Detailed Usages

DAMON provides below interfaces for different users.

- *DAMON user space tool.* This is for privileged people such as system administrators who want a just-working human-friendly interface. Using this, users can use the DAMONâ€™s major features in a human-friendly way. It may not be highly tuned for special cases, though. It supports both virtual and physical address spaces monitoring. For more detail, please refer to its usage document.
- *sysfs interface.* :ref:`This <sysfs_interface>` is for privileged user space programmers who want more optimized use of DAMON. Using this, users can use DAMONâ€™s major features by reading from and writing to special sysfs files. Therefore, you can write and use your personalized DAMON sysfs wrapper programs that reads/writes the sysfs files instead of you. The DAMON user space tool is one example of such programs. It supports both virtual and physical address spaces monitoring. Note that this interface provides only simple :ref:`statistics <damos_stats>` for the monitoring results. For detailed monitoring results, DAMON provides a :ref:`tracepoint <tracepoint>`.

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst`, line 17); *backlink*
  >
  > Unknown interpreted text role "ref".

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst`, line 17); *backlink*
  >
  > Unknown interpreted text role "ref".

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst`, line 17); *backlink*
  >
  > Unknown interpreted text role "ref".

- *debugfs interface.* :ref:`This <debugfs_interface>` is almost identical to :ref:`sysfs interface <sysfs_interface>`. This will be removed after next LTS kernel is released, so users should move to the :ref:`sysfs interface <sysfs_interface>`.

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst`, line 28); *backlink*
  >
  > Unknown interpreted text role "ref".

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst`, line 28); *backlink*
  >
  > Unknown interpreted text role "ref".

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst`, line 28); *backlink*
  >
  > Unknown interpreted text role "ref".

- *Kernel Space Programming Interface.* :doc:`This </vm/damon/api>` is for kernel space programmers. Using this, users can utilize every feature of DAMON most flexibly and efficiently by writing kernel space DAMON application programs for you. You can even extend DAMON for various address spaces. For detail, please refer to the interface :doc:`document </vm/damon/api>`.

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst`, line 32); *backlink*

Unknown interpreted text role "doc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst, line 32); *backlink*

Unknown interpreted text role "doc".

sysfs Interface

DAMON sysfs interface is built when `CONFIG_DAMON_SYSFS` is defined. It creates multiple directories and files under its sysfs directory, `<sysfs>/kernel/mm/damon/`. You can control DAMON by writing to and reading from the files under the directory.

For a short example, users can monitor the virtual address space of a given workload as below.

```
# cd /sys/kernel/mm/damon/admin/
# echo 1 > kdamonds/nr && echo 1 > kdamonds/0/contexts/nr
# echo vaddr > kdamonds/0/contexts/0/operations
# echo 1 > kdamonds/0/contexts/0/targets/nr
# echo $(pidof <workload>) > kdamonds/0/contexts/0/targets/0/pid
# echo on > kdamonds/0/state
```

Files Hierarchy

The files hierarchy of DAMON sysfs interface is shown below. In the below figure, parents-children relations are represented with indentations, each directory is having `/` suffix, and files in each directory are separated by comma (",").

```
/sys/kernel/mm/damon/admin
â”, kdamonds/nr_kdamonds
â”, â”, 0/state,pid
â”, â”, â”, contexts/nr_contexts
â”, â”, â”, â”, 0/operations
â”, â”, â”, â”, â”, monitoring_attrs/
â”, â”, â”, â”, â”, â”, intervals/sample_us,aggr_us,update_us
â”, â”, â”, â”, â”, â”, nr_regions/min,max
â”, â”, â”, â”, â”, targets/nr_targets
â”, â”, â”, â”, â”, â”, 0/pid_target
â”, â”, â”, â”, â”, â”, â”, regions/nr_regions
â”, â”, â”, â”, â”, â”, â”, â”, 0/start,end
â”, â”, â”, â”, â”, â”, â”, â”, ...
â”, â”, â”, â”, â”, â”, ...
â”, â”, â”, â”, â”, schemes/nr_schemes
â”, â”, â”, â”, â”, â”, 0/action
â”, â”, â”, â”, â”, â”, â”, access_pattern/
â”, â”, â”, â”, â”, â”, â”, â”, sz/min,max
â”, â”, â”, â”, â”, â”, â”, â”, nr_accesses/min,max
â”, â”, â”, â”, â”, â”, â”, â”, age/min,max
â”, â”, â”, â”, â”, â”, â”, quotas/ms,bytes,reset_interval_ms
â”, â”, â”, â”, â”, â”, â”, â”, weights/sz_permil,nr_accesses_permil,age_permil
â”, â”, â”, â”, â”, â”, â”, watermarks/metric,interval_us,high,mid,low
â”, â”, â”, â”, â”, â”, â”, stats/nr_tried,sz_tried,nr_applied,sz_applied,qt_exceeds
â”, â”, â”, â”, â”, â”, ...
â”, â”, â”, â”, ...
â”, â”, ...
```

