

实验案例六：内核子系统—物理虚拟内存(指导文档)

实验案例六：内核子系统—物理虚拟内存(指导文档)

- 一、实验简介
- 二、实验内容及要求
 - 任务一: 虚实映射测试

一、实验简介

物理内存和虚拟内存是现代计算机系统的重要概念，在 OpenHarmony 的学习过程中同样至关重要。物理内存是计算机中直接与 CPU 交互的硬件存储，用于存储正在运行的程序 and 数据的临时区域。虚拟内存是一种内存管理技术，它为每个进程创建独立的虚拟地址空间，该空间可大于实际物理内存。操作系统通过虚拟内存管理和分配物理内存资源，确保每个进程拥有足够的内存空间，即使物理内存不足。此机制通过页表映射和页面调度实现，可将不常用的数据或代码移至磁盘，从而释放物理内存供其他进程使用。

本节实验基于 LiteOS-A 内核，旨在学习 OpenHarmony 中物理内存与虚拟内存的概念、重要性及实现方法，并通过在内核中增添代码的实践，深化对该知识点的理解。

二、实验内容及要求

本次实验需要依次完成下列实验要求，并提交在QEMU中运行的最终结果截图。

任务一：虚实映射测试

请参考函数 `LOS_vmalloc` 的实现尝试在内核空间中申请一个物理页 `ptr` 与两个虚拟页 `region1` 和 `region2`，最后将上述虚拟地址 `region1` 和 `region2` 映射到同一个物理页 `ptr` 上。在完成虚实映射任务之后，尝试运行如下代码：

```
/*
任务一：
1. 申请物理页和两份虚拟页va1与va2
2. 将两份虚拟页映射到同一个物理页上
*/
VADDR_T va1 = region1->range.base, va2 = region2->range.base;
int *va11 = (int*)va1, *va12 = (int*)va2;
PRINTK("virtual address of va1: %p\n", va11);
PRINTK("value of va1: %d\n", *va11);
PRINTK("virtual address of va2: %p\n", va12);
PRINTK("value of va2: %d\n", *va12);
*va11 = 2025;
PRINTK("Assigning a value to va1: %d\n", *va12);
PRINTK("virtual address of va1: %p\n", va11);
PRINTK("value of va1: %d\n", *va11);
PRINTK("virtual address of va2: %p\n", va12);
PRINTK("value of va2: %d\n", *va12);
/*
任务一：
3. 释放申请的物理页和虚拟页
*/
```

假如实现正确的话，其打印信息应展示如下，虽然va1和va2指向着不同的虚拟地址，但是改变其中任意一个变量，都会影响到另一方。

```
virtual address of va1: 0xc0000000
value of va1: 0
virtual address of va2: 0xc0001000
value of va2: 0
Assigning a value to va1: 2025
virtual address of va1: 0xc0000000
value of va1: 2025
virtual address of va2: 0xc0001000
value of va2: 2025
```

流程

在LiteOS中虚拟地址空间由 `LosVmSpace` 控制，因此想要申请虚拟地址就无法绕开虚拟空间。而在LiteOS中每个用户态进程都有独立的进程空间，而内核态进程会共同使用同一个内核进程空间 `g_kVmSpace` 与内核动态分配空间 `g_vMallocSpace`。其中用户空间在内核运行时无法使用，而内核进程空间的虚拟地址已经在初始化时使用完，内核动态分配空间是 `vmalloc` 所使用的虚拟内存分配空间。因此本次实验选择内核动态分配空间 `g_vMallocSpace` 申请虚拟页。

