Lab2 内存管理

代码仓库

本实验的代码在 `lab2` 分支中。做完 lab1 后，请在 `lab1` 分支上做一次 Commit，然后加入 `lab2` 分支的代码：

```
# After commit
git checkout lab2

git merge lab1
# If merge conflicts exist, You should handle them and then commit
```

实验内容简介

实验目标

在本实验中，需要实现教学操作系统内核的内存管理系统。

内存管理系统分为两部分：

内核的物理内存分配器，用于分配和回收物理内存，主体位于 `src/core/memory_manage.c` 。

内核的页表，用于将虚拟地址映射到物理地址，主体位于 `src/core/virtual_memory.c` 。

参考文档

因为内核需要与硬件大量交互，将参考 ARM 架构文档来理解相关硬件配置。我们会在实验文档中给出可以参考 ARM 架构文档的哪一部分。

以下是一些指代方式的例子：

`ref: C5.2` 表示相关部分在 ARMv8 Reference Manual 的 C5.2
`A53: 4.3.30` 表示相关部分在 ARM Cortex-A53 Manual 的 4.3.30
`guide: 10.1` 表示相关部分在 ARMv8-A Programmer Guide 的 10.1

物理内存管理

物理内存是指 DRAM 中的储存单元，每个字节的物理内存都有一个地址，称为物理地址。

虚拟内存是程序（用户进程、内核）看到的存储空间。

通常，硬件会内置一个（或许可编程的）转换单元，在 ICS-2021Spring-FDU 的 MIPS CPU 设计实验中，我们也实现了类似的机制。

在本实验中，你需要为内核实现一个物理内存分配器为后续用户进程的页表、栈等分配空间。

为了方便同学们自主发挥和扩展，我们抽象出了这样几个功能函数，并统一使用内存管理表 PMemory 进行管理。

```
typedef struct {
    SpinLock lock;
    void *struct_ptr;
    void (*page_init)(void *datastructure_ptr, void *start, void *end);
    void *(*page_alloc)(void *datastructure_ptr);
    void (*page_free)(void *datastructure_ptr, void *page_address);
} PMemory;
```

其中，每一个函数指针应指向你实现的相应函数。

接下来，我们将描述每一个相应函数的参数和理应实现的功能。

```
NORETURN void (*page_init)(void *datastructure_ptr, void *start, void *end);
/*  parameters:
 *      void *data_structure_ptr:
 *          This should point at the data structure for page management.
 *          For example: head of a linked-list or root of a tree.
 *      void *start:
 *          This should be the first page's address.
 *      void *end:
 *          This should be the last page's address.
 *  function:
 *      As initialization, put all free pages into your own data structure.
 */
void *(*page_alloc)(void *datastructure_ptr);
/*  parameters:
 *      void *data_structure_ptr:
 *          This should point at the data structure for page management.
 *          For example: head of a linked-list or root of a tree.
 *  function:
 *      Take one page out of your data structure and return its address.
 */
NORETURN void (*page_free)(void *datastructure_ptr, void *page_address);
/*  parameters:
 *      void *data_structure_ptr:
 *          This should point at the data structure for page management.
 *          For example: head of a linked-list or root of a tree.
 *      void *page_address:
 *          The address of the page that needs to be freed.
 *  function:
 *      Save the page into your data structure.
 */
```

敏锐的同学会发现，这个表里本质只有一些函数指针，指针指向的函数还需自己定义。

在具体的实现部分，我们推荐大家维护一个链表来存储空闲页，链表中每个节点的结构如下：

```
typedef struct {
    void *next;
} FreeListNode;
```

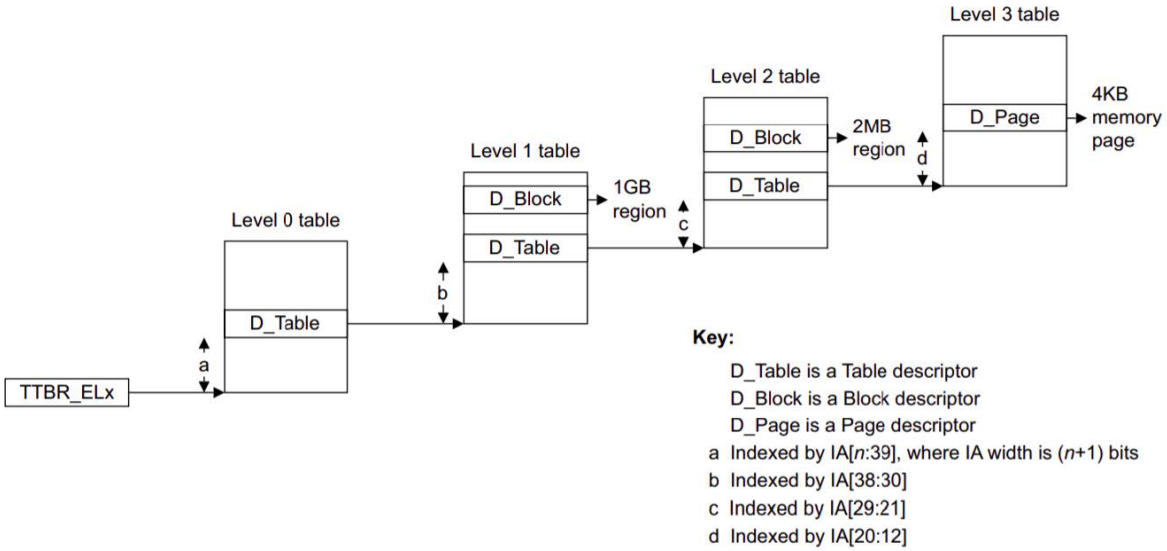
思考题：页大小为 4KB，但为什么这样就足够了?

