LAB4 实验报告

一、小组成员及贡献排序：

汪值 林元灿 吕若晢 陈景昊 木扎拜·塔依尔

二、实验内容

1．阅读源码，了解 Linux2.6 分页机制实现和以及虚拟内存管理的方法

2．添加系统调用 memsys，其从调用者接收一个代表进程 ID 的字符串参数，打印分配给该进程用户空间的所有物理页面

3．编译并测试系统调用

三、实验过程

1. Linux2.6 分页机制和内存管理的实现

Linux 中与内存管理相关的代码在/mm 目录下，主要分为 4 个模块，内存映射(mmap),负责逻辑地址到物理地址的映射；交换模块(swap),负责页面的换入换出；核心模块(core)负责对页的初始化、分配、释放等功能；剩下特定结构的模块用于处理页错误和为硬件结构提供接口。各部分关系如下图所示：

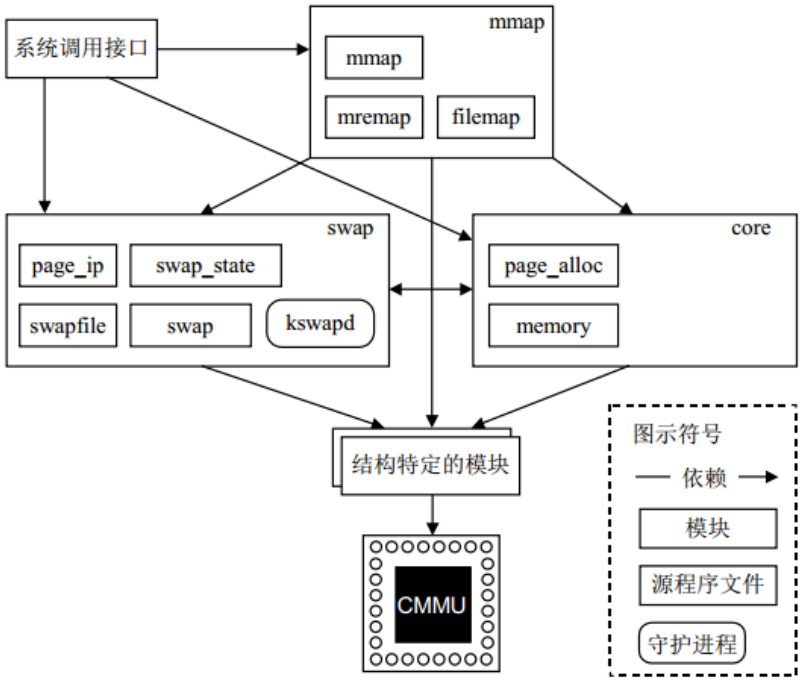

图 6.2　Linux 虚拟内存的实现结构

Linux2.6 中页面大小仍然为 4K，与之前版本不同之处在于 2.6 版本采用多级分页模型。32 位系统中一般采用二级分页，在 linux2.6.11 之后的版本中，采用四级分页模型，可以支持 64 位系统。

在二级分页模型中，存在页表和页目录。页全局目录中包含若干个页表的地址，页表中每个页表项指向一个页面。逻辑地址结构：10 位页目录+10 位页表+12 位页内偏移量，转换为物理地址：20 位物理地址+12 位偏移量。

2. 添加系统调用

（1）.在系统调用表中添加系统调用

打开 syscalls/syscall_64.tbl 在末尾添加对应的系统调用号

（2）.在头文件中声明函数

打开 include/linux/syscalls.h，在末尾添加函数声明：

asmlinkage int sys_memsys(int mypid);

*syscalls.h

```
                                    const struct iovec __user *rvec,
                                    unsigned long riovcnt,
                                    unsigned long flags);
asmlinkage long sys_process_vm_writev(pid_t pid,
                                     const struct iovec __user *lvec,
                                     unsigned long liovcnt,
                                     const struct iovec __user *rvec,
                                     unsigned long riovcnt,
                                     unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                              unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
                             const char __user *const __user *argv,
                             const char __user *const __user *envp, int flags);

#endif

asmlinkage int sys_memsys(int mypid);
```

