Facultad de ingeniería
Escuela de Sistemas
Sistemas Operativos 1

Proyecto 1

Manual de módulos de CPU y Memoria

Pedro Luis Garcia Chavarria - 200721968
Jenny Sharai Montenegro Contreras - 201318664

Creacion de modulos

structs que se Utilizaron para leer la informacion del kernel.

Sysinfo

```
struct sysinfo {
   long uptime;             /* Seconds since boot */
   unsigned long loads[3];  /* 1, 5, and 15 minute load averages */
   unsigned long totalram;  /* Total usable main memory size */
   unsigned long freeram;   /* Available memory size */
   unsigned long sharedram; /* Amount of shared memory */
   unsigned long bufferram; /* Memory used by buffers */
   unsigned long totalswap; /* Total swap space size */
   unsigned long freeswap;  /* Swap space still available */
   unsigned short procs;    /* Number of current processes */
   char _f[22];             /* Pads structure to 64 bytes */
};
```

Retorna informacion sobre la memoria RAM y la memoria SWAP

task_struct

```
struct task_struct {
/* these are hardcoded - don't touch */
  volatile long       state;        /* -1 unrunnable, 0 runnable, >0 stopped */
  long                counter;
  long                priority;
  unsigned            long signal;
  unsigned            long blocked;   /* bitmap of masked signals */
  unsigned            long flags;     /* per process flags, defined below */
  int errno;
  long                debugreg[8];    /* Hardware debugging registers */
  struct exec_domain   *exec_domain;
/* various fields */
  struct linux_binfmt  *binfmt;
  struct task_struct   *next_task, *prev_task;
  struct task_struct   *next_run,  *prev_run;
  unsigned long       saved_kernel_stack;
  unsigned long       kernel_stack_page;
  int                 exit_code, exit_signal;
  /* ??? */
  unsigned long       personality;


     int              dumpable:1;
     int              did_exec:1;
     int              pid;
     int              pgrp;
     int              tty_old_pgrp;
     int              session;
     /* boolean value for session group leader */
     int              leader;
     int              groups[NGROUPS];
     /*
      * pointers to (original) parent process, youngest child, younger sibling,
      * older sibling, respectively.  (p->father can be replaced with
      * p->p_pptr->pid)
      */
```

```
  struct task_struct   *p_opptr, *p_pptr, *p_cptr,
                       *p_ysptr, *p_osptr;
  struct wait_queue    *wait_chldexit;
  unsigned short       uid,euid,suid,fsuid;
  unsigned short       gid,egid,sgid,fsgid;
  unsigned long        timeout, policy, rt_priority;
  unsigned long        it_real_value, it_prof_value, it_virt_value;
  unsigned long        it_real_incr, it_prof_incr, it_virt_incr;
  struct timer_list    real_timer;
  long                 utime, stime, cutime, cstime, start_time;
/* mm fault and swap info: this can arguably be seen as either
   mm-specific or thread-specific */
  unsigned long        min_flt, maj_flt, nswap, cmin_flt, cmaj_flt, cnswap;
  int swappable:1;
  unsigned long        swap_address;
  unsigned long        old_maj_flt;    /* old value of maj_flt */
  unsigned long        dec_flt;        /* page fault count of the last time */
  unsigned long        swap_cnt;       /* number of pages to swap on next pass */
/* limits */
  struct rlimit        rlim[RLIM_NLIMITS];
  unsigned short       used_math;
  char                 comm[16];
/* file system info */
  int                  link_count;
  struct tty_struct    *tty;           /* NULL if no tty */
/* ipc stuff */
  struct sem_undo      *semundo;
  struct sem_queue     *semsleeping;
/* ldt for this task - used by Wine.  If NULL, default_ldt is used */
  struct desc_struct *ldt;
/* tss for this task */
```

```
  struct files_struct  *files;
/* memory management info */
  struct mm_struct     *mm;
/* signal handlers */
  struct signal_struct *sig;
#ifdef __SMP__
  int              processor;
  int              last_processor;
  int              lock_depth;     /* Lock depth.
                                      We can context switch in and out
                                      of holding a syscall kernel lock... */
#endif
};
```

Describe un proceso o tarea en el sistema.

Para crear un modulo se definen basicamente dos funciones importantes que son las que se utilizan para su carga y descarga, esto son el init y exit que se declararian asi
module_init(nombre _funcion)
module_exit(nombre_funcion)

1) Para poder ejecutar estos modulos debemos de crear un Makefile para cada uno de ellos con el siguiente codigo, en el nombre del objeto salida variamos cpu por memo:
2) Una vez guardamos los Makefile ingresamos el siguiente comando “make” y generara un archivo con extension ko que sera el que cargaremos al kernel.

3) Para cargar los modulos al kernel basta con introducir el siguiente comando “insmod nombre.ko”.

