ATLAS Internal Note
ATL-D-ES-0006
Version 1.10
2 February 2021

A package for the allocation of contiguous memory on Linux systems

Author : M. Joos

Comments and queries to Markus Joos, CERN
+41 22 767 2364
Markus.Joos@cern.ch

Abstract

This note describes a software package (Linux driver and library) for the allocation of contiguous buffers from user processes on standard Linux kernels.

1 Introduction

The purpose of this Linux driver and library is to provide the user with a means of allocating buffers of contiguous memory for DMA operations and other purposes.

Linux offers several methods for the allocation of contiguous memory that all have their advantages and disadvantages. An evaluation of CMA (contiguous memory allocation) in 2018 has concluded that this solution does not (yet) meet the requirements of the FELIX project (partially because the support provided by IT and RedHat was poor and documentation missing). The cmem_rcc package, that is well established in ATLAS, will therefore be the baseline solution for the time to come.

Internally cmem_rcc is based on the system calls `alloc_pages_node()` and `alloc_pages()` that allow to allocate contiguous buffers from a defined NUMA node or without taking NUMA into account.
The main limitation of `alloc_pages(_node)` is it cannot allocate memory buffers above a certain size. For current kernels, this limit is 4 MB. Even though this limit can be changed via a kernel configuration parameter, it is not recommended to do so and getting buffers in the GB range is not possible anyway.

cmem_rcc uses `alloc_pages(_node)` in two different ways. The user can (via library and driver) call `alloc_pages(_node)` at run-time to get individual buffers of up to 4 MB.
Alternatively, an algorithm based on `alloc_pages_node` can be used at the time of loading the driver to allocate pages until a contiguous area of memory with a user-defined size has been found. Applications can then obtain fragments of this large buffer.

The cmem_rcc driver has been developed paying special attention to issues that may arise on SMP systems and mechanisms such as spinlocks were used in order to prevent race conditions. The driver and library support 64-bit CPUs and Linux kernels.

A number of special cases have been taken into account by the user:

- The allocation of RAM accessible with 32bit PCI addresses (i.e. below the 4GB limit) is possible. This feature has been added for the use on VMEbus SBCs that are based on the Universe ASIC
- The allocation of buffers from a defined NUMA ID is supported. For technical reasons it is not possible to pre-allocate at driver load time a buffer that is larger than the size of one NUMA zone.
- The memory can be mapped alternatively in a way that makes it usable to RDM drivers (as used in FELIX)

2 Application Program Interface

2.1 Overview

The following list is an overview of the functions provided by the cmem_rcc API:

- CMEM_Open, CMEM_Open_Nopage
- CMEM_SegmentAllocate, CMEM_SegmentAllocateNuma
- CMEM_GFPBPASegmentAllocate, CMEM_BPASegmentAllocate, CMEM_NumaSegmentAllocate
- CMEM_SegmentFree, CMEM_BPASegmentFree, CMEM_GFPBPASegmentFree, CMEM_SegmentUnlockAndFree
- CMEM_SegmentSize
- CMEM_SegmentLock, CMEM_SegmentUnlock
- CMEM_SegmentGet
- CMEM_SegmentVirtualAddress
- CMEM_SegmentPhysicalAddress
- CMEM_Dump
- CMEM_Close

The following remarks apply to all functions defined in the API:

- This package is based on the “rcc_error” error handling package. The rcc_error package basically is a renamed copy of the iom_error[4] package
- All functions of this package return CMEM_RCC_SUCCESS on success.
- The type (CMEM_Error_code_t) of the function return value defaults to “unsigned int”.

CMEM_Open()

Synopsis

```
#include <cmem_rcc.h>
CMEM_ErrorCode_t CMEM_Open(void);
```

Parameters

None

Description

This function opens the package and the device file (/dev/cmem_rcc) of the driver. Linux knows several methods for the provision of user virtual addresses. If you are using this function to open the library, these addresses will be obtained via the remap_pfn_range() system call. This is the “traditional” method of cmem_rcc.

