Virtual Accelerator Switchboard (VAS) userspace API

Introduction

Power9 processor introduced Virtual Accelerator Switchboard (VAS) which allows both userspace and kernel communicate to co-processor (hardware accelerator) referred to as the Nest Accelerator (NX). The NX unit comprises of one or more hardware engines or co-processor types such as 842 compression, GZIP compression and encryption. On power9, userspace applications will have access to only GZIP Compression engine which supports ZLIB and GZIP compression algorithms in the hardware.

To communicate with NX, kernel has to establish a channel or window and then requests can be submitted directly without kernel involvement. Requests to the GZIP engine must be formatted as a co-processor Request Block (CRB) and these CRBs must be submitted to the NX using COPY/PASTE instructions to paste the CRB to hardware address that is associated with the engine's request queue.

The GZIP engine provides two priority levels of requests: Normal and High. Only Normal requests are supported from userspace right now.

This document explains userspace API that is used to interact with kernel to setup channel / window which can be used to send compression requests directly to NX accelerator.

Overview

Application access to the GZIP engine is provided through /dev/crypto/nx-gzip device node implemented by the VAS/NX device driver. An application must open the /dev/crypto/nx-gzip device to obtain a file descriptor (fd). Then should issue VAS_TX_WIN_OPEN ioctl with this fd to establish connection to the engine. It means send window is opened on GZIP engine for this process. Once a connection is established, the application should use the mmap() system call to map the hardware address of engine's request queue into the application's virtual address space.

The application can then submit one or more requests to the engine by using copy/paste instructions and pasting the CRBs to the virtual address (aka paste_address) returned by mmap(). User space can close the established connection or send window by closing the file descriptior (close(fd)) or upon the process exit.

Note that applications can send several requests with the same window or can establish multiple windows, but one window for each file descriptor.

Following sections provide additional details and references about the individual steps.

NX-GZIP Device Node

There is one /dev/crypto/nx-gzip node in the system and it provides access to all GZIP engines in the system. The only valid operations on /dev/crypto/nx-gzip are:

- open() the device for read and write.
- issue VAS_TX_WIN_OPEN ioctl
- mmap() the engine's request queue into application's virtual address space (i.e. get a paste_address for the co-processor engine).
- close the device node.

Other file operations on this device node are undefined.

Note that the copy and paste operations go directly to the hardware and do not go through this device. Refer COPY/PASTE document for more details.

Although a system may have several instances of the NX co-processor engines (typically, one per P9 chip) there is just one /dev/crypto/nx-gzip device node in the system. When the nx-gzip device node is opened, Kernel opens send window on a suitable instance of NX accelerator. It finds CPU on which the user process is executing and determine the NX instance for the corresponding chip on which this CPU belongs.

Applications may chose a specific instance of the NX co-processor using the vas_id field in the VAS_TX_WIN_OPEN ioctl as detailed below.

A userspace library libnxz is available here but still in development:

https://github.com/abalib/power-gzip

Applications that use inflate / deflate calls can link with libnxz instead of libz and use NX GZIP compression without any modification.

Open /dev/crypto/nx-gzip

The nx-gzip device should be opened for read and write. No special privileges are needed to open the device. Each window corresponds to one file descriptor. So if the userspace process needs multiple windows, several open calls have to be issued.

See open(2) system call man pages for other details such as return values, error codes and restrictions.

VAS_TX_WIN_OPEN ioctl

Applications should use the VAS_TX_WIN_OPEN ioctl as follows to establish a connection with NX co-processor engine:

```
struct vas_tx_win_open_attr {
        __u32   version;
        __s16   vas_id; /* specific instance of vas or -1
                           for default */
        __u16   reserved1;
        __u64   flags;  /* For future use */
        __u64   reserved2[6];
};
```

version:
: The version field must be currently set to 1.

vas_id:
: If '-1' is passed, kernel will make a best-effort attempt to assign an optimal instance of NX for the process. To select the specific VAS instance, refer "Discovery of available VAS engines" section below.

flags, reserved1 and reserved2[6] fields are for future extension and must be set to 0.

The attributes attr for the VAS_TX_WIN_OPEN ioctl are defined as follows:

```
#define VAS_MAGIC 'v'
#define VAS_TX_WIN_OPEN _IOW(VAS_MAGIC, 1,
                            struct vas_tx_win_open_attr)

struct vas_tx_win_open_attr attr;
rc = ioctl(fd, VAS_TX_WIN_OPEN, &attr);
```

The VAS_TX_WIN_OPEN ioctl returns 0 on success. On errors, it returns -1 and sets the errno variable to indicate the error.