（3）.添加函数原型

kernel/sys.c

sys.c　syscalls.h　syscall_64.tbl

```
2454
2455  asmlinkage int sys_memsys(int mypid)
2456  {
2457      //获取子进程的进程描述符
2458      struct pid * kpid=find_get_pid(mypid);
2459      //获取进程所属的任务的任务描述符
2460      struct task_struct * task=pid_task(kpid, PIDTYPE_PID);
2461      //获取任务对应进程的进程描述符
2462      pid_t result1=__task_pid_nr_ns(task,PIDTYPE_PID,kpid->numbers[kpid->level
2463      //获取任务的内存描述符
2464      struct mm_struct * mm_task=get_task_mm(task);
2465      struct vm_area_struct * vm_area = mm_task->mmap;
2466      int k = 0;
2467      while(vm_area != NULL && k <10000){
2468          printk("<%d> logical addr: %p,  physical addr: %p\n", k, vm_area, __
2469          vm_area = vm_area->vm_next;
2470          k++;
2471      }
2472      printk("<k> the mm_count of the mm_area_struct is: %d\n",mm_task->map_cou
2473      return 0;
2474  }
2475
2476
```

（4）. 重新编译安装内核

（5）. C 语言测试文件: memsys.c

```c
// memsys.c
#include <stdio.h>
#include <stdlib.h>

#define _MEMSYS_CALL_ 392    //系统调用号

int main(int argc,char* argv[]){
    if(argc <= 1){
        printf("Params Error!");
    }
    int pid;
    sscanf(argv[1], "%d", &pid);
    printf("进程号: %d\n", pid);
    int tmp = syscall(_MEMSYS_CALL_, pid);
    return 0;
}
```

（6）.执行测试代码

编译：gcc memsys.c -o memsys

执行：./memsys #进程号

测试结果

```
ycnalin@ubuntu:~/Desktop$ gedit memsys_test.c &
[2] 2499
[1]   Done                    gedit memsys_test.c
ycnalin@ubuntu:~/Desktop$ ./demo "2499"
进程号: 2499
ycnalin@ubuntu:~/Desktop$ dmesg | grep logical addr
grep: addr: No such file or directory
ycnalin@ubuntu:~/Desktop$ dmesg | grep "logical addr"
[  243.415309] <0> logical addr: ffff8800a35cb4d8,  physical addr: 00000000a35cb4d8
[  243.415312] <1> logical addr: ffff8800a35cacf0,  physical addr: 00000000a35cacf0
[  243.415313] <2> logical addr: ffff8800ac9d5ee8,  physical addr: 00000000ac9d5ee8
[  243.415313] <3> logical addr: ffff8800ac9d5c08,  physical addr: 00000000ac9d5c08
[  243.415314] <4> logical addr: ffff8800a3a0cc38,  physical addr: 00000000a3a0cc38
[  243.415315] <5> logical addr: ffff8800a3a0c0b8,  physical addr: 00000000a3a0c0b8
[  243.415316] <6> logical addr: ffff8800ac9d5700,  physical addr: 00000000ac9d5700
[  243.415316] <7> logical addr: ffff8800ac9d5590,  physical addr: 00000000ac9d5590
[  243.415317] <8> logical addr: ffff8800ac9d6ac8,  physical addr: 00000000ac9d6ac8
[  243.415318] <9> logical addr: ffff8800ac9d72b0,  physical addr: 00000000ac9d72b0
[  243.415318] <10> logical addr: ffff8800ac9d74d8,  physical addr: 00000000ac9d74d8
[  243.415319] <11> logical addr: ffff8800ac9d7648,  physical addr: 00000000ac9d7648
[  243.415320] <12> logical addr: ffff8800ac9d79e0,  physical addr: 00000000ac9d79e0
[  243.415320] <13> logical addr: ffff8800ac9d7a98,  physical addr: 00000000ac9d7a98
[  243.415321] <14> logical addr: ffff8800ac9d6958,  physical addr: 00000000ac9d6958
[  243.415322] <15> logical addr: ffff8800ac9d7700,  physical addr: 00000000ac9d7700
```

打印gedit编辑器进程物理页面

```
ycnalin@ubuntu:~/Desktop$ dmesg | grep "mm_count"
[  243.415907] <k> the mm_count of the mm_area_struct is: 531
[  467.720123] <k> the mm_count of the mm_area_struct is: 531
```