Return Values

CMEM_RCC_ERROR_FAIL	Failed to open the rcc_error package
CMEM_RCC_FILE	Failed to open the file /dev/cmem_rcc

CMEM_OpenNopage()

Synopsis

```
#include <cmem_rcc.h>
CMEM_ErrorCode_t CMEM_OpenNopage(void);
```

Parameters

None

Description

This function opens the package and the device file (/dev/cmem_rcc_2) of the driver. In contrast to CMEM_Open, this function uses the “nopage” algorithm to create user virtual addresses. It has been added to support RDMA data transfers with Mellanox NICs in the FELIX project but may also be useful for other use cases.

Return Values

CMEM_RCC_ERROR_FAIL	Failed to open the rcc_error package
CMEM_RCC_FILE	Failed to open the file /dev/cmem_rcc_2

CMEM_Close()

Synopsis

```
#include <cmem_rcc.h>
CMEM_ErrorCode_t CMEM_Close(void);
```

Parameters

None

Description

This function closes the package and releases the access to the device file. Segments that have not been freed will be de-allocated unless they have been locked.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened

CMEM_SegmentAllocate()

Synopsis

```
#include <cmem_rcc.h>
CMEM_ErrorCode_t CMEM_SegmentAllocate(u_int size, char
*name, int *segment_identifier);
```

Parameters

u_int size	in	The size of the contiguous buffer in bytes
char *name	in	The name of the buffer. The string must not contain more than CMEM_MAX_NAME characters (see cmem_rcc.h)
int *segment_identifier	out	A handle to the buffer. To be used with the functions described below

Description

This function allocates a contiguous buffer by means of the `alloc_pages()` system call. The size of the buffer has to be specified via the value of *size*. The function guarantees that the buffer will at least be *size* bytes long; it may, however, be larger. See also Appendix A.

Please note: This function allocates memory from the pool managed by Linux. As it uses the `alloc_pages()` system call it inherits the limitations of this call. Depending on the kernel, the `size` is therefore limited to 2 or 4 MB. If you need a larger, contiguous buffer, use the function `CMEM_GFPBPASegmentAllocate()`.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This indicates also that no memory has been allocated
CMEM_RCC_MMAP	The buffer was allocated but no user virtual address could be assigned
CMEM_RCC_TOOBIG	A buffer of the requested size can never be allocated (see appendix A)
CMEM_RCC_NOSIZE	The parameter *size* is zero

CMEM_SegmentAllocateNuma()

Synopsis

```
#include <cmem_rcc.h>
CMEM_ErrorCode_t CMEM_SegmentAllocateNuma(u_int size, u_int
numa_id, char *name, int *segment_identifier);
```

Parameters

u_int size	in	The size of the contiguous buffer in bytes
u_int numa_id	in	The ID of the NUMA segment from which the memory is to be allocated
char *name	in	The name of the buffer. The string must not contain more than CMEM_MAX_NAME characters (see cmem_rcc.h)
int *segment_identifier	out	A handle to the buffer. To be used with the functions described below

Description

This function works like `CMEM_SegmentAllocate()` in the sense that is dynamically allocates a contiguous buffer but it uses the `alloc_pages_node()` system call and therefore operates on a defined NUMA zone. The size of the buffer has to be specified via the value of *size*. The function guarantees that the buffer will at least be *size* bytes long; it may, however, be larger. See also Appendix A.

Caveat: Don't confuse this function with `CMEM_NumaSegmentAllocate()`

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This indicates also that no memory has been allocated
CMEM_RCC_MMAP	The buffer was allocated but no user virtual address could be assigned
CMEM_RCC_TOOBIG	A buffer of the requested size can never be allocated (see appendix A)
CMEM_RCC_NOSIZE	The parameter *size* is zero