Root

The root of the DAMON sysfs interface is `<sysfs>/kernel/mm/damon/`, and it has one directory named `admin`. The directory contains the files for privileged user space programs' control of DAMON. User space tools or deamons having the root permission could use this directory.

kdamonds/

The monitoring-related information including request specifications and results are called DAMON context. DAMON executes each context with a kernel thread called kdamond, and multiple kdamonds could run in parallel.

Under the `admin` directory, one directory, `kdamonds`, which has files for controlling the kdamonds exist. In the beginning, this directory has only one file, `nr_kdamonds`. Writing a number (`N`) to the file creates the number of child directories named `0` to `N-1`. Each directory represents each kdamond.

kdamonds/<N>/

In each kdamond directory, two files (`state` and `pid`) and one directory (`contexts`) exist.

Reading `state` returns `on` if the kdamond is currently running, or `off` if it is not running. Writing `on` or `off` makes the kdamond be in

the state. Writing `update_schemes_stats` to `state` file updates the contents of stats files for each DAMON-based operation scheme of the kdamond. For details of the stats, please refer to :ref:`stats section <sysfs_schemes_stats>`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst, line 122); *backlink***
>
> Unknown interpreted text role "ref".

If the state is `on`, reading `pid` shows the pid of the kdamond thread.

`contexts` directory contains files for controlling the monitoring contexts that this kdamond will execute.

kdamonds/<N>/contexts/

In the beginning, this directory has only one file, `nr_contexts`. Writing a number (`N`) to the file creates the number of child directories named as `0` to `N-1`. Each directory represents each monitoring context. At the moment, only one context per kdamond is supported, so only `0` or `1` can be written to the file.

contexts/<N>/

In each context directory, one file (`operations`) and three directories (`monitoring_attrs`, `targets`, and `schemes`) exist.

DAMON supports multiple types of monitoring operations, including those for virtual address space and the physical address space. You can set and get what type of monitoring operations DAMON will use for the context by writing one of below keywords to, and reading from the file.

- vaddr: Monitor virtual address spaces of specific processes
- paddr: Monitor the physical address space of the system

contexts/<N>/monitoring_attrs/

Files for specifying attributes of the monitoring including required quality and efficiency of the monitoring are in `monitoring_attrs` directory. Specifically, two directories, `intervals` and `nr_regions` exist in this directory.

Under `intervals` directory, three files for DAMON's sampling interval (`sample_us`), aggregation interval (`aggr_us`), and update interval (`update_us`) exist. You can set and get the values in micro-seconds by writing to and reading from the files.

Under `nr_regions` directory, two files for the lower-bound and upper-bound of DAMON's monitoring regions (`min` and `max`, respectively), which controls the monitoring overhead, exist. You can set and get the values by writing to and rading from the files.

For more details about the intervals and monitoring regions range, please refer to the Design document (:doc:`/vm/damon/design`).

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst, line 175); *backlink***
>
> Unknown interpreted text role "doc".

contexts/<N>/targets/

In the beginning, this directory has only one file, `nr_targets`. Writing a number (`N`) to the file creates the number of child directories named `0` to `N-1`. Each directory represents each monitoring target.

targets/<N>/

In each target directory, one file (`pid_target`) and one directory (`regions`) exist.