在本次实验中，助教已经把指针指向了对应的函数，因此同学们只需要填充以下几个函数的内容即可。

```
NORETURN static void freelist_init(void *freelist_ptr, void *start, void *end);
static void *freelist_alloc(void *freelist_ptr);
NORETURN static void freelist_free(void *freelist_ptr, void *page_address);
```

页表管理

虚拟内存与页表的管理（页的粒度、虚拟地址的大小等）与指令集架构密切相关，实验使用的 ARM 架构提供了三种页的粒度：64 KB(16 bits)，16 KB(14 bits)，4 KB(12 bits)，本学期的内核实验我们将同一且仅采用 4 KB 的页大小，在 48-bit 索引的虚拟内存中虚拟地址通过页表转换为物理地址的过程如下图。



当 CPU 拿到 64-bit 的虚拟地址时：

如果前 16 位全 0，CPU 从 ttbr0_el1 读出 level-0 页表。

如果前 16 位全 1，CPU 从 ttbr1_el1 开始 level-0 页表。

否则，报错。

拿到 level-0 页表后，CPU 会依次以 va[47:39] , va[38:39] , va[29:21] , va[20:12] 为索引来获取下一级页表信息（页表地址、访问权限）。

```
// Simplified code of ARMv8 MMU
module MMU #(
    parameter paddr_t = uint64_t, // physical address
    parameter vaddr_t = uint64_t  // virtual address
)(
    input req_valid,
    input vaddr_t va,               // Software memory requests must use va
    output paddr_t pa,              // Give pa
    output paddr_t dreq_addr,       // Hardware uses pa to get next-level
    input uint64_t dresp_data,      // The pa of next-leval pgtable is
{dresp_data[63:12], 12'b0}
    input paddr_t ttbr0, ttbr1      // Translation table base registers store pa
);
    localparam state_t = enum {
        INIT,
        LOAD_L1,
        LOAD_L2,
```

```
        LOAD_L3
    };
    state_t state;

    always_comb begin
        unique case (state) begin
            INIT: begin
                if (req_valid) begin
                    if (cache_hit(va)) begin // this cache would be TLB
                        pa = cache_read(va);
                    end else begin
                        state = LOAD_L1;
                        ttbr = select(va, ttbr0, ttbr1); // based on va[63:48]
                        dreq_addr = ttbr | L1_INDEX(va);
                    end
                end
            end
            LOAD_L1: begin
                if (IS_BLOCK(dresp_data)) begin
                    state = INIT;
                    pa = L1_ALIGN(dresp_data) | L1_OFFSET(va);
                end else begin
                    state = LOAD_L2;
                    dreq_addr = (PAGE_ALIGN(dresp_data)) | L2_INDEX(va);
                end
            end
            // LOAD_L2 and LOAD_L3 are similar to LOAD_L1
            LOAD_L2: begin
                if (IS_BLOCK(dresp_data)) begin
                    state = INIT;
                    pa = L2_ALIGN(dresp_data) | L2_OFFSET(va);
                end else begin
                    state = LOAD_L3;
                    dreq_addr = (PAGE_ALIGN(dresp_data)) | L3_INDEX(va);
                end
            end
            LOAD_L3: begin
                if (IS_PAGE(dresp_data)) begin
                    state = INIT;
                    pa = L3_ALIGN(dresp_data) | L3_OFFSET(va);
                end else begin

                end
            end
        end
    end

endmodule
```

每级页表中，页表项 entry 后 12 位用于权限控制，前若干位用来指示页表项所指页表的物理地址。在 level-1、level-2 中 entry[1] 可用于指示当前项的属性（为 0 时 block、为 1 时 table）(ref:D4.3.1)，在 level-3 中 entry[1] 只能为 1，指示当前项中包含物理地址 (ref:D4.3.2)，entry[0] 用于指示当前项是否有为空（可用于映射物理地址）(ref:D4.3.2)，entry[7:6] 用于权限管理 (ref:D4.4.4)。