CMEM_GFPBPASegmentAllocate()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_GFPBPASegmentAllocate(u_int size,
char *name, int *segment_identifier);
```

Parameters

u_int size	in	The size of the contiguous buffer in bytes
char *name	in	The name of the buffer. The string must not contain more than CMEM_MAX_NAME characters (see cmem_rcc.h)
int *segment_identifier	out	A handle to the buffer. To be used with the functions described below

Description

This function allocates a contiguous buffer from a pre-allocated memory pool.

The function only works if the driver was loaded with appropriate parameters (see chapter 4)

The size of the buffer has to be specified via the value of *size*. The function guarantees that the buffer will at least be *size* bytes long; it may, however, be larger. See also Appendix A.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This indicates also that no memory has been allocated
CMEM_RCC_MMAP	The buffer was allocated but no user virtual address could be assigned

CMEM_NumaSegmentAllocate()

Synopsis

```
#include <cmem_rcc.h>;

CMEM_ErrorCode_t CMEM_NumaSegmentAllocate(u_int size, u_int
numa_id, char *name, int *segment_identifier);
```

Parameters

u_int size	in	The size of the contiguous buffer in bytes
u_int numa_id	in	The ID of the NUMA node from which the memory is to be allocated
char *name	in	The name of the buffer. The string must not contain more than CMEM_MAX_NAME characters (see cmem_rcc.h)
int *segment_identifier	out	A handle to the buffer. To be used with the functions described below

Description

This function allocates a contiguous buffer from a pre-allocated memory pool of a defined NUMA node.

The function only works if the driver was loaded with appropriate parameters (see chapter 4)

The size of the buffer has to be specified via the value of *size*. The function guarantees that the buffer will at least be *size* bytes long; it may, however, be larger. See also Appendix A.

Caveat: Don't confuse this function with `CMEM_SegmentAllocateNuma()`

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This indicates also that no memory has been allocated
CMEM_RCC_MMAP	The buffer was allocated but no user virtual address could be assigned

CMEM_BPASegmentAllocate()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_BPASegmentAllocate(u_int size,
char *name, int *segment_identifier);
```

Parameters

u_int size	in	The size of the contiguous buffer in bytes
char *name	in	The name of the buffer. The string must not contain more than CMEM_MAX_NAME characters (see cmem_rcc.h)
int *segment_identifier	out	A handle to the buffer. To be used with the functions described below

Description

This function is provided for compatibility purposes. It maps directly onto CMEM_GFPBPASegmentAllocate().

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This indicates also that no memory has been allocated
CMEM_RCC_MMAP	The buffer was allocated but no user virtual address could be assigned

CMEM_SegmentFree(), CMEM_BPASegmentFree(), CMEM_GFPBPASegmentFree()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentFree(int segment_identifier);
CMEM_ErrorCode_t CMEM_GFPBPASegmentFree(int
segment_identifier);
CMEM_ErrorCode_t CMEM_BPASegmentFree(int
segment_identifier);
```

Parameters

int segment_identifier	in	The handle describing the buffer to be returned

Description

These functions return a buffer to the pool of free memory.
It does not matter which function has been used to allocate that buffer. The actual job is done by `CMEM_SegmentFree()`. The other two functions have become obsolete but are kept in the API for compatibility with applications that still use them.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_MUNMAP	The user virtual address could not be de-allocated
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This indicates that the memory could not be freed
CMEM_RCC_GETP	The value passed for *segment_identifier* does not refer to a known buffer
CMEM_RCC_OVERFLOW	The library has been compiled without support for BPA

CMEM_SegmentUnlockAndFree(char* name)

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentUnlockAndFree(char* name)
```

Parameters

char* name	in	The name of the buffer to be returned

Description

This function returns a buffer to the pool of free memory. The buffer is identified by the name rather than descriptor.

