任务一：进程内存初始化与页表建立

中国科学院大学 操作系统研讨课

2017.11.29

1. 介绍

本任务的主要目的是实现进程内存初始化。

1.1. 需要了解的部分

- 虚拟内存映射结构

2. 初始代码

2.1. 文件介绍

- Makefile: 编译文件。
- bootblock.s: 内核启动程序，**请使用作业一中自己写的代码**。
- Createimage.c: 生成内核镜像的Linux工具，**请使用作业一中自己写的代码**。
- Entry.S: 中断处理函数。
- kernel.c: 内核最先执行的文件，放在内核的起始处。
- scheduler.c: 调度器，实现*task*的调度。
- syslib.S: 系统调用函数
- syslib.c: 系统调用接口
- interrupt.c: 系统调用和中断处理相关的函数。
- queue.c: 队列处理函数，提供了队列操作的一些接口。
- print*.c：提供一些输出函数，可以用于调试以及显示信息。
- sync.c: 一些同步操作。
- mbox.c: 邮箱的操作。
- util.c: 提供了一些输出函数，可以用于调试以及显示信息。
- file.c: 将所有任务放在File的机构体中，供do_spawn查找使用
- ramdisk.c：提供操作File的一些接口
- *.h: 相应.c/.S文件的头文件。
- Memory.c: 需要实现的关于虚拟内存的函数。
- Memory.h: 需要实现的关于虚拟内存的一些定义。

2.2. 获取

课程网站。

2.3. 运行

Makefile 文件提供编译功能。
make 编译命令
make clean 对编译产生的文件进行清除
sudo dd if=image of=/dev/sdb 将产生的 image 写进 SD 卡中
在 minicom 中执行 *loadboot* 运行程序

3. 任务

3.1. 设计和评审

帮助学生发现设计的错误，及时完成任务。学生需要对这次的作业进行全面考虑，在实现代码之前有清晰的思路。学生讲解设计思路时可以用不同的形式，如伪代码、流程图等，建议使用 PPT。

3.1.1.设计介绍

- MIPS虚拟内存规划
- TLB重填操作（tlb_refill）和页表切换

3.2. 开发

3.2.1.要求

本次任务主要是将用户进程加载到需要进行虚存转换的内存区域，并实现虚拟内存的初始化。主要实现 memory.h 中的 page_map 结构定义和 memory.c 中的各个函数：

- init_memory：该函数实现对page_map的初始化
- page_paddr：该函数需要实现根据传入的物理页号返回对应的物理页框地址
- page_vaddr：该函数需要实现根据传入的物理页号返回对应的虚拟地址
- page_alloc：该函数实现从物理页框中找到一个空闲页框，并初始化页框属性

- setup_page_table：该函数需要实现建立进程的页表。

此外，本次任务还需要在 entry.S 实现 TLB 的例外处理，即实现 handle_tlb 完成 TLB 重填操作，以及实现页表切换。

3.2.2.注意事项（请仔细阅读注意事项）

- 本次任务不考虑内核线程的虚拟内存。
- 进程栈的设置

本任务中，进程栈仍然使用内核空间，不需要为进程栈建立页表。

- 物理页框

本任务中的物理页框是对从 MEM_START 开始的 PAGEABLE_PAGES 个物理页框依次编号(即物理页号)，用 memory.c 中的 page_map[PAGEABLE_PAGES]表示，每个页表包含 PAGE_N_ENTRIES 个页表项

上述常量均定义在 memory.h 中，其他相关常量请仔细阅读 memory.h

- Page map entry

memory.h 中需要定义一个描述物理页框的结构，相当于每个页框的 metadata，用于追踪所有页框的信息，请设计好该结构需要维护页框的哪些信息。后续任务也会用到该结构，可以根据实际需要逐步添加。

- page_paddr和page_vaddr

上述两个地址转换的函数并不是给进程用的，请注意 memory.c 是内核的代码，所以函数操作的地址都是内核的虚地址，page_paddr 函数的用途是可以通过物理页号得到相应物理地址。page_vaddr 则是通过物理页号得到该页在内核地址空间中的虚拟地址。此处需要考虑物理页号和物理地址的对应关系，以及内核虚址和物理页号如何换算。

上述函数应用场景为：内核为进程分配一个物理页面，后续往该物理页拷贝内容时，需要把该物理页映射到内核的地址空间里才能进行操作，此时需要用到上述函数。

- page_alloc

本任务中假设物理页面足够，不考虑页替换。

- setup_page_table:

建立进程的页表具体流程：如下

首先，申请空闲页面(page_alloc)作为页表。

其次，为进程分配若干页面(根据进程的大小)，并填充页表项，注意：此处页表项标志位需要设置成有效，即这里的页面需要马上分配。另外，填充一个页表项之后，为了保证和 TLB 的一致性，需要将刚刚填充到页表的页表项在 TLB 中无效掉，我们提供了一个函数：tlb_flush 来进行该操作。TLB 操作的介绍请参看任务二中的介绍，以及我们提供的参考文档，也可以参考 entry.S 中的 tlb_flush 函数。

- handle_tlb操作

我们将页表项填充到页表后，处理器并看不到，在访问某个地址时，处理器会先查询 TLB，如果没有该转换项，会产生 TLB 例外，此处需要在 entry.S 中的 handle_tlb 处理该例外。你需要实现该函数，这里只需要考虑 TLB 中没有转换项的情况（即 tlb_refill 流程）。tlb_refill 流程如下

- 当处理器访问地址发生错误时，会自动将当前的出错地址保存到CP0_BADVADDR寄存器中(参考cp0regdefs.h文件)
- 首先需要获得当前的页表(参考“页表切换”注意事项)
- 其次，根据出错的地址在页表中查到该页表项（在本任务中，该项应该已在setup_page_table中进行填充）
- 最后，将页表项填到TLB中。TLB操作的介绍请参看任务二中的介绍，以及我们提供的参考文档。

- 页表切换

每个进程都有自己的一个页表，但是 MIPS 不像 x86 有专门的寄存器维护当前页表，因此你需要考虑如何保存记录当前使用的页表，以及如何切换当前使用的页表。entry.S 中提供了一个名为 set_pt 的 LEAF 过程，可以在这实现页表切换，或者设计你自己的方法进行页表切换。

- 代码段载入(注意此段描述中的地址如果不特别指出，均为虚拟地址)

本任务中没有文件系统，因此我们将代码段预先放置在内存中，例如之前作业中开发的 createimage 会先在 ELF 文件中查找代码，然后写入文件。假设内核的起始地址是 0x80800200，大小是 0x1000，进程 1 的起始地址是 0x80802000，大小是 0x1000，进程 2 的起始地址是 0x80804000，大小是 0x1000，则 createimage 会依次存放内核、进程 1 代码、进程 2 代码，其中不连续的区域会有 padding。

在本次任务中，进程的地址会指定在 0x00000000 -- 0x7FFFFFFF 区域，例如进程起始地址指定为 0x1000，此时进程的虚址与实际 image 会不一致。你需要考虑如何处理该情况，即如何保证进程代码能正确加载到对应虚址上。

此处提供一个参考方法：createimage 中把内核写入 image 之后，继续写进程 1 的内容时记录下进程 1 当前在 image 中的偏移。后续启动进程时，将该偏移处的进程 1 代码拷贝到指定的进程起始地址（虚址）（注意此处的虚址需要已有分配的物理页面和其对应），然后跳转到起始地址开始执行。例如，整个 image 载入内存时，假设内核结尾地址为 0x80801200，进程 1 代码段处于 0x80801200 开始的地方，但是我们指定的进程起始地址是 0x1000，此时需要把 0x80801200 处的进程 1 代码拷贝到 0x1000 地址处，这样进程 1 才能够正常执行

上述方法在 start code 中对应三个关键变量：参考 task.c，一个 entry_point 表示进程的虚拟空间的起始虚地址，loc 表示进程代码存放的虚拟地址。启动进程时要把 loc 处的代码拷贝到 entry_point 才能正常运行。size 表示进程代码段的大小。entry_point 是在 Makefile 中通过 PROC_ADDR 指定，loc 通过 Makefile 中 PROC_LOC 指定，请参考我们提供的 Makefile。另外，请注意你的 createimage，需要能够得到进程的 loc 和 size，并修改对应的 Makefile。

4. 测试

只有将内存初始化成功，提供的 process1.c 和 process2.c 中的进程才能够正常运行。我们这里提供了两个进程，一个是飞机进程，一个是计算前 n 项和的进程。

本任务测试使用 make 直接编译后，loadboot 执行即可。