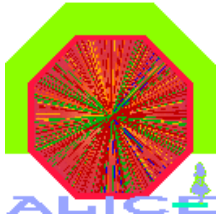


**EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH**  
**European Laboratory for Particle Physics**



**CERN**

CH-1211 Genève 23  
Switzerland

**Internal Note/**

ALICE reference number

ALICE-INT-2010-XXX Version 2

Institute reference number

[-]

Date of last change

2010-09-15

## **Physem package migration guide**

### **Authors:**

E. Dénes (KFKI-RMKI, Budapest, Hungary)

### **Abstract:**

The purpose of this document is to assist the software developers during the migration from Linux SLC4 32 bit to Linux SLC5 32/64 bit of existing code making use of the physem package.

# 1. Introduction

The DATE phymem package gives access to a fixed part of the physical memory of a Linux machine for un-paged, un-swapped physical I/O. This document describes how to migrate software that use the phymem package between Linux SLC4 32 bit and Linux SLC5 32 and 64 bit.

A key issue in the migration concerns the physical memory space of the PCs. As shown in Fig.1, the memory space is used to access several resources of the machine (BIOS, I/O, etc) and one area indicated as “Free Memory” is used for the physical memory. In a 32 bit machine, it consists of a single area with a contiguous memory address space whereas it is split in two distinct areas on 64 bit machines.