Note: The only constraint that cmem_rcc puts on the names of buffers is that they must be at most 40 characters long. It is possible to give the same name to several buffers. Whenever possible, a buffer should be returned by handle (see functions above) because the handle identifies the buffer in an unambiguous way. If several buffers with the same name exist, this function will return the one with the lowest handle. In case a buffer has been locked, it will be automatically unlocked. The function will can be called in a loop to return buffers with the same name. The error code "CMEM_RCC_ILLNAME" will be returned if no more buffers of the specified name exist.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_MUNMAP	The user virtual address could not be de-allocated
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This indicates that the memory could not be freed
CMEM_RCC_ILLNAME	A buffer with the specified name did not exist

CMEM_SegmentVirtualAddress()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentVirtualAddress(int
segment_identifier, u_long *virtual_address);
```

Parameters

int segment_identifier	in	A handle describing a buffer
u_long *virtual_address	out	The user virtual address of the first byte of the buffer

Description

This function returns the "user-space virtual address" of a buffer.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_GETP	The value passed for *segment_identifier* does not point to an open buffer

CMEM_SegmentSize()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentSize(int segment_identifier,
u_int *actual_size);
```

Parameters

int segment_identifier	in	A handle describing a buffer
u_int *actual_size	out	The actual size of the buffer in bytes

Description

This function returns the actual size of a buffer. The actual size may be larger then the requested size due to buffer granularity. See CMEM_[BPA]SegmentAllocate().

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_GETP	The value passed for *segment_identifier* does not point to an open buffer

CMEM_SegmentPhysicalAddress()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentPhysicalAddress(int
segment_identifier, u_long *physical_address);
```

Parameters

int segment_identifier	in	A handle describing a buffer
u_long *physical_address	out	The physical address of the first byte of the buffer

Description

This function returns the physical address of the first word of a buffer. This address is identical to the PCI address of the buffer.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_GETP	The value passed for *segment_identifier* does not point to an open buffer

CMEM_Dump()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_Dump(void);
```

Parameters

None.

Description

This function dumps the system parameters of all currently open buffers. You can get the same information with the command “more /proc/cmem_rcc”.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver

CMEM_SegmentLock ()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentLock(int segment_identifier);
```

Parameters

int segment_identifier	in	A handle describing a buffer

Description

This function locks a buffer. Once a buffer is locked it can no longer be freed with the CMEM_BPASegmentFree() or CMEM_BPASegmentFree() functions. It also does not get de-allocated by the garbage collector in the driver if the application that created the buffer exits. Usually buffers should not be locked as this may lead to memory leaks. This function is provided for the rare case were, e.g. during the boot process, one application has to allocate a buffer that will be used by other applications later on.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This could indicate that *segment_identifier* does not refer to a currently open buffer

CMEM_SegmentUnlock()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentUnlock(int
segment_identifier);
```

Parameters

int segment_identifier	in	A handle describing a buffer

Description

This function unlocks a buffer.

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This could indicate that *segment_identifier* does not refer to a currently open buffer

CMEM_SegmentGet ()

Synopsis

```
#include <cmem_rcc.h>;
CMEM_ErrorCode_t CMEM_SegmentGet (int segment_identifier,
cmem_rcc_t *params);
```

Parameters

int segment_identifier	in	A handle describing a buffer
cmem_rcc_t *params	out	A structure with the parameters of s buffer

Description

This function can be used to retrieve the parameters of one buffer. The structure cmem_rcc_t is defined in cmem_rcc_common.h:

```
typedef struct
{
  unsigned int paddr;   //The physical address the buffer
  unsigned int uaddr;   //The used virtual address the buffer
  unsigned long kaddr;  //The kernel virtual address the buffer
  unsigned int size;    //The size of the buffer
  unsigned int order;   //The encoded size of GFP buffers
  unsigned int locked;  //A flag indicating if the buffer is locked
  unsigned int type;    //The type of the buffer (TYPE_BPA or TYPE_GFP)
  unsigned int handle;  //The segment identifier
  char name[CMEM_MAX_NAME]; //The name of the buffer
} cmem_rcc_t;
```

Return Values

CMEM_RCC_NOTOPEN	The package has not yet been opened
CMEM_RCC_IOCTL	Error from a call to the cmem_rcc driver. This could indicate that *segment_identifier* does not refer to a currently open buffer

4 Software distribution and installation

The source code of the cmem_rcc library can be found in the ATLAS TDAQ package "cmem_rcc". It is located in the TDAQ repository at https://gitlab.cern.ch/atlas-tdaq-software/cmem_rcc. The driver can be found in the package ROSRCDdrivers at https://gitlab.cern.ch/atlas-tdaq-software/ROSRCDdrivers/

The source code in the cmem_rcc package allows building the library and some test programs.