If you wrote `vaddr` to the `contexts/<N>/operations`, each target should be a process. You can specify the process to DAMON by writing the pid of the process to the `pid_target` file.

targets/<N>/regions

When `vaddr` monitoring operations set is being used (`vaddr` is written to the `contexts/<N>/operations` file), DAMON automatically sets and updates the monitoring target regions so that entire memory mappings of target processes can be covered. However, users could want to set the initial monitoring region to specific address ranges.

In contrast, DAMON do not automatically sets and updates the monitoring target regions when `paddr` monitoring operations set is being used (`paddr` is written to the `contexts/<N>/operations`). Therefore, users should set the monitoring target regions by themselves in the case.

For such cases, users can explicitly set the initial monitoring target regions as they want, by writing proper values to the files under this directory.

In the beginning, this directory has only one file, `nr_regions`. Writing a number (`N`) to the file creates the number of child directories named `0` to `N-1`. Each directory represents each initial monitoring target region.

regions/<N>/

In each region directory, you will find two files (`start` and `end`). You can set and get the start and end addresses of the initial monitoring target region by writing to and reading from the files, respectively.

contexts/<N>/schemes/

For usual DAMON-based data access aware memory management optimizations, users would normally want the system to apply a memory management action to a memory region of a specific access pattern. DAMON receives such formalized operation schemes from the user and applies those to the target memory regions. Users can get and set the schemes by reading from and writing to files under this directory.

In the beginning, this directory has only one file, `nr_schemes`. Writing a number (`N`) to the file creates the number of child directories named `0` to `N-1`. Each directory represents each DAMON-based operation scheme.

schemes/<N>/

In each scheme directory, four directories (`access_pattern`, `quotas`, `watermarks`, and `stats`) and one file (`action`) exist.

The `action` file is for setting and getting what action you want to apply to memory regions having specific access pattern of the interest. The keywords that can be written to and read from the file and their meaning are as below.

- `willneed`: Call `madvise()` for the region with `MADV_WILLNEED`
- `cold`: Call `madvise()` for the region with `MADV_COLD`
- `pageout`: Call `madvise()` for the region with `MADV_PAGEOUT`
- `hugepage`: Call `madvise()` for the region with `MADV_HUGEPAGE`
- `nohugepage`: Call `madvise()` for the region with `MADV_NOHUGEPAGE`
- `stat`: Do nothing but count the statistics

schemes/<N>/access_pattern/

The target access pattern of each DAMON-based operation scheme is constructed with three ranges including the size of the region in bytes, number of monitored accesses per aggregate interval, and number of aggregated intervals for the age of the region.

Under the `access_pattern` directory, three directories (`sz`, `nr_accesses`, and `age`) each having two files (`min` and `max`) exist. You can set and get the access pattern for the given scheme by writing to and reading from the `min` and `max` files under `sz`, `nr_accesses`, and `age` directories, respectively.

schemes/<N>/quotas/

Optimal `target access pattern` for each `action` is workload dependent, so not easy to find. Worse yet, setting a scheme of some action too aggressive can cause severe overhead. To avoid such overhead, users can limit time and size quota for each scheme. In detail, users can ask DAMON to try to use only up to specific time (`time quota`) for applying the action, and to apply the action to only up to specific amount (`size quota`) of memory regions having the target access pattern within a given time interval (`reset interval`).

When the quota limit is expected to be exceeded, DAMON prioritizes found memory regions of the `target access pattern` based on their size, access frequency, and age. For personalized prioritization, users can set the weights for the three properties.

Under `quotas` directory, three files (`ms`, `bytes`, `reset_interval_ms`) and one directory (`weights`) having three files (`sz_permil`, `nr_accesses_permil`, and `age_permil`) in it exist.

You can set the `time quota` in milliseconds, `size quota` in bytes, and `reset interval` in milliseconds by writing the values to the three files, respectively. You can also set the prioritization weights for size, access frequency, and age in per-thousand unit by writing the values to the three files under the `weights` directory.

schemes/<N>/watermarks/

To allow easy activation and deactivation of each scheme based on system status, DAMON provides a feature called watermarks. The feature receives five values called `metric`, `interval`, `high`, `mid`, and `low`. The `metric` is the system metric such as free memory ratio that can be measured. If the metric value of the system is higher than the value in `high` or lower than `low` at the memoent, the scheme is deactivated. If the value is lower than `mid`, the scheme is activated.