Error conditions:

EINVAL	fd does not refer to a valid VAS device.
EINVAL	Invalid vas ID
EINVAL	version is not set with proper value
EEXIST	Window is already opened for the given fd
ENOMEM	Memory is not available to allocate window
ENOSPC	System has too many active windows (connections) opened
EINVAL	reserved fields are not set to 0.

See the ioctl(2) man page for more details, error codes and restrictions.

mmap() NX-GZIP device

The mmap() system call for a NX-GZIP device fd returns a paste_address that the application can use to copy/paste its CRB to the hardware engines.

```
paste_addr = mmap(addr, size, prot, flags, fd, offset);
```

Only restrictions on mmap for a NX-GZIP device fd are:

- size should be PAGE_SIZE
- offset parameter should be 0ULL

Refer to mmap(2) man page for additional details/restrictions. In addition to the error conditions listed on the mmap(2) man page, can also fail with one of the following error codes:

EINVAL	fd is not associated with an open window (i.e mmap() does not follow a successful call to the VAS_TX_WIN_OPEN ioctl).
EINVAL	offset field is not 0ULL.

Discovery of available VAS engines

Each available VAS instance in the system will have a device tree node like /proc/device-tree/vas@* or /proc/device-

tree/xscom@*/vas@*. Determine the chip or VAS instance and use the corresponding ibm,vas-id property value in this node to select specific VAS instance.

Copy/Paste operations

Applications should use the copy and paste instructions to send CRB to NX. Refer section 4.4 in PowerISA for Copy/Paste instructions: https://openpowerfoundation.org/?resource_lib=power-isa-version-3-0

CRB Specification and use NX

Applications should format requests to the co-processor using the co-processor Request Block (CRBs). Refer NX-GZIP user's manual for the format of CRB and use NX from userspace such as sending requests and checking request status.

NX Fault handling

Applications send requests to NX and wait for the status by polling on co-processor Status Block (CSB) flags. NX updates status in CSB after each request is processed. Refer NX-GZIP user's manual for the format of CSB and status flags.

In case if NX encounters translation error (called NX page fault) on CSB address or any request buffer, raises an interrupt on the CPU to handle the fault. Page fault can happen if an application passes invalid addresses or request buffers are not in memory. The operating system handles the fault by updating CSB with the following data:

```
csb.flags = CSB_V;
csb.cc = CSB_CC_FAULT_ADDRESS;
csb.ce = CSB_CE_TERMINATION;
csb.address = fault_address;
```

When an application receives translation error, it can touch or access the page that has a fault address so that this page will be in memory. Then the application can resend this request to NX.

If the OS can not update CSB due to invalid CSB address, sends SEGV signal to the process who opened the send window on which the original request was issued. This signal returns with the following siginfo struct:

```
siginfo.si_signo = SIGSEGV;
siginfo.si_errno = EFAULT;
siginfo.si_code = SEGV_MAPERR;
siginfo.si_addr = CSB adress;
```

In the case of multi-thread applications, NX send windows can be shared across all threads. For example, a child thread can open a send window, but other threads can send requests to NX using this window. These requests will be successful even in the case of OS handling faults as long as CSB address is valid. If the NX request contains an invalid CSB address, the signal will be sent to the child thread that opened the window. But if the thread is exited without closing the window and the request is issued using this window. the signal will be issued to the thread group leader (tgid). It is up to the application whether to ignore or handle these signals.

NX-GZIP User's Manual: https://github.com/libnxz/power-gzip/blob/master/doc/power_nx_gzip_um.pdf

Simple example

```
int use_nx_gzip()
{
        int rc, fd;
        void *addr;
        struct vas_setup_attr txattr;

        fd = open("/dev/crypto/nx-gzip", O_RDWR);
        if (fd < 0) {
                fprintf(stderr, "open nx-gzip failed\n");
                return -1;
        }
        memset(&txattr, 0, sizeof(txattr));
        txattr.version = 1;
        txattr.vas_id = -1
        rc = ioctl(fd, VAS_TX_WIN_OPEN,
                        (unsigned long)&txattr);
        if (rc < 0) {
                fprintf(stderr, "ioctl() n %d, error %d\n",
                                rc, errno);
                return rc;
        }
        addr = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
                        MAP_SHARED, fd, 0ULL);
        if (addr == MAP_FAILED) {
                fprintf(stderr, "mmap() failed, errno %d\n",
                                errno);
                return -errno;
```

```
        }
        do {
                //Format CRB request with compression or
                //uncompression
                // Refer tests for vas_copy/vas_paste
                vas_copy((&crb, 0, 1);
                vas_paste(addr, 0, 1);
                // Poll on csb.flags with timeout
                // csb address is listed in CRB
        } while (true)
        close(fd) or window can be closed upon process exit
}
```

Refer https://github.com/libnxz/power-gzip for tests or more use cases.