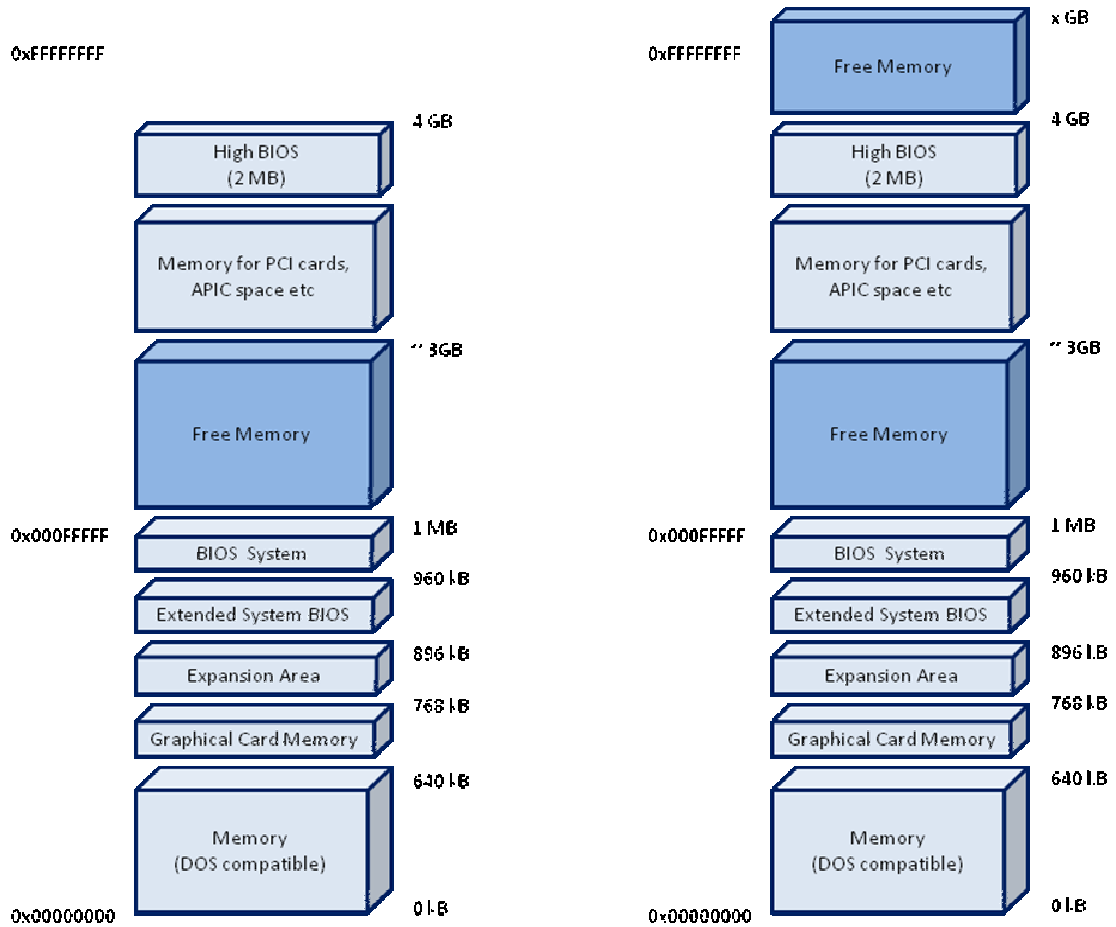


Fig. 1 The layout of the physical address space of 32 bit (left) and 64 bit (right) machines.

This difference has obviously a major impact on the addressing performed to access the physical memory. The software package phymem used to access the physical memory had to be modified to take into account this change.

## 2. Migration procedure

In the case of 64 bit machine architecture the physmem memory can be divided into separate areas, so called memory zones. The reason of this is that the BIOS assigns some memory areas to devices such as the video memory just below the 4 GB limit (generally between 3.3 GB and 4 GB). This part of the memory cannot be used by the physmem devices. Therefore one of the physmem devices (generally `/dev/physmem1`) is divided into two zones. Each physmem device is accessed through a continuous physical address space but the “gap” addresses between the zones cannot be used.

The file operation `mmap()` can map the physmem devices in two different way: *usable* or *full* mapping. In the first case the user addresses are continuous for the usable part of the device. This means that the last user address of a memory zone is immediately followed by the first user address of the next zone. There is no gap in the user’s space. To calculate the physical addresses of a given user address one has to take into account the physical gap between the zones.

In the case of *full* mapping the user address space is continuous, i.e. the gap area has user address as well. This facilitate the calculation of physical address of a given user address. On the other hand the user can accidentally read or write from and to the forbidden gap area, which can lead to a system crash.

The two mapping methods can be selected by an `ioctl()` call before calling `mmap()` operation. The physical addresses and sizes of the memory zones can also be received by an `ioctl()` call.

If the user limits himself to use physmem below 3.3 GB, or loads the physmem driver so the device starts above 4GB then all the old program binaries can be used as is. Only the new, 64 bit physmem driver has to be loaded. All pointers and associated arithmetic’s have to be changed to carry 64 bits though.

## 3. API

We will now review the API, highlighting the new features related to 64 bit addressing.

### 3.1. Common definitions and variables

The following definitions and variables are referenced by the physmem API:

```
int err;
int physmem_fd;
void *addr_user_physmem;
unsigned long phys_addr_physmem;
unsigned long phys_size_physmem;
unsigned long phys_full_size_physmem;
unsigned int phys_num_zones;
struct memZoneStruct *phys_zones;
```

### 3.2. Open the device

Common call for both 32 and 64 bit:

```
physmem_fd = open( "/dev/physmemx", O_RDWR );
```

### 3.3. Get physical start address

Common 32/64 bit:

```
err = ioctl( physmem_fd,
             PHYSMEM_GETADDR,
             &phys_addr_physmem );
```

### 3.4. Get available size in bytes

This call should be used to select between *usable* or *full* mapping.

