Per-task statistics interface

Taskstats is a netlink-based interface for sending per-task and per-process statistics from the kernel to userspace.

Taskstats was designed for the following benefits:

- efficiently provide statistics during lifetime of a task and on its exit
- unified interface for multiple accounting subsystems
- extensibility for use by future accounting patches

Terminology

"pid", "tid" and "task" are used interchangeably and refer to the standard Linux task defined by struct task_struct. per-pid stats are the same as per-task stats.

"tgid", "process" and "thread group" are used interchangeably and refer to the tasks that share an mm_struct i.e. the traditional Unix process. Despite the use of tgid, there is no special treatment for the task that is thread group leader - a process is deemed alive as long as it has any task belonging to it.

Usage

To get statistics during a task's lifetime, userspace opens a unicast netlink socket (NETLINK_GENERIC family) and sends commands specifying a pid or a tgid. The response contains statistics for a task (if pid is specified) or the sum of statistics for all tasks of the process (if tgid is specified).

To obtain statistics for tasks which are exiting, the userspace listener sends a register command and specifies a cpumask. Whenever a task exits on one of the cpus in the cpumask, its per-pid statistics are sent to the registered listener. Using cpumasks allows the data received by one listener to be limited and assists in flow control over the netlink interface and is explained in more detail below.

If the exiting task is the last thread exiting its thread group, an additional record containing the per-tgid stats is also sent to userspace. The latter contains the sum of per-pid stats for all threads in the thread group, both past and present.

getdelays.c is a simple utility demonstrating usage of the taskstats interface for reporting delay accounting statistics. Users can register cpumasks, send commands and process responses, listen for per-tid/tgid exit data, write the data received to a file and do basic flow control by increasing receive buffer sizes.

Interface

The user-kernel interface is encapsulated in include/linux/taskstats.h

To avoid this documentation becoming obsolete as the interface evolves, only an outline of the current version is given. taskstats.h always overrides the description here.

struct taskstats is the common accounting structure for both per-pid and per-tgid data. It is versioned and can be extended by each accounting subsystem that is added to the kernel. The fields and their semantics are defined in the taskstats.h file.

The data exchanged between user and kernel space is a netlink message belonging to the NETLINK_GENERIC family and using the netlink attributes interface. The messages are in the format:

```
+----------+- - -+-------------+-------------------+
| nlmsghdr | Pad |  genlmsghdr | taskstats payload |
+----------+- - -+-------------+-------------------+
```

The taskstats payload is one of the following three kinds:

1. Commands: Sent from user to kernel. Commands to get data on a pid/tgid consist of one attribute, of type TASKSTATS_CMD_ATTR_PID/TGID, containing a u32 pid or tgid in the attribute payload. The pid/tgid denotes the task/process for which userspace wants statistics.

Commands to register/deregister interest in exit data from a set of cpus consist of one attribute, of type TASKSTATS_CMD_ATTR_REGISTER/DEREGISTER_CPUMASK and contain a cpumask in the attribute payload. The cpumask is specified as an ascii string of comma-separated cpu ranges e.g. to listen to exit data from cpus 1,2,3,5,7,8 the cpumask would be "1-3,5,7-8". If userspace forgets to deregister interest in cpus before closing the listening socket, the kernel cleans up its interest set over time. However, for the sake of efficiency, an explicit deregistration is advisable.

2. Response for a command: sent from the kernel in response to a userspace command. The payload is a series of three attributes of type:

a) TASKSTATS_TYPE_AGGR_PID/TGID : attribute containing no payload but indicates a pid/tgid will be followed by some stats.

b) TASKSTATS_TYPE_PID/TGID: attribute whose payload is the pid/tgid whose stats are being returned.

c) TASKSTATS_TYPE_STATS: attribute with a struct taskstats as payload. The same structure is used for both per-pid and per-tgid stats.

3. New message sent by kernel whenever a task exits. The payload consists of a series of attributes of the following type:

a. TASKSTATS_TYPE_AGGR_PID: indicates next two attributes will be pid+stats
b. TASKSTATS_TYPE_PID: contains exiting task's pid
c. TASKSTATS_TYPE_STATS: contains the exiting task's per-pid stats
d. TASKSTATS_TYPE_AGGR_TGID: indicates next two attributes will be tgid+stats
e. TASKSTATS_TYPE_TGID: contains tgid of process to which task belongs
f. TASKSTATS_TYPE_STATS: contains the per-tgid stats for exiting task's process

per-tgid stats

Taskstats provides per-process stats, in addition to per-task stats, since resource management is often done at a process granularity and aggregating task stats in userspace alone is inefficient and potentially inaccurate (due to lack of atomicity).

However, maintaining per-process, in addition to per-task stats, within the kernel has space and time overheads. To address this, the taskstats code accumulates each exiting task's statistics into a process-wide data structure. When the last task of a process exits, the process level data accumulated also gets sent to userspace (along with the per-task data).

When a user queries to get per-tgid data, the sum of all other live threads in the group is added up and added to the accumulated total for previously exited threads of the same thread group.

Extending taskstats

There are two ways to extend the taskstats interface to export more per-task/process stats as patches to collect them get added to the kernel in future:

1. Adding more fields to the end of the existing struct taskstats. Backward compatibility is ensured by the version number within the structure. Userspace will use only the fields of the struct that correspond to the version its using.
2. Defining separate statistic structs and using the netlink attributes interface to return them. Since userspace processes each netlink attribute independently, it can always ignore attributes whose type it does not understand (because it is using an older version of the interface).

Choosing between 1. and 2. is a matter of trading off flexibility and overhead. If only a few fields need to be added, then 1. is the preferable path since the kernel and userspace don't need to incur the overhead of processing new netlink attributes. But if the new fields expand the existing struct too much, requiring disparate userspace accounting utilities to unnecessarily receive large structures whose fields are of no interest, then extending the attributes structure would be worthwhile.

Flow control for taskstats

When the rate of task exits becomes large, a listener may not be able to keep up with the kernel's rate of sending per-tid/tgid exit data leading to data loss. This possibility gets compounded when the taskstats structure gets extended and the number of cpus grows large.

To avoid losing statistics, userspace should do one or more of the following:

- increase the receive buffer sizes for the netlink sockets opened by listeners to receive exit data.
- create more listeners and reduce the number of cpus being listened to by each listener. In the extreme case, there could be one listener for each cpu. Users may also consider setting the cpu affinity of the listener to the subset of cpus to which it listens, especially if they are listening to just one cpu.

Despite these measures, if the userspace receives ENOBUFS error messages indicated overflow of receive buffers, it should take measures to handle the loss of data.