内核动态空间通过 `LOS_GetVmallocSpace` 获得，而向进程空间申请虚拟地址则使用 `LOS_RegionAlloc` 完成，并利用 `LOS_RegionFree` 释放 故而任务的申请和释放两个虚拟页va1和va2和通过下述代码完成：

```
//申请虚拟页
LosVmSpace *vSpace = LOS_GetVmallocSpace(); // 获得地址空间
LosVmMapRegion *region1, *region2;
region1 = LOS_RegionAlloc(vSpace, 0, PAGE_SIZE, VM_MAP_REGION_FLAG_PERM_READ |
VM_MAP_REGION_FLAG_PERM_WRITE, 0);
region2 = LOS_RegionAlloc(vSpace, 0, PAGE_SIZE, VM_MAP_REGION_FLAG_PERM_READ |
VM_MAP_REGION_FLAG_PERM_WRITE, 0);
VADDR_T va1 = region1->range.base, va2 = region2->range.base;
...
// 释放虚拟页
LOS_RegionFree(vSpace, region1);
LOS_RegionFree(vSpace, region2);
```

物理内存通过伙伴算法管理，对于申请连续物理内存的请求，LiteOS提供了接口函数

`LOS_PhysPagesAllocContiguous` 实现，并提供函数 `LOS_PhysPagesFreeContiguous` 释放物理内存，故而对物理内存的申请和释放代码使用如下：

```
// 申请物理页
VOID *ptr = LOS_PhysPagesAllocContiguous(1);
...
// 释放物理页
LOS_PhysPagesFreeContiguous(ptr, 1);
```

LiteOS的虚实映射与解除映射关系则分别通过 `LOS_ArchMmuMap` 与 `LOS_ArchMmuUnmap` 实现，故而这部分代码实现如下：

```
// 虚实映射
LOS_ArchMmuMap(&vSpace->archMmu, va1, pa, 1, region1->regionFlags);
LOS_ArchMmuMap(&vSpace->archMmu, va2, pa, 1, region2->regionFlags);

...
// 解除映射
LOS_ArchMmuUnmap(&vSpace->archMmu, va1, 1);
LOS_ArchMmuUnmap(&vSpace->archMmu, va2, 1);
```

将上述代码组合起来，即得到

```
/*
任务一：
1. 申请一份物理页和两份虚拟页va1与va2
2. 将两份虚拟页映射到同一个物理页上
*/
LosVmSpace *vSpace = LOS_GetVmAllocSpace();
VOID *ptr = LOS_PhysPagesAllocContiguous(1);
LosVmMapRegion *region1, *region2;
region1 = LOS_RegionAlloc(vSpace, 0, PAGE_SIZE, VM_MAP_REGION_FLAG_PERM_READ |
VM_MAP_REGION_FLAG_PERM_WRITE, 0);
region2 = LOS_RegionAlloc(vSpace, 0, PAGE_SIZE, VM_MAP_REGION_FLAG_PERM_READ |
VM_MAP_REGION_FLAG_PERM_WRITE, 0);
VADDR_T va1 = region1->range.base, va2 = region2->range.base;
PADDR_T pa = OsKVaddrToPaddr((VADDR_T)ptr);
LOS_ArchMmuMap(&vSpace->archMmu, va1, pa, 1, region1->regionFlags);
LOS_ArchMmuMap(&vSpace->archMmu, va2, pa, 1, region2->regionFlags);
// *****
int *va1 = (int*)va1, *va2 = (int*)va2;
PRINTK("virtual address of va1: %p\n", va1);
PRINTK("value of va1: %d\n", *va1);
PRINTK("virtual address of va2: %p\n", va2);
PRINTK("value of va2: %d\n", *va2);
*va1 = 2025;
PRINTK("Assigning a value to va1: %d\n", *va2);
PRINTK("virtual address of va1: %p\n", va1);
PRINTK("value of va1: %d\n", *va1);
PRINTK("virtual address of va2: %p\n", va2);
PRINTK("value of va2: %d\n", *va2);
// *****
/*
任务一：
3. 释放申请的物理页和虚拟页
*/
LOS_ArchMmuUnmap(&vSpace->archMmu, va1, 1);
LOS_ArchMmuUnmap(&vSpace->archMmu, va2, 1);
LOS_RegionFree(vSpace, region1);
LOS_RegionFree(vSpace, region2);
LOS_PhysPagesFreeContiguous(ptr, 1);
```

最后重新编译与运行OHOS即可得到任务目标结果。

