Documentation for userland software suspend interface

C. 2006 Rafael J. Wysocki <rjw@sisk.pl>

First, the warnings at the beginning of swsusp.txt still apply.

Second, you should read the FAQ in swsusp.txt _now_ if you have not done it already.

Now, to use the userland interface for software suspend you need special utilities that will read/write the system memory snapshot from/to the kernel. Such utilities are available, for example, from <http://suspend.sourceforge.net>. You may want to have a look at them if you are going to develop your own suspend/resume utilities.

The interface consists of a character device providing the open(), release(), read(), and write() operations as well as several ioctl() commands defined in include/linux/suspend_ioctls.h . The major and minor numbers of the device are, respectively, 10 and 231, and they can be read from /sys/class/misc/snapshot/dev.

The device can be open either for reading or for writing. If open for reading, it is considered to be in the suspend mode. Otherwise it is assumed to be in the resume mode. The device cannot be open for simultaneous reading and writing. It is also impossible to have the device open more than once at a time.

Even opening the device has side effects. Data structures are allocated, and PM_HIBERNATION_PREPARE / PM_RESTORE_PREPARE chains are called.

The ioctl() commands recognized by the device are:

SNAPSHOT_FREEZE
: freeze user space processes (the current process is not frozen); this is required for SNAPSHOT_CREATE_IMAGE and SNAPSHOT_ATOMIC_RESTORE to succeed

SNAPSHOT_UNFREEZE
: thaw user space processes frozen by SNAPSHOT_FREEZE

SNAPSHOT_CREATE_IMAGE
: create a snapshot of the system memory; the last argument of ioctl() should be a pointer to an int variable, the value of which will indicate whether the call returned after creating the snapshot (1) or after restoring the system memory state from it (0) (after resume the system finds itself finishing the SNAPSHOT_CREATE_IMAGE ioctl() again); after the snapshot has been created the read() operation can be used to transfer it out of the kernel

SNAPSHOT_ATOMIC_RESTORE
: restore the system memory state from the uploaded snapshot image; before calling it you should transfer the system memory snapshot back to the kernel using the write() operation; this call will not succeed if the snapshot image is not available to the kernel

SNAPSHOT_FREE
: free memory allocated for the snapshot image

SNAPSHOT_PREF_IMAGE_SIZE
: set the preferred maximum size of the image (the kernel will do its best to ensure the image size will not exceed this number, but if it turns out to be impossible, the kernel will create the smallest image possible)

SNAPSHOT_GET_IMAGE_SIZE
: return the actual size of the hibernation image (the last argument should be a pointer to a loff_t variable that will contain the result if the call is successful)

SNAPSHOT_AVAIL_SWAP_SIZE
: return the amount of available swap in bytes (the last argument should be a pointer to a loff_t variable that will contain the result if the call is successful)

SNAPSHOT_ALLOC_SWAP_PAGE
: allocate a swap page from the resume partition (the last argument should be a pointer to a loff_t variable that will contain the swap page offset if the call is successful)

SNAPSHOT_FREE_SWAP_PAGES
: free all swap pages allocated by SNAPSHOT_ALLOC_SWAP_PAGE

SNAPSHOT_SET_SWAP_AREA
: set the resume partition and the offset (in <PAGE_SIZE> units) from the beginning of the partition at which the swap header is located (the last ioctl() argument should point to a struct resume_swap_area, as defined in kernel/power/suspend_ioctls.h, containing the resume device specification and the offset); for swap partitions the offset is always 0, but it is different from zero for swap files (see Documentation/power/swsusp-and-swap-files.rst for details).

