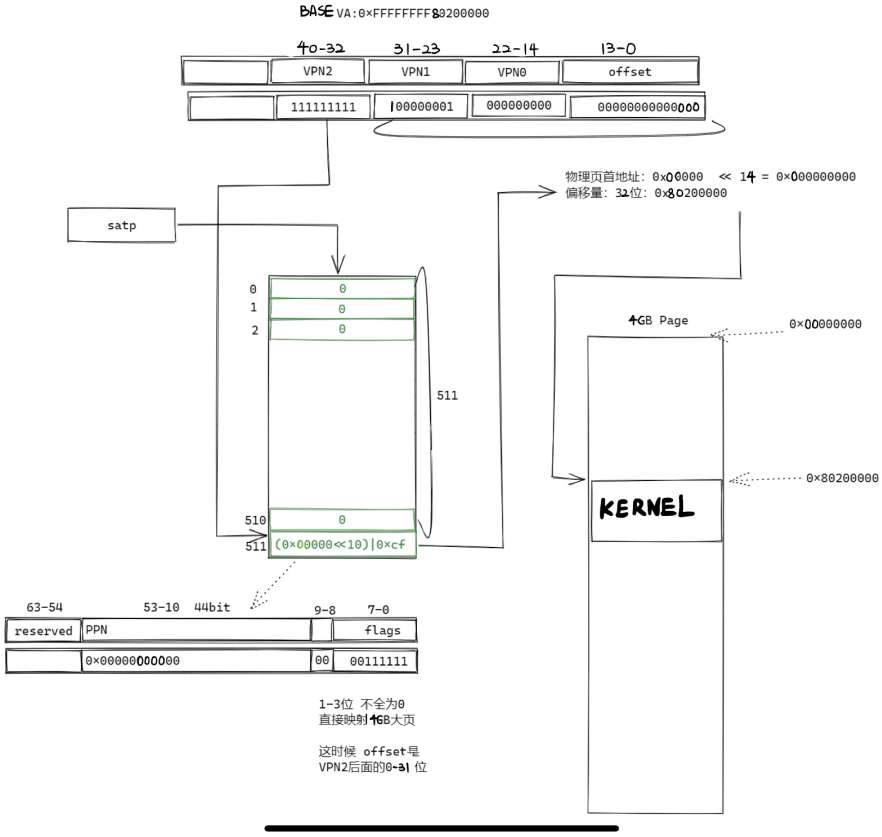
CS334 Project Report

成员及负责工作

成员	负责工作
罗嘉俊 (12012023)	基于Sv39 设计了虚拟地址和物理地址的划分格式以及虚拟地址到物理地址的转换模式，实现了16KOS 以及 CFS， 报告
廖铭骞 (12012919)	物理内存分配算法(first fit, worst fit 以及 best fit, next fit)以及进程管理(RR)，报告

物理地址和虚拟地址格式与转换

下图展示了在本次project 当中，我们的物理地址和虚拟地址的格式以及转换方式



16K OS 整体架构说明

Qemu 修改部分

在本次 project 当中，对于 Qemu 的修改部分如代码截图所示

```
/* Leaf page shift amount */
#define PGSHIFT             14
```

```
#endif
#define RISCV_PGSHIFT           14
#define RISCV_PGSIZE            (1 << RISCV_PGSHIFT)
```

```
# define TARGET_VIRT_ADDR_SPACE_BITS 32 /* sv32 */
#endif
#define TARGET_PAGE_BITS 14 /* 16 KiB Pages */
#define NB_MMU_MODES 4
```

物理内存分配及测试

在物理内存分配算法上面，我们实现了基于 16KOS 的三种物理内存分配算法。分别是 `best fit`、`worst fit`以及 `next fit`

对于 `best fit`，代码思路是在需要分配 `n` 页的时候，遍历一遍所有的空闲块，找到空闲块大小最接近 `n`的空闲块，之后从这个空闲块当中分配出 `n` 个页面。