For *usable* mapping call `ioctl()` with `PHYSMEM_GETSIZE` parameter, as in the case of 32 bit package:

```
err = ioctl( physmem_fd,
             PHYSMEM_GETSIZE,
             &phys_size_physmem );
```

For *full* mapping use `PHYSMEM_GETFULLSIZE`:

```
err = ioctl( physmem_fd,
             PHYSMEM_GETFULLSIZE,
             &phys_full_size_physmem );
```

### 3.5. Get the number of memory zones

New call, to be used only for 64 bit addressing:

```
err = ioctl( physmem_fd,
             PHYSMEM_GETNUMMEMZONES,
             &phys_num_zones );
```

### 3.6. Get the start address, end address, and size of memory zones

New call, to be used only for 64 bit addressing:

```
phys_zones = (struct memZoneStruct *)
              malloc( phys_num_zones*sizeof (struct memZoneStruct));
err = ioctl( physmem_fd, PHYSMEM_GETMEMZONES, phys_zones );
```

Structure `memZoneStruct` holds info about a physical memory zone. It has the following members (defined in `physmem.h`):

```

struct memZoneStruct {
    unsigned long start; // physical start address of the zone
    unsigned long end;   // physical end address of the zone + 1
    unsigned long size;  // size of the zone in bytes
};

```

### 3.7. Map to user space

Common call for both 32 and 64 bit:

```

addr_user_physmem = mmap( 0,
                          <size>,
                          PROT_READ|PROT_WRITE,
                          MAP_SHARED,
                          physmem_fd,
                          0 );

```

where <size> is `phys_size_physmem` or `phys_full_size_physmem`, according to the last `ioctl()` call (32 bit: same value, the size of the `physmem` block).

### 3.8. Unmap from user space

Common 32/64 bit:

```

err = munmap( addr_user_physmem, <size> );

```

### 3.9. Close the device

Common call for both 32 and 64 bit:

```

err = close( physmem_fd );

```

### 3.10. Physmem open routine of the 64 bit API library

The 64 bit `physmem` package contains a C API library for programs using `physmem` memory. It comprises functions for opening and closing `physmem` device, for displaying memory zones, for calculating physical address from user address and vice versa, and for checking if a memory area contains gaps or not. Here only the open routine is described which fulfils all the steps described in the previous sections 3.2 to 3.7: opens the `physmem` device, gets the size and zone information and maps the area. The library's full description can be found in [1]. The synopsis of the routine:

```

#include physmem_lib.h
int physmemOpen ( physmemHandler_t *device,
                  int             minor,
                  int             mapping)

```

The return value is 0 if the open and mapping were successful and -1 in any other cases.

The routine has the following parameters:

- *device*: points to device handler to be filled by the routine in case of successful open. The type `physmemHandler_t` is defined in `physmem_lib.h` in the following way:

```
typedef struct{
    char      *physmem_dev; // physmem device name
    int       fd;           // file descriptor
    int       minor;       // physmem device minor
    int       full_size;   // 1 if GETFULLSIZE
    int       num_zones;   // number of memory areas
    unsigned long phys_addr_physmem; // start of physmem
    volatile unsigned long *addr_user_physmem; // user start
    unsigned long phys_size_physmem; //
    unsigned long full_size_physmem; //
    unsigned long sum_gaps; // total size of memory gaps
    struct memZoneStruc *zones; // memory areas info
    offZone_t *phys_offs; // physical offsets of memory areas
    offZone_t *user_offs; // user offsets of memory areas
} physmemHandler_t;
```

The handler contains pointers to structures holding information about the zone physical addresses (as defined in section 3.6) and also to structures including the physical and user offsets (distances in bytes from the start of the physmem device). The user and physical offsets of the same memory location can differ if the mapping is *usable*. The type `offZone_t` is also defined in `physmem_lib.h`:

```
typedef struct{
    unsigned long start;
    unsigned long end;
} offZone_t;
```

- *minor*: defines the minor number of the physmem device. Its value could be 0 or 1.
- *mapping*: defines the mapping method of the area. Value 0 specifies *usable* mapping, while value 1 specifies *full* mapping.

## 4. References

1. ALICE DAQ and ECS User's Guide, Internal note ALICE-INT-2010-001 (EDMS:<https://edms.cern.ch/document/1056364/1>).