4) Una vez ya no sea necesario el modulo podemos descargarlo con el siguiente comando “rmmod nombre.ko”

5) Para ver los mensajes de salida al momento de la carga y descarga utilizaremos el comando “dmesg”

```
obj-m += (cpu|mem)_201318664_200721968.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules



clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

memo_201318664_200721968.c

```
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <asm/uaccess.h>
#include <linux/hugetlb.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>

#define BUFSIZE         150

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Escribir informacion de la memoria RAM.");
MODULE_AUTHOR("Jenny Montenegro - Pedro Garcia ");
```

```
struct sysinfo inf;

static int escribir_archivo(struct seq_file * archivo, void *v) {
   si_meminfo(&inf);
   long total_memoria          = (inf.totalram * 4);
   long memoria_libre          = (inf.freeram * 4 );
   seq_printf(archivo,
"*************************************************\n");
   seq_printf(archivo, "* Laboratorio Sistemas Operativos 1            *\n");
   seq_printf(archivo, "* Vacaciones de Junio 2020                     *\n");
   seq_printf(archivo, "* - Jenny Montenegro                          *\n");
   seq_printf(archivo, "* 201318664                                    *\n");
   seq_printf(archivo, "* - Pedro Garcia                               *\n");
   seq_printf(archivo, "* 200721968                                    *\n");
   seq_printf(archivo, "*                                      *\n");
   seq_printf(archivo, "*    PROYECTO1  *MODULO 1 - MEMORIA RAM*
*\n");
   seq_printf(archivo,
"*************************************************\n");
   seq_printf(archivo, "  Memoria Total : \t %8lu KB - %8lu
MB\n",total_memoria, total_memoria / 1024);
   seq_printf(archivo, "  Memoria Libre : \t %8lu KB - %8lu MB \n",
memoria_libre, memoria_libre / 1024);
   seq_printf(archivo, "  Memoria en uso: \t %i %%\n", (memoria_libre *
100)/total_memoria) ;
   return 0;
}

static int al_abrir(struct inode *inode, struct  file *file) {
  return single_open(file, escribir_archivo, NULL);
}

static struct file_operations operaciones =
{
   .open = al_abrir,
   .read = seq_read
};

static int iniciar(void)
{
   proc_create("memo_201318664_200721968", 0, NULL, &operaciones);
```

```
    printk(KERN_INFO "Carnet: 201318664/200721968\n");
    return 0;
}

static void salir(void)
{
    remove_proc_entry("memo_201318664_200721968", NULL);
    printk(KERN_INFO "Curso: Sistemas Operativos 1\n");
}

module_init(iniciar);
module_exit(salir);
```

cpu_201318664_200721968.c

```
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/list.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/seq_file.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/types.h>


void readProcess(struct seq_file *m, struct task_struct *s)
{
    struct list_head *list;
    struct task_struct *task;

    char estado[50];

    switch(s->state){
        case TASK_RUNNING:
    strcpy(estado,"Ejecucion");
    break;

case TASK_STOPPED:
```

```
            strcpy(estado,"Detenido");
    break;

case TASK_INTERRUPTIBLE:
            strcpy(estado,"Interrumpible");
    break;

case TASK_UNINTERRUPTIBLE:
            strcpy(estado,"Ininterrumpible");
    break;

case EXIT_ZOMBIE:
            strcpy(estado,"Zombi");
    break;

default:
    strcpy(estado, "Desconocido");
    }

    seq_printf(m,"PID: %d\t\tNombre: %s\t\tEstado:%s\n",s->pid, s->comm, estado);

    list_for_each(list, &s->children) {
        task = list_entry(list, struct task_struct, sibling);
        readProcess(m, task);
    }
}

static int pstree(struct seq_file *m, void *v)
{
    struct task_struct *parent = current;
    while (parent->pid != 1)
        parent = parent->parent;
    readProcess(m, parent);
    return 0;
}

static int meminfo_proc_open(struct inode *inode, struct file *file)
{
    return single_open(file, pstree, NULL);
}

static const struct file_operations meminfo_proc_fops = {
```

```
    .open       = meminfo_proc_open,
    .read       = seq_read,
    .llseek     = seq_lseek,
    .release    = single_release,
};

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Pedro Garcia/Jenny Montenegro");
MODULE_DESCRIPTION("Modulo de CPU - Sistemas Operativos 1 Junio 2020");

static int __init cpu_201318664_200721968_init(void)
{
printk(KERN_INFO "Pedro Garcia/Jenny Montenegro \n");
proc_create("cpu_201318664_200721968", 0, NULL, &meminfo_proc_fops);
return 0;
}

static void __exit cpu_201318664_200721968_cleanup(void)
{
remove_proc_entry("cpu_201318664_200721968", NULL);
printk(KERN_INFO "Sistemas Operativos 1\n");
}

module_init(cpu_201318664_200721968_init);
module_exit(cpu_201318664_200721968_cleanup);
```