主要代码如下

```
static struct Page *
best_fit_alloc_pages(size_t n) {
    assert(n > 0);
    if (n > nr_free) {
        return NULL;
    }
    struct Page *page = NULL;
    list_entry_t *le = &free_list;
    int best=1<<20;//设置初始值为较大值
    while ((le = list_next(le)) != &free_list) { // 循环直到找到一个符合大小条件的页面
        struct Page *p = le2page(le, page_link);
        if (p->property >= n) {
            if(p->property<best){//best fit
                best=p->property;
                page = p; // 记录符合条件的页面
            }
        }
    }
    if (page != NULL) {
        list_entry_t* prev = list_prev(&(page->page_link));
        list_del(&(page->page_link));
        if (page->property > n) { // 重新设置相关成员
            struct Page *p = page + n;
            p->property = page->property - n;
            SetPageProperty(p);
            list_add(prev, &(p->page_link));
        }
        nr_free -= n;
        ClearPageProperty(page);
    }
    return page;
}
```

运行测试的结果如下：

```
grading: 1 / 6 points
grading: 2 / 6 points
grading: 3 / 6 points
grading: 4 / 6 points
grading: 5 / 6 points
grading: 6 / 6 points
check_alloc_page() succeeded!
```

对于 `worst fit`，代码思路是在需要分配 `n` 页的时候，遍历一遍所有的空闲块，找到空闲块大小最不接近 `n` 的空闲块，之后从这个空闲块当中分配出 `n` 个页面。

主要部分的代码如下

```
static struct Page *
worst_fit_alloc_pages(size_t n)
{
    assert(n > 0);
    if (n > nr_free)
    {
        return NULL;
    }
    struct Page *page = NULL;
    list_entry_t *le = &free_list;
    int worst = -1;
    while ((le = list_next(le)) != &free_list)
    {
        struct Page *p = le2page(le, page_link);
        if (p->property >= n)
        {
            if (p->property > worst)
            { // worst fit
                worst = p->property;
                page = p;
            }
            // if (p->property >= n) {
            //     page = p;
            //     break;
            // }
        }
    }
    if (page != NULL)
    {
        list_entry_t *prev = list_prev(&(page->page_link));
        list_del(&(page->page_link));
        if (page->property > n)
        {
            struct Page *p = page + n;
            p->property = page->property - n;
            SetPageProperty(p);
            list_add(prev, &(p->page_link));
        }
        nr_free -= n;
```

```
        ClearPageProperty(page);
    }
    return page;
}
```

运行之后，可以通过所有的测试，效果截图如下：

```
os is loading ...
memory management: worst_fit_pmm_manager
physcial memory map:
  memory: 0x0000000007e00000, [0x0000000080200000, 0x000000087fffffff].
grading: 1 / 6 points
grading: 2 / 6 points
grading: 3 / 6 points
grading: 4 / 6 points
grading: 5 / 6 points
grading: 6 / 6 points
check alloc page() succeeded!
```

对于实现的 `next_fit` 算法，基本思想是记录每一次分配的页面的空闲块所在的位置，下一次从这里开始遍历，而不是每一次都从 `free_list` 的队首开始遍历。

代码主要部分实现如下：

```
static struct Page *
next_alloc_pages(size_t n) {
    assert(n > 0);
    if (n > nr_free) {
        return NULL;
    }
    struct Page *page = NULL;
    list_entry_t *le = NULL;
    list_entry_t *tmp = NULL;
    if(last_list == NULL){ // 初始的时候，从 free_list 队首开始遍历
        le = &free_list;
        tmp = &free_list;
    }else{
        le = last_list; // 如果 last_list 不为空，则从上一次遍历到的空闲块位置开始执行
        tmp = le;
    }

    while ((le = list_next(le)) != tmp) {
        struct Page *p = le2page(le, page_link);
        if (p->property >= n) {
            page = p;
            last_list = le;
            break;
        }
    }
    if (page != NULL) {
        list_entry_t* prev = list_prev(&(page->page_link));
        list_del(&(page->page_link));
```

```
        if (page->property > n) {
            struct Page *p = page + n;
            p->property = page->property - n;
            SetPageProperty(p);
            list_add(prev, &(p->page_link));
        }
        nr_free -= n;
        ClearPageProperty(page);
    }
    return page;
}
```

由于这个方法只是改变了开始遍历的起始位置，所以测试和 `default_pmm` 的测试是一样的，也可以完全通过测试

```
os is loading ...
memory management: next_pmm_manager
physcial memory map:
  memory: 0x0000000007e00000, [0x0000000080200000, 0x000000087fffffff].
grading: 1 / 3 points
grading: 2 / 3 points
grading: 3 / 3 points
check_alloc_page() succeeded!
```