The installation of the driver should be performed with the script "drivers_tdaq" from the ROSRCDdrivers package which is meant to be copied to /etc/rc.d/init.d and called via a link from e.g. /etc/rc.d/rc[3/5].d/S95drivers_tdaq. At ATLAS P1 the drivers are installed with a script called "atlas_tdaq_drivers" which is under the control of the sysadmins.

Depending on which technique will be used for the pre-allocation of memory, the driver needs additional parameters.

Pre-allocation with `__alloc_pages_node`:

Parameter name: gfpbpa_size
Description: The amount of RAM in MB that will be used for the internal-BPA. On systems with more than one NUMA zone, the value of gfpbpa_size is limited a (somewhat less) than the size of a single NUMA zone.

Parameter name: gfpbpa_quantum
Description: The size (in MB) of a page allocated via `alloc_pages_node()` for the internal-BPA. This parameter is optional. The default quantum is 1 MB

Parameter name: numa_zones
Description: This parameter must be set to the number of NUMA zones that are present in the computer. A buffer of size "gfpbpa_size" will be allocated from each of these NUMA zones.

Examples:
Allocate 32 MB with alloc_pages_node:

```
/sbin/insmod cmem_rcc-2.6.9-55.EL.cern.ko gfpbpa_size=32
```

You can display the definition of the driver parameters with "/sbin/modinfo <driver>". E.g.:

```
/sbin/modinfo cmem_rcc-2.6.9-55.EL.cern.ko
```

Once the driver is running, users can monitor its activity via the file /proc/cmem_rcc.

Appendix A: Parametrization

The number of buffers (of either type) is limited by the size of some tables in the driver. The dimension of these tables is set by the parameter MAX_BUFFS in cmem_rcc_drv.h. If the default value (currently 1000) is not sufficient it can be increased to the required number of buffers. A change to this parameter requires a re-compilation of the entire package.

Directly in the source code of the driver two additional parameters are located that may have to be adapted to the target system. The parameter MAX_GFPBPA_SIZE (currently set to 256 GB) has to larger or equal than the amount of memory installed in the target computer. The second parameter, MAX_NUMA_ZONES (currently set to 4) defines how many NUMA IDs the driver can handle.

Appendix B: Properties of `alloc_pages(_node)`

This chapter gives some background information on the implementation of the CMEM_RCC API based on the `alloc_pages(_node)` system calls [5] of Linux. As it inherits some limitations (buffer granularity and max. size) from this lowest level of memory allocation, it is necessary to understand some of the details in order to develop efficient code.

This functions `CMEM_SegmantAllocate(Numa)` pass the "size" parameter to the driver where it is converted to an "order" which is defined as the base two logarithm of the number of pages to be allocated. The default page size of Linux is 4KB. Examples:

Range of *size*	Resulting value of *order*	Resulting actual buffer size [in bytes]
1..4096 bytes	0	4096
4097..8192 bytes	1	8192
8193..16348 bytes	2	16348
> 2 Mbytes	out of range	0

The maximum value of *order* for current kernels is 10 (corresponding to 1024 pages = 4 MB). The command "more /proc/buddyinfo" tells you how many blocks of each order are available. The conversion from *size* to *order* guarantees that the buffer will at least be *size* bytes long; it may, however, be larger. The function will fail for *size* > 4 MB but can also fail for smaller values if a contiguous buffer of the requested size cannot be found.

5 References

[1] http://atddoc.cern.ch/Atlas/Notes/153/Note153-1.html
[2] http://www.polyware.nl/~middelin/En/hob-v4l.html#bigphysarea
[3] http://atddoc.cern.ch/Atlas/Notes/136/Note136-1.html
[4] http://atddoc.cern.ch/Atlas/Notes/051/Note051-1.html
[5] http://www.xml.com/ldd/chapter/book/index.html