## Exercício Programa 1 – Relatório

MAC0422 – Sistemas Operacionais

André Spanguero Kanayama 7156873 Pedro Paulo Vezzá Campos 7538743

16 de setembro de 2013

## 1 Enunciado

Para este primeiro exercício-programa de MAC0422 – Sistemas Operacionais, o professor requisitou que os alunos alterassem o comportamento do comando F5 no Minix. Após pressionar esta tecla deveria ser exibido na tela um resumo da tabela de processos:

- 1. Process ID
- 2. Tempo de CPU
- 3. Tempo de sistema
- 4. Tempo dos filhos
- 5. Endereço do ponteiro da pilha
- 6. Endereço dos segmentos data, bss, text

## 2 Descoberta do comportamento do comando F5

Após uma busca na Internet por "Minix function keys", o primeiro resultado indicou o primeiro arquivo importante para o EP: /usr/src/servers/is/dmp.c. Nele, havia uma estrutura bastante intuitiva, um mapeamento de cada tecla para sua respectiva função. Aqui foi feita a primeira modificação:

```
1 struct hook_entry {
2   int key;
3   void (*function)(void);
4   char *name;
5 } hooks[] = {
6   { F1,   proctab_dmp, "Kernel process table" },
7   { F2,   memmap_dmp, "Process memory maps" },
8   { F3,  image_dmp,  "System image" },
9   { F4,   privileges_dmp,  "Process privileges" },
```

```
{ F5, monparams_dmp, "Boot monitor parameters" },
11 -
        12 >
        13 >
      { F5, pt_dmp, "Print process table" },
14 >
        15 >
        16 >
17
     F6, irqtab_dmp, "IRQ hooks and policies" },
18
      F7, kmessages_dmp, "Kernel messages" },
19
      F8, vm_dmp, "VM status and process maps"
20
      F10.
           kenv_dmp, "Kernel parameters" },
           \begin{array}{lll} timing\_dmp\,, & "Timing & details & (if & enabled)\," \end{array} \}\,, \\ mproc\_dmp\,, & "Process & manager & process & table\," \end{array} \}\,,
21
      F11,
^{22}
      SF1,
           sigaction_dmp, "Signals" },
23
      SF2,
           fproc_dmp, "Filesystem process table" },
24
      SF3,
           dtab_dmp, "Device/Driver mapping" },
25
      SF4.
           mapping_dmp, "Print key mappings" },
26
      SF5.
           rproc_dmp, "Reincarnation server process table" },
27
      SF6,
           data_store_dmp, "Data store contents" },
procstack_dmp, "Processes with stack traces" },
28
      SF8,
29
      SF9,
30 };
```

Vasculhando o diretório /usr/src/servers/is verificamos que todo o EP pode ser feito através de modificações no *Information Server* (IS).

## 3 Desenvolvimento da função pt\_dmp

A função pt\_dmp, de process table dump, é o trecho de código principal do EP. Ela foi derivada da função mproc\_dmp, que já estava implementada no Minix e é responsável por exibir a tabela de processos do Process Manager (PM).

Passamos a estudar as diferentes tabelas de processos existentes no Minix. A primeira relevante para este trabalho é a tabela de processos do PM, definida no arquivo /usr/src/servers/pm/mproc.h. Nela, encontramos parte das informações necessárias ao EP:

**Process ID** Definido no campo pid\_t mp\_pid;

Tempo dos filhos Definido no campo clock\_t mp\_child\_utime;

Ao perceber que as outras informações necessárias não estavam disponíveis neste local passamos a vasculhar o código em busca de outras tabelas úteis. Após ver o código de impressão da tabela de processos do kernel que está no arquivo /usr/src/servers/is/kernel\_dmp.c encontramos o arquivo header da tabela de processos em /usr/src/kernel/proc.h. Neste arquivo estavam presentes as outras informações necessárias:

Tempo de CPU Disponível no campo clock\_t p\_user\_time;

Tempo de sistema Disponível no campo clock\_t p\_sys\_time;

Endereço do ponteiro da pilha Disponível no campo p\_memmap[S].mem\_phys;

Endereço do segmento data Disponível no campo p\_memmap[D].mem\_phys;

Endereço do segmento text Disponível no campo p\_memmap[T].mem\_phys;

Endereço do segmento bss O endereço do segmento bss é definido para arquivos executáveis ELF como sendo o primeiro endereço de memória após o segmento data.

Assim, ele pode ser calculado como sendo p\_memmap[D].mem\_phys + p\_memmap[D].mem\_len;

O código final para a função pt\_dmp foi inserido no arquivo /usr/src/servers/is/dmp\_pm.c e está descrito abaixo:

```
3
4 PUBLIC void pt_dmp()
5 {
6
   struct mproc *mp;
7
   int i, n=0;
8
   int result;
9
   phys_bytes p;
10
   static int prev_i = 0;
11
12
   struct proc *pr;
13
14
   printf("Process table\n");
15
   getsysinfo(PM_PROC_NR, SI_PROC_TAB, mproc);
16
17
   sys_getproctab(lproc);
18
19
   printf("-pid--cpu_t--sys_t--chld_t----stackpointer---data---
      ---bss--- ---text---\n");
20
   for (i=prev_i; i<NR_PROCS; i++) {</pre>
21
    pr = &lproc[i];
22
     if(pr->p_nr < 0)
23
        continue;
24
     mp = &mproc[pr->p_nr];
25
     if (mp->mp_pid == 0 && i != PM_PROC_NR) continue;
26
     if (++n > 22) break;
27
     28
29
      mp->mp_pid,
30
       pr->p_user_time,
31
       pr->p_sys_time,
32
       mp->mp_child_utime,
33
       pr->p_memmap[S].mem_phys,
34
       pr->p_memmap[D].mem_phys,
35
       pr->p_memmap[D].mem_vir + pr->p_memmap[D].mem_len,
36
       pr->p_memmap[T].mem_phys);
37
     printf("\n");
38
   }
39
   if (i \ge NR_PROCS) i = 0;
```

Por fim, foi necessário editar o arquivo de protótipos de funções do IS para incluir a nova função criada. A edição foi feita no arquivo /usr/src/servers/is/proto.h e está reproduzida abaixo: