README

设计思路

目前完成到了 A-core

App-info

设计的 App-info, 位于bootblock、main之后，完整的布局如下所示

```
0x0000 0000 ____________________________________________________
                bootblock(0)
0x0000 0200 ____________________________________________________
                main(1 - 8)    filesz: 4072      memorysz: 4512
0x0000 1200 ____________________________________________________
                App info(9 -10)
0x0000 1600 ____________________________________________________
                 bss(11)  (1)   filesz:  352     memorysz: 408
0x0000 1800 ____________________________________________________
                auipc(12-13)  (2)  filesz: 826   memorysz: 826
0x0000 1c00 ____________________________________________________
                data(14-22)  (9)   filesz: 4504      memorysz: 4504
0x0000 2E00 ____________________________________________________
                2048(23-34)  (12)   filesz: 5896      memorysz: 5896
0x0000 4600 ————————————————————————————————————————————————————
```

main随着之后的设计还会增长，所以 main 部分的扇区数大小不确定。

task_info_t

修改的函数，添加了 load_task_img_by_name的部分，并且添加了根据当前 task_name 返回 filesz 和 memorysz

```c
typedef struct {
    char taskname[EI_NIDENT];
    int start_block_id;
    int total_block_num;
    long task_filesz;
    long task_memorysz;
}task_info_t;

uint64_t load_task_img(int taskid);
```

```
long load_taskfilesz(int tasknum, char *taskname);
long load_taskmemorysz(int tasknum, char *taskname);
uint64_t load_task_img_by_name(int tasknum, char *taskname);
```

在main函数里，为了和 用户程序，以及 crt0.S 中设计的用户程序栈的位置作区别，

将 app_info 固定放在内存的 0x54000000 处。

其中对于 bss 段的更新，需要从 ELF文件的 phdr 读取 p_filesz 和 p_memsz, 从 p_filesz 到 p_memsz 的一段空间全部清零，用作bss段的处理。

参数的传递方法通过， load.h 中定义的 load_taskfilesz 和 load_taskmemorysz

```
        long task_enterance_address = load_task_img_by_name((int)total_task_num,
        long current_task_filesz = load_taskfilesz((int)total_task_num, buf_task
        long current_task_memorysz = load_taskmemorysz((int)total_task_num, buf_
        asm volatile( "mv a6, %0\n"
            : : "r"(current_task_filesz));
        asm volatile("mv a7, %0\n"
            : : "r"(current_task_memorysz));
        ( *(void(*)(void))task_enterance_address)();
```

通过在 main.c 中进行内联汇编的方法，将 task_filesz 放入 a6 寄存器，task_memorysz 放入 a7 寄存器。由于调用了 bios_sd_read, 因此用户程序所在的内存地址可以从 a1寄存器(默认函数返回值)中读到。

crt0.S

```
#include <asm.h>

# define USER_STACK_BASE 0x53500000
# define STACK_SIZE 0x10000
# define KERNEL_ENTER_POINT 0x50201000

.section ".entry_function","ax"
ENTRY(_start)
    /* TODO: [p1-task3] setup C runtime environment for the user program */

    lui t3, %hi(USER_STACK_BASE)
    addi t3, t3, %lo(USER_STACK_BASE)   // t3 = USER_STACK_BASE

    mv sp, t3                           // Every program has it's own stack_poir

    // clean bss segment
```

```
    mv t0,  a1                          // enterance address
    add t1, t0, a6                     // a6 has task_filesz, t1 = enterance_ado
    add t2, t0, a7                     // a7 has task_memorysz, t2 = enterance_a
    addi t2, t2, 1
    bge  t1, t2, bss_clean_done         // t1 >= t2 + 1

bss_clean_loop:
    sw zero, 0(t1)                      // clean
    addi t1, t1, 4                      // store word in 4
    blt t1, t2, bss_clean_loop

bss_clean_done:

    /* TODO: [p1-task3] enter main function */
    j main
    /* TODO: [p1-task3] finish task and return to the kernel, replace this in p3
    lui t5, %hi(KERNEL_ENTER_POINT)
    addi t5, t5, %lo(KERNEL_ENTER_POINT)
    jr t5
    /*************************************************************/
        /* Do not touch this comment. Reserved for future projects. */
        /*************************************************************/
// while(1) loop, unreachable here
loop:
    wfi
    j loop

END(_start)
```

使用 t1 表示从 a1(例如 bss 用户程序被放在 0x5200 0000) 加上偏移 filesz 的地址位置

使用 t2 表示从 a1，加上偏移 memsz 的地址位置

采用 sw zero, 0(t1) ，进行4字节为单位的清零操作，清空bss段。

设置用户栈的地址为 USER_STACK_BASE 0x53500000

这里固定了该值，后面对于不同的用户程序，需要将栈空间的设置分开。

编译命令

由于目前没有完成 C-core, 所以不需要对 Makefile 进行修改

```
make all
make run
```

之后输入 loadboot即可

之后需要用户输入两段信息，第一段测试打印字符功能，输入 0 表示退出

第二段测试能否根据用户程序名进行索引，输入 * 表示结束。