Under the watermarks directory, five files (`metric`, `interval_us`, `high`, `mid`, and `low`) for setting each value exist. You can set and get the five values by writing to the files, respectively.

Keywords and meanings of those that can be written to the `metric` file are as below.

- none: Ignore the watermarks
- free_mem_rate: System's free memory rate (per thousand)

The `interval` should written in microseconds unit.

schemes/<N>/stats/

DAMON counts the total number and bytes of regions that each scheme is tried to be applied, the two numbers for the regions that each scheme is successfully applied, and the total number of the quota limit exceeds. This statistics can be used for online analysis or tuning of the schemes.

The statistics can be retrieved by reading the files under `stats` directory (`nr_tried`, `sz_tried`, `nr_applied`, `sz_applied`, and `qt_exceeds`), respectively. The files are not updated in real time, so you should ask DAMON sysfs interface to updte the content of the files for the stats by writing a special keyword, `update_schemes_stats` to the relevant `kdamonds/<N>/state` file.

Example

Below commands applies a scheme saying "If a memory region of size in [4KiB, 8KiB] is showing accesses per aggregate interval in [0, 5] for aggregate interval in [10, 20], page out the region. For the paging out, use only up to 10ms per second, and also don't page out more than 1GiB per second. Under the limitation, page out memory regions having longer age first. Also, check the free memory rate of the system every 5 seconds, start the monitoring and paging out when the free memory rate becomes lower than 50%, but stop it if the free memory rate becomes larger than 60%, or lower than 30%".

```
# cd <sysfs>/kernel/mm/damon/admin
# # populate directories
# echo 1 > kdamonds/nr_kdamonds; echo 1 > kdamonds/0/contexts/nr_contexts;
# echo 1 > kdamonds/0/contexts/0/schemes/nr_schemes
# cd kdamonds/0/contexts/0/schemes/0
# # set the basic access pattern and the action
# echo 4096 > access_patterns/sz/min
# echo 8192 > access_patterns/sz/max
# echo 0 > access_patterns/nr_accesses/min
# echo 5 > access_patterns/nr_accesses/max
# echo 10 > access_patterns/age/min
# echo 20 > access_patterns/age/max
# echo pageout > action
# # set quotas
# echo 10 > quotas/ms
# echo $((1024*1024*1024)) > quotas/bytes
# echo 1000 > quotas/reset_interval_ms
# # set watermark
# echo free_mem_rate > watermarks/metric
# echo 5000000 > watermarks/interval_us
# echo 600 > watermarks/high
# echo 500 > watermarks/mid
# echo 300 > watermarks/low
```

Please note that it's highly recommended to use user space tools like damo rather than manually reading and writing the files as above. Above is only for an example.

debugfs Interface

DAMON exports eight files, `attrs`, `target_ids`, `init_regions`, `schemes`, `monitor_on`, `kdamond_pid`, `mk_contexts` and `rm_contexts` under its debugfs directory, `<debugfs>/damon/`.

Attributes

Users can get and set the `sampling interval`, `aggregation interval`, `update interval`, and min/max number of monitoring target regions by reading from and writing to the `attrs` file. To know about the monitoring attributes in detail, please refer to the :doc:`/vm/damon/design`. For example, below commands set those values to 5 ms, 100 ms, 1,000 ms, 10 and 1000, and then check it again:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\damon\[linux-master][Documentation][admin-guide][mm][damon]usage.rst, line 387); *backlink***
>
> Unknown interpreted text role "doc".

```
# cd <debugfs>/damon
# echo 5000 100000 1000000 10 1000 > attrs
# cat attrs
5000 100000 1000000 10 1000
```

Target IDs

Some types of address spaces supports multiple monitoring target. For example, the virtual memory address spaces monitoring can have multiple processes as the monitoring targets. Users can set the targets by writing relevant id values of the targets to, and get the

ids of the current targets by reading from the `target_ids` file. In case of the virtual address spaces monitoring, the values should be pids of the monitoring target processes. For example, below commands set processes having pids 42 and 4242 as the monitoring targets and check it again:

```
# cd <debugfs>/damon
# echo 42 4242 > target_ids
# cat target_ids
42 4242
```

Users can also monitor the physical memory address space of the system by writing a special keyword, "`paddr\n`" to the file. Because physical address space monitoring doesn't support multiple targets, reading the file will show a fake value, `42`, as below:

```
# cd <debugfs>/damon
# echo paddr > target_ids
# cat target_ids
42
```

Note that setting the target ids doesn't start the monitoring.