进程调度管理

在本次项目当中，我们实现了两种进程调度管理的方式，分别是轮转调度（Round Robin, RR）以及完全公平性调度（Completely Fair Scheduler, CFS）

轮转调度（RR）

轮转调度的主要思想是按照一定的时间片，均匀地调度每个进程，使得每个进程轮流地运行一段时间片，以此来实现较为公平的调度。

主要的代码部分如下所示：

```
int main(void)
{

     int i, time;
     memset(pids, 0, sizeof(pids));
     for (i = 0; i < TOTAL; i++)
     {
          acc[i] = 0;
          if ((pids[i] = fork()) == 0)
          {
               acc[i] = 0;
               while (1)
               {
                    spin_delay();
                    ++acc[i];
                    if (acc[i] % 4000 == 0)
```

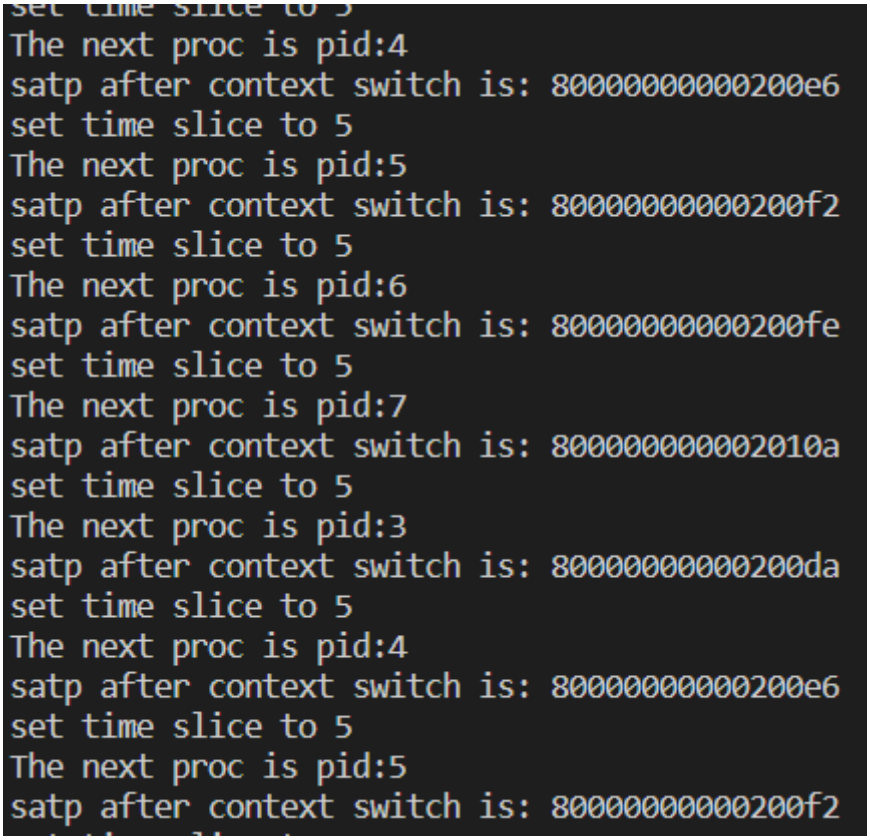
```
                    {
                         if ((time = gettime_msec()) > MAX_TIME)
                         {
                              cprintf("child pid %d, acc %d, time
%d\n", getpid(), acc[i], time);
                              exit(acc[i]);
                         }
                    }
               }
          }
          if (pids[i] < 0)
          {
               goto failed;
          }
     }

     cprintf("main: fork ok,now need to wait pids.\n");

     for (i = 0; i < TOTAL; i++)
     {
          status[i] = 0;
          waitpid(pids[i], &status[i]);
          // cprintf("main: pid %d, acc %d, time
%d\n",pids[i],status[i],gettime_msec());
     }
     cprintf("main: wait pids over\n");
     return 0;

failed:
     for (i = 0; i < TOTAL; i++)
     {
          if (pids[i] > 0)
          {
               kill(pids[i]);
          }
     }
     panic("FAIL: T.T\n");
}
```

在运行之后，也可以正确地通过所有的测试，测试结果如下所示



在初始分配的时间片是 5 时，进程调度可以按照 RR 的顺序轮转执行。

完全公平性调度（CFS）