return supported features (if zero), or feature capabilities (if not zero). Possible values for 'feat' are:

- **`0`:** Return the bitmask of supported features, in the location pointed to by `(int *)arg3`. The buffer should allow space for 8 bytes.
- `ISOL_F_QUIESCE`:

  Return a structure containing which kernel activities are supported for quiescing, in the location pointed to by `(int *)arg3`:

  ```
  struct task_isol_quiesce_extensions {
          __u64 flags;
          __u64 supported_quiesce_bits;
          __u64 pad[6];
  };
  ```

  Where:
  flags: Additional flags (should be zero).

  ***supported_quiesce_bits*: Bitmask indicating** which features are supported for quiescing.

  pad: Additional space for future enhancements.

Features and its capabilities are defined at include/uapi/linux/task_isolation.h.

PR_ISOL_CFG_GET:

Retrieve task isolation configuration. The general format is:

```
prctl(PR_ISOL_CFG_GET, what, arg3, arg4, arg5);
```

The 'what' argument specifies what to configure. Possible values are:

- `I_CFG_FEAT`:

  Return configuration of task isolation features. The 'arg3' argument specifies whether to return configured features (if zero), or individual feature configuration (if not zero), as follows.

  - `0`:

    Return the bitmask of configured features, in the location pointed to by `(int *)arg4`. The buffer should allow space for 8 bytes.

  - `ISOL_F_QUIESCE`:

    Return the control structure for quiescing of background kernel activities, in the location pointed to by `(int *)arg4`:

    ```
    struct task_isol_quiesce_control {
            __u64 flags;
            __u64 quiesce_mask;
            __u64 pad[6];
    };
    ```

    Where:
    flags: Additional flags (should be zero).
    quiesce_mask: A bitmask containing which activities are configured for quiescing.
    pad: Additional space for future enhancements.

- `I_CFG_INHERIT`:

  Retrieve inheritance configuration across fork/clone. Return the structure which configures inheritance across fork/clone, in the location pointed to by `(int *)arg4`:

  ```
  struct task_isol_inherit_control {
          __u8    inherit_mask;
          __u8    pad[7];
  };
  ```

  See PR_ISOL_CFG_SET description for meaning of bits.

PR_ISOL_CFG_SET:

Set task isolation configuration. The general format is:

```
prctl(PR_ISOL_CFG_SET, what, arg3, arg4, arg5);
```

The 'what' argument specifies what to configure. Possible values are:

- `I_CFG_FEAT`:

  Set configuration of task isolation features. 'arg3' specifies the feature. Possible values are:

  - `ISOL_F_QUIESCE`:

    Set the control structure for quiescing of background kernel activities, from the location pointed to by `(int *)arg4`:

    ```
    struct task_isol_quiesce_control {
            __u64 flags;
            __u64 quiesce_mask;
            __u64 pad[6];
    };
    ```

Where:
flags: Additional flags (should be zero).
quiesce_mask: A bitmask containing which kernel activities to quiesce.
pad: Additional space for future enhancements.
For quiesce_mask, possible bit sets are:

`* ISOL_F_QUIESCE_VMSTATS`

VM statistics are maintained in per-CPU counters to improve performance. When a CPU modifies a VM statistic, this modification is kept in the per-CPU counter. Certain activities require a global count, which involves requesting each CPU to flush its local counters to the global VM counters.
This flush is implemented via a workqueue item, which might schedule a workqueue on isolated CPUs.
To avoid this interruption, task isolation can be configured to, upon return from system calls, synchronize the per-CPU counters to global counters, thus avoiding the interruption.
To ensure the application returns to userspace with no modified per-CPU counters, its necessary to use mlockall() in addition to this isolcpus flag.

- **`I_CFG_INHERIT`:** Set inheritance configuration when a new task is created via fork and clone.
  The `(int *)arg4` argument is a pointer to:

  ```
  struct task_isol_inherit_control {
          __u8    inherit_mask;
          __u8    pad[7];
  };
  ```

  inherit_mask is a bitmask that specifies which part of task isolation should be inherited:

  - Bit ISOL_INHERIT_CONF: Inherit task isolation configuration. This is the stated written via prctl(PR_ISOL_CFG_SET, ...).
  - Bit ISOL_INHERIT_ACTIVE: Inherit task isolation activation (requires ISOL_INHERIT_CONF to be set). The new task should behave, after fork/clone, in the same manner as the parent task after it executed:

    prctl(PR_ISOL_ACTIVATE_SET, &mask, ...);

PR_ISOL_ACTIVATE_GET:

Retrieve task isolation activation state.

The general format is:

```
prctl(PR_ISOL_ACTIVATE_GET, pmask, arg3, arg4, arg5);
```

'pmask' specifies the location of a feature mask, where the current active mask will be copied. See PR_ISOL_ACTIVATE_SET for description of individual bits.

PR_ISOL_ACTIVATE_SET:

Set task isolation activation state (activates/deactivates task isolation).

The general format is:

```
prctl(PR_ISOL_ACTIVATE_SET, pmask, arg3, arg4, arg5);
```

The 'pmask' argument specifies the location of an 8 byte mask containing which features should be activated. Features whose bits are cleared will be deactivated. The possible bits for this mask are:

- `ISOL_F_QUIESCE`:
  Activate quiescing of background kernel activities. Quiescing happens on return to userspace from this system call, and on return from subsequent system calls (unless quiesce_oneshot_mask is configured, see below).

If the arg3 argument is non-zero, it specifies a pointer to:

```
struct task_isol_activate_control {
        __u64 flags;
        __u64 quiesce_oneshot_mask;
        __u64 pad[6];
};
```

Where:

flags: Additional flags (should be zero).

***quiesce_oneshot_mask*: Quiescing for the kernel activities** with bits set on this mask will happen on the return from this system call, but not on return from subsequent ones.

Quiescing can be adjusted (while active) by prctl(PR_ISOL_ACTIVATE_SET, &new_mask, ...).

Userspace support

Task isolation is divided in two main steps: configuration and activation.

Each step can be performed by an external tool or the latency sensitive application itself. util-linux contains the "chisol" tool for this purpose.

This results in three combinations:

1. Both configuration and activation performed by the latency sensitive application. Allows fine grained control of what task isolation features are enabled and when (see samples section below).

2. Only activation can be performed by the latency sensitive app (and configuration performed by chisol). This allows the admin/user to control task isolation parameters, and applications have to be modified only once.

3. Configuration and activation performed by an external tool. This allows unmodified applications to take advantage of task isolation. Activation is performed by the "-a" option of chisol.

Examples

The `samples/task_isolation/` directory contains 3 examples:

- task_isol_userloop.c:

  Example of program with a loop on userspace scenario.

- task_isol_computation.c:

  Example of program that enters task isolated mode, performs an amount of computation, exits task isolated mode, and writes the computation to disk.

- task_isol_oneshot.c:

  Example of program that enables one-shot mode for quiescing, enters a processing loop, then upon an external event performs a number of syscalls to handle that event.

This is a snippet of code to activate task isolation if it has been previously configured (by chisol for example):

```
#include <sys/prctl.h>
#include <linux/types.h>

#ifdef PR_ISOL_CFG_GET
unsigned long long fmask;

ret = prctl(PR_ISOL_CFG_GET, I_CFG_FEAT, 0, &fmask, 0);
if (ret != -1 && fmask != 0) {
        ret = prctl(PR_ISOL_ACTIVATE_SET, &fmask, 0, 0, 0);
        if (ret == -1) {
                perror("prctl PR_ISOL_ACTIVATE_SET");
                return ret;
        }
}
#endif
```