Initial Monitoring Target Regions

In case of the virtual address space monitoring, DAMON automatically sets and updates the monitoring target regions so that entire memory mappings of target processes can be covered. However, users can want to limit the monitoring region to specific address ranges, such as the heap, the stack, or specific file-mapped area. Or, some users can know the initial access pattern of their workloads and therefore want to set optimal initial regions for the 'adaptive regions adjustment'.

In contrast, DAMON do not automatically sets and updates the monitoring target regions in case of physical memory monitoring. Therefore, users should set the monitoring target regions by themselves.

In such cases, users can explicitly set the initial monitoring target regions as they want, by writing proper values to the `init_regions` file. Each line of the input should represent one region in below form.:

```
<target idx> <start address> <end address>
```

The `target idx` should be the index of the target in `target_ids` file, starting from `0`, and the regions should be passed in address order. For example, below commands will set a couple of address ranges, `1-100` and `100-200` as the initial monitoring target region of pid 42, which is the first one (index `0`) in `target_ids`, and another couple of address ranges, `20-40` and `50-100` as that of pid 4242, which is the second one (index `1`) in `target_ids`.:

```
# cd <debugfs>/damon
# cat target_ids
42 4242
# echo "0   1       100
        0   100     200
        1   20      40
        1   50      100" > init_regions
```

Note that this sets the initial monitoring target regions only. In case of virtual memory monitoring, DAMON will automatically updates the boundary of the regions after one `update interval`. Therefore, users should set the `update interval` large enough in this case, if they don't want the update.

Schemes

For usual DAMON-based data access aware memory management optimizations, users would simply want the system to apply a memory management action to a memory region of a specific access pattern. DAMON receives such formalized operation schemes from the user and applies those to the target processes.

Users can get and set the schemes by reading from and writing to `schemes` debugfs file. Reading the file also shows the statistics of each scheme. To the file, each of the schemes should be represented in each line in below form:

```
<target access pattern> <action> <quota> <watermarks>
```

You can disable schemes by simply writing an empty string to the file.

Target Access Pattern

The `<target access pattern>` is constructed with three ranges in below form:

```
min-size max-size min-acc max-acc min-age max-age
```

Specifically, bytes for the size of regions (`min-size` and `max-size`), number of monitored accesses per aggregate interval for access frequency (`min-acc` and `max-acc`), number of aggregate intervals for the age of regions (`min-age` and `max-age`) are specified. Note that the ranges are closed interval.

Action

The `<action>` is a predefined integer for memory management actions, which DAMON will apply to the regions having the target

access pattern. The supported numbers and their meanings are as below.

- 0: Call `madvise()` for the region with `MADV_WILLNEED`
- 1: Call `madvise()` for the region with `MADV_COLD`
- 2: Call `madvise()` for the region with `MADV_PAGEOUT`
- 3: Call `madvise()` for the region with `MADV_HUGEPAGE`
- 4: Call `madvise()` for the region with `MADV_NOHUGEPAGE`
- 5: Do nothing but count the statistics

Quota

Optimal `target access pattern` for each `action` is workload dependent, so not easy to find. Worse yet, setting a scheme of some action too aggressive can cause severe overhead. To avoid such overhead, users can limit time and size quota for the scheme via the `<quota>` in below form:

```
<ms> <sz> <reset interval> <priority weights>
```

This makes DAMON to try to use only up to `<ms>` milliseconds for applying the action to memory regions of the `target access pattern` within the `<reset interval>` milliseconds, and to apply the action to only up to `<sz>` bytes of memory regions within the `<reset interval>`. Setting both `<ms>` and `<sz>` zero disables the quota limits.

