**PRACTICA 2: MODIFICACIONS AL KERNEL**

**Flavio Ferreira Machado**

Abans de començar a documentar una mica la pràctica he de comentar d'un cert problema que he tingut, disposo d'un portatil de 2GB de RAM amb ubuntu instal·lat natiu, la primera practica he pogut fer perfectament les coses desde la maquina que tenia, virtualizar qemu, etc… El problema ve quan no sabia com compilar els arxius en C ja que tenia d'un sistema operatiu diferent i llavors quan compilaba y pasaba l'arxiu a la maquina guest em sortia errors als arxius teus que testegen la práctica, llavors, el problema és que amb 2GB de RAM no podía al meu linux virtualizar el linux que es recomana per a la práctica y després a aquest instal·lar el qemu per a fer les pràctiques. Dit això procedeixo a l'explicació de la, o almenys de les parts que he pogut implementar.

Tot i que el codi estigui comentat, posaré captures de les parts més importants de cada part.

**1 - Implementació nova crida del sistema**

**1.3 - Lliuraments corresponents a aquest exercici**
**Els lliuraments corresponents a aquest exercici són:**
**• El codi font del mòdul que implementa la nova crida al sistema.**
**• Breu explicació de la feina feta (format pdf, incloent captura de pantalla del resultat del programa de proves).**

```
//tractament dels casos trivials

// type = 0 --> calcular NUM_CHILDREN
// type = 1 --> calcular NUM_SIBLINGS
// type != 0 o 1 --> retornar error, tipus d'operació no correcte
// si el tipus de operació no es correcte, codi error -EINVAL
if((type != NUM_CHILDREN) || (type != NUM_SIBLINGS)){
  return -EINVAL;
}

//si el pid no esta definit
if(!pid){
  return -ESRCH;
}

//busco si existeix el pid que s'ha pasat com a parametre
struct task_struct *task = NULL;
task = pid_task(find_vpid(pid), PIDTYPE_PID);

// si el process que s'especifica no existeis, codi error -ESRCH
if(!task){
  return -ESRCH;
}
```

```c
// contadors per a guardar els valors de children i siblings
// i que quan un process no tingui ni childrens ni sibligns que retorni 0
// com a cas trivial que demanes, per aixo els inicio a 0
int numero_childrens = 0;
int numero_siblings = 0;

// crear structura de llista per a iterar sobre els posibles children i siblings de la task
struct list_head *list;

if(type == NUM_CHILDREN){
  //si type = 0; calcular NUM_CHILDREN
  list_for_each(list, &(task->children)){
    numero_childrens++;
  }
  return numero_childrens;
} else if (type == NUM_SIBLINGS){
  //si type = 1; calcular NUM_SIBLINGS
  list_for_each(list, &(task->siblings)){
    numero_siblings++;
  }
  return numero_siblings;
}
```

**2 - Implementació d'un driver**

**2.3 - Lliuraments corresponents a aquest exercici**

**Els lliuraments corresponents a aquest exercici són:**

**• El codi font del mòdul que implementa el driver.**

**• Breu explicació de la feina feta (format pdf, incloent captura de pantalla del resultat del programa de proves).**

```
open --> ok
read -->
lseek --> ok
ioctl -->
write --> ok
release --> ok
```

```c
// Varaible per a controlar els accessos al fitxer
static int Device_Open = 0; //  dispositiu ja obert? utilitzat per a prevenir multiples accesos

// Variable semafor per a cambiar les varaibles critiques
static struct semaphore sem;
```

```c
int
do_open (struct inode *inode, struct file *file)
{
  // Si variable == 1, recurs ocupat
  if(Device_Open == 1)
    return -EBUSY;

  /* Just O_RDONLY mode */
  if (file->f_flags != O_RDONLY)
    return -EACCES;

  /* Utilitzo semafor per a cambiar la variable critica Device_Open */
  down(&sem);// V
  // Posar variable a 1 per a que bloqueji altres accessos
  Device_Open++;
  up(&sem);// P

  return 0;
}
```

```c
/* close system call */
int
do_release (struct inode *inode, struct file *file)
{
  /* Utilitzo semafor per a cambiar la variable critica Device_Open */
  down(&sem);// V
  // Posar variable a 0 un altre cop per a es permeti accedir un altre cop
  Device_Open--;
  up(&sem);// P

  return 0;
}
```

```c
static loff_t
do_llseek (struct file *file, loff_t offset, int orig)
{
  loff_t ret;

  switch (orig)
    {
    case SEEK_CUR:
      ret = file->f_pos + offset;
      break;
    case SEEK_SET:
      ret = offset;
      break;
    default:
      // si orig es igual a SEEK_END --> llavors retorna -EINVAL, no permes
      ret = -EINVAL;
    }

  if (ret >= 0)
    file->f_pos = ret;
  else
    ret = -EINVAL;

  return ret;
}
```

```c
ssize_t
do_write (struct file * file, const char *buf, size_t count, loff_t * f_pos)
{
  // no fa falta implementar
  return 0;
}
```