类似地，我们使用 VirtualMemoryTable 来管理所用到的函数（从而完成这一级抽象）。

```
typedef struct {
    PTEntriesPtr (*pgdir_init)(void);
    PTEntriesPtr (*pgdir_walk)(PTEntriesPtr pgdir, void *kernel_address, int
alloc);
    PTEntriesPtr (*uvm_copy)(PTEntriesPtr pgdir);
    NORETURN void (*vm_free) (PTEntriesPtr pgdir);
    int (*uvm_map)(PTEntriesPtr pgdir, void *kernel_address, size_t size,
uint64_t physical_address);
    int (*uvm_alloc) (PTEntriesPtr pgdir, size_t base, size_t stksz, size_t
oldsz, size_t newsz);
    int (*uvm_dealloc) (PTEntriesPtr pgdir, size_t base, size_t oldsz, size_t
newsz);
    int (*copyout)(PTEntriesPtr pgdir, void *tgt_address, void *src_address,
size_t len);
} VMemory;
```

通常，该级抽象不需要独有的数据结构（因为依赖于页表，所以我们没有为这一层提供数据结构的指针）。

假如你的代码需要这样的指针，请自己添加，并记得修改对应接口，并在报告中描述这样做的理由和好处（方便之后助教提供调试方面的帮助）。

```
PTEntriesPtr (*pgdir_init)(void);
/*  parameters:
 *  function:
 *      Generate a empty page to work as page directory.
 */
PTEntriesPtr (*pgdir_walk)(PTEntriesPtr pgdir, void *kernel_address, int alloc);
/*  parameters:
 *      PTEntriesPtr pgdir:
 *          The level-0 page directory that you want to walk through.
 *      void *kernel_address:
 *          The virtual address of the page that you want to get.
 *      int alloc:
 *          A 0/1 variable, stands for whether to alloc a new page if the page
 *          with that virtual_address doesn't exit.
 *  function:
 *      Get the page on that virtual_address within the page directory.
 */
NORETURN void (*vm_free) (PTEntriesPtr pgdir);
/*  parameters:
 *      PTEntriesPtr pgdir:
 *          The level-0 page directory that you want to walk through.
 *  function:
 *      Free the entire page directory.
 */
int (*uvm_map)(PTEntriesPtr pgdir, void *virtual_address,
               size_t size, uint64_t physical_address);
/*  parameters:
 *      PTEntriesPtr pgdir:
 *          The level-0 page directory that you want to walk through.
 *      void *kernel_address:
 *          The beginning of page's vitual address to map.
 *      size_t size:
 *          The memory size (size bytes) to map.
 *      uint64_t physical_address:
 *          The beginning of page's phisic address to map.
```

```
 *  function:
 *      Within page directory, map aligned phisic space to virtual space.
 *      So that if you get the data of virtual address from the page directory,
 *      you could get the data of coresponding phisic address.
 */
```

类似地，助教也已经把指针指向了对应的函数，因此同学们只需要填充以下几个函数的内容即可。

```
static PTEntriesPtr my_pgdir_init();
static PTEntriesPtr my_pgdir_walk(PTEntriesPtr pgdir, void *vak, int alloc);
NORETURN static void my_vm_free(PTEntriesPtr pgdir);
static int my_uvm_map(PTEntriesPtr pgdir, void *va, size_t sz, uint64_t pa);
```

Exercise

截止时间：`2021-10-8 15:24:59`。

提交方式：将实验报告提交到`elearning`上，文件名：`学号-lab2.pdf`。

物理内存管理

完成物理内存管理所对应的以下几个函数（或者自己改写相关代码使得能够实现对应功能）。

```
NORETURN static void freelist_init(void *freelist_ptr, void *start, void *end);
static void *freelist_alloc(void *freelist_ptr);
NORETURN static void freelist_free(void *freelist_ptr, void *page_address);
```

页表管理

完成页表管理所对应的以下几个函数（或者自己改写相关代码使得能够实现对应功能）。

```
static PTEntriesPtr my_pgdir_init();
static PTEntriesPtr my_pgdir_walk(PTEntriesPtr pgdir, void *vak, int alloc);
NORETURN static void my_vm_free(PTEntriesPtr pgdir);
static int my_uvm_map(PTEntriesPtr pgdir, void *va, size_t sz, uint64_t pa);
```

测试

`src/core/virtual_memory.c`中有一个简单的测试。大家可以自行添加测试。

后续，我们会用更完备的测试来检查大家是否正确地实现了上述功能。测试思路为：

```
1. Allocate thousands of pages `v[N]`, `N is about 1e6`.
2. Map those pages (`v[i]` -> `p[i]`).
3. For each page, call pgdir_walk(). Assert(`p[i]` == pgdir_walk(`v[i]`)).
4. Free some pages.
5. For each page, call pgdir_walk(). For pages freed, the return value should be
0. Otherwise, the same as 3.
6. Free the remaining pages.
7. For each page, call pgdir_walk(). Assert (0 == pgdirwalk(`v[i]`))
```

可能会用到的宏或者函数

```
/* All of these are in <aarch64/mmu.h> or <common/type.h>.
 * Only to mention that K stands for kernel(space), P for phisic(space).
 */
PAGE_SIZE
PTE_VALID
K2P(x)
P2K(x)
PTE_ADDRESS(x)
PTE_FLAGS(x)
ROUNDDOWN(x, size)
ROUNDUP(x, size)
```