SNAPSHOT_PLATFORM_SUPPORT
: enable/disable the hibernation platform support, depending on the argument value (enable, if the argument is nonzero)

SNAPSHOT_POWER_OFF
: make the kernel transition the system to the hibernation state (eg. ACPI S4) using the platform (eg. ACPI) driver

SNAPSHOT_S2RAM
: suspend to RAM; using this call causes the kernel to immediately enter the suspend-to-RAM state, so this call must always be preceded by the SNAPSHOT_FREEZE call and it is also necessary to use the SNAPSHOT_UNFREEZE call after the

system wakes up. This call is needed to implement the suspend-to-both mechanism in which the suspend image is first created, as though the system had been suspended to disk, and then the system is suspended to RAM (this makes it possible to resume the system from RAM if there's enough battery power or restore its state on the basis of the saved suspend image otherwise)

The device's read() operation can be used to transfer the snapshot image from the kernel. It has the following limitations:

- you cannot read() more than one virtual memory page at a time
- read()s across page boundaries are impossible (ie. if you read() 1/2 of a page in the previous call, you will only be able to read() **at most** 1/2 of the page in the next call)

The device's write() operation is used for uploading the system memory snapshot into the kernel. It has the same limitations as the read() operation.

The release() operation frees all memory allocated for the snapshot image and all swap pages allocated with SNAPSHOT_ALLOC_SWAP_PAGE (if any). Thus it is not necessary to use either SNAPSHOT_FREE or SNAPSHOT_FREE_SWAP_PAGES before closing the device (in fact it will also unfreeze user space processes frozen by SNAPSHOT_UNFREEZE if they are still frozen when the device is being closed).

Currently it is assumed that the userland utilities reading/writing the snapshot image from/to the kernel will use a swap partition, called the resume partition, or a swap file as storage space (if a swap file is used, the resume partition is the partition that holds this file). However, this is not really required, as they can use, for example, a special (blank) suspend partition or a file on a partition that is unmounted before SNAPSHOT_CREATE_IMAGE and mounted afterwards.

These utilities MUST NOT make any assumptions regarding the ordering of data within the snapshot image. The contents of the image are entirely owned by the kernel and its structure may be changed in future kernel releases.

The snapshot image MUST be written to the kernel unaltered (ie. all of the image data, metadata and header MUST be written in _exactly_ the same amount, form and order in which they have been read). Otherwise, the behavior of the resumed system may be totally unpredictable.

While executing SNAPSHOT_ATOMIC_RESTORE the kernel checks if the structure of the snapshot image is consistent with the information stored in the image header. If any inconsistencies are detected, SNAPSHOT_ATOMIC_RESTORE will not succeed. Still, this is not a fool-proof mechanism and the userland utilities using the interface SHOULD use additional means, such as checksums, to ensure the integrity of the snapshot image.

The suspending and resuming utilities MUST lock themselves in memory, preferably using mlockall(), before calling SNAPSHOT_FREEZE.

The suspending utility MUST check the value stored by SNAPSHOT_CREATE_IMAGE in the memory location pointed to by the last argument of ioctl() and proceed in accordance with it:

1. If the value is 1 (ie. the system memory snapshot has just been created and the system is ready for saving it):
    a. The suspending utility MUST NOT close the snapshot device _unless_ the whole suspend procedure is to be cancelled, in which case, if the snapshot image has already been saved, the suspending utility SHOULD destroy it, preferably by zapping its header. If the suspend is not to be cancelled, the system MUST be powered off or rebooted after the snapshot image has been saved.
    b. The suspending utility SHOULD NOT attempt to perform any file system operations (including reads) on the file systems that were mounted before SNAPSHOT_CREATE_IMAGE has been called. However, it MAY mount a file system that was not mounted at that time and perform some operations on it (eg. use it for saving the image).
2. If the value is 0 (ie. the system state has just been restored from the snapshot image), the suspending utility MUST close the snapshot device. Afterwards it will be treated as a regular userland process, so it need not exit.

The resuming utility SHOULD NOT attempt to mount any file systems that could be mounted before suspend and SHOULD NOT attempt to perform any operations involving such file systems.

For details, please refer to the source code.