When the quota limit is expected to be exceeded, DAMON prioritizes found memory regions of the `target access pattern` based on their size, access frequency, and age. For personalized prioritization, users can set the weights for the three properties in `<priority weights>` in below form:

```
<size weight> <access frequency weight> <age weight>
```

Watermarks

Some schemes would need to run based on current value of the system's specific metrics like free memory ratio. For such cases, users can specify watermarks for the condition.:

```
<metric> <check interval> <high mark> <middle mark> <low mark>
```

`<metric>` is a predefined integer for the metric to be checked. The supported numbers and their meanings are as below.

- 0: Ignore the watermarks
- 1: System's free memory rate (per thousand)

The value of the metric is checked every `<check interval>` microseconds.

If the value is higher than `<high mark>` or lower than `<low mark>`, the scheme is deactivated. If the value is lower than `<mid mark>`, the scheme is activated.

Statistics

It also counts the total number and bytes of regions that each scheme is tried to be applied, the two numbers for the regions that each scheme is successfully applied, and the total number of the quota limit exceeds. This statistics can be used for online analysis or tuning of the schemes.

The statistics can be shown by reading the `schemes` file. Reading the file will show each scheme you entered in each line, and the five numbers for the statistics will be added at the end of each line.

Example

Below commands applies a scheme saying "If a memory region of size in [4KiB, 8KiB] is showing accesses per aggregate interval in [0, 5] for aggregate interval in [10, 20], page out the region. For the paging out, use only up to 10ms per second, and also don't page out more than 1GiB per second. Under the limitation, page out memory regions having longer age first. Also, check the free memory rate of the system every 5 seconds, start the monitoring and paging out when the free memory rate becomes lower than 50%, but stop it if the free memory rate becomes larger than 60%, or lower than 30%".:

```
# cd <debugfs>/damon
# scheme="4096 8192  0 5    10 20    2"  # target access pattern and action
# scheme+=" 10 $((1024*1024*1024)) 1000" # quotas
# scheme+=" 0 0 100"                     # prioritization weights
# scheme+=" 1 5000000 600 500 300"       # watermarks
# echo "$scheme" > schemes
```

Turning On/Off

Setting the files as described above doesn't incur effect unless you explicitly start the monitoring. You can start, stop, and check the current status of the monitoring by writing to and reading from the `monitor_on` file. Writing `on` to the file starts the monitoring of the targets with the attributes. Writing `off` to the file stops those. DAMON also stops if every target process is terminated. Below example commands turn on, off, and check the status of DAMON:

```
# cd <debugfs>/damon
# echo on > monitor_on
# echo off > monitor_on
# cat monitor_on
off
```

Please note that you cannot write to the above-mentioned debugfs files while the monitoring is turned on. If you write to the files while DAMON is running, an error code such as `-EBUSY` will be returned.

Monitoring Thread PID

DAMON does requested monitoring with a kernel thread called `kdamond`. You can get the pid of the thread by reading the `kdamond_pid` file. When the monitoring is turned off, reading the file returns `none`.

```
# cd <debugfs>/damon
# cat monitor_on
off
# cat kdamond_pid
none
# echo on > monitor_on
# cat kdamond_pid
18594
```

Using Multiple Monitoring Threads

One `kdamond` thread is created for each monitoring context. You can create and remove monitoring contexts for multiple `kdamond` required use case using the `mk_contexts` and `rm_contexts` files.

Writing the name of the new context to the `mk_contexts` file creates a directory of the name on the DAMON debugfs directory. The directory will have DAMON debugfs files for the context.

```
# cd <debugfs>/damon
# ls foo
# ls: cannot access 'foo': No such file or directory
# echo foo > mk_contexts
# ls foo
# attrs  init_regions  kdamond_pid  schemes  target_ids
```

If the context is not needed anymore, you can remove it and the corresponding directory by putting the name of the context to the `rm_contexts` file.

```
# echo foo > rm_contexts
# ls foo
# ls: cannot access 'foo': No such file or directory
```

Note that `mk_contexts`, `rm_contexts`, and `monitor_on` files are in the root directory only.

Tracepoint for Monitoring Results

DAMON provides the monitoring results via a tracepoint, `damon:damon_aggregated`. While the monitoring is turned on, you could record the tracepoint events and show results using tracepoint supporting tools like `perf`. For example:

```
# echo on > monitor_on
# perf record -e damon:damon_aggregated &
# sleep 5
# kill 9 $(pidof perf)
# echo off > monitor_on
# perf script
```