Exercício Programa 1 – Relatório

MAC0422 – Sistemas Operacionais

André Spanguero Kanayama 7156873
Pedro Paulo Vezzá Campos 7538743

18 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 Observações

- O professor informou verbalmente aos alunos na aula do dia 16/09 que não era mais necessário imprimir a tabela de processos segundo a ordem de escalonamento.
- A versão do Minix escolhida para este EP foi a versão 3.1.7 devido à sua melhor compatibilidade com o VirtualBox e maior semelhança com a versão 3.1.0, a utilizada pelo livro-texto da disciplina.

3 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" },
10 <   { F5, monparams_dmp, "Boot monitor parameters" },
11 ---
12 >     /*??????????????????????????????????????????????????????????*/
13 >     /*??????????????????????????????????????????????????????????*/
14 >   { F5, pt_dmp, "Print process table" },
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" },
21   { F11,  timing_dmp, "Timing details (if enabled)" },
22   { SF1,  mproc_dmp, "Process manager process table" },
23   { SF2,  sigaction_dmp, "Signals" },
24   { SF3,  fproc_dmp, "Filesystem process table" },
25   { SF4,  dtab_dmp, "Device/Driver mapping" },
26   { SF5,  mapping_dmp, "Print key mappings" },
27   { SF6,  rproc_dmp, "Reincarnation server process table" },
28   { SF8,  data_store_dmp, "Data store contents" },
29   { SF9,  procstack_dmp, "Processes with stack traces" },
30 };
```

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

4 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 como sendo o primeiro endereço de memória após o segmento `data`. Tentamos calcular este endereço usando endereços virtuais com a seguinte fórmula:

`p_memmap[D].mem_vir + p_memmap[D].mem_len`

Em seguida mapeando-o para o seu respectivo endereço físico através da função `sys_umap`. Porém, a execução chamada de sistema foi negada. O *kernel* informava o seguinte erro:

`SYSTEM: denied request 14 from 73137.`

Assim, contornamos este problema realizando o cálculo através da seguinte fórmula:

`p_memmap[D].mem_phys + p_memmap[D].mem_len`

É importante ressaltar que apesar de representarem os mesmos processos (salvo algumas exceções), as entradas de ambas tabelas não estão na mesma ordem. Para resolver este problema o próprio Minix apresenta um mapeamento na tabela de processos do *kernel* de uma entrada nesta tabela para a tabela de processos do PM. Esta informação foi descoberta após vasculhar o código do programa `top`. O campo relevante é o campo `p_nr`. Processos gerenciados pelo *kernel* mas não pelo PM possuem o campo `p_nr` < 0. Dessa forma, definimos a regra que para um dado processo `proc[i]` com `proc[i].p_nr` ≥ 0 e $0 \leq$ `i` $<$ `NR_PROCS` é equivalente ao processo `mproc[proc[i].p_nr]`. Para garantir que esse mapeamento é válido, fizemos um teste imprimindo o nome dos processos equivalentes em ambas tabelas. O experimento foi bem sucedido.

O código final para a função `pt_dmp` foi inserido no arquivo `/usr/src/servers/is/dmp_pm.c` e está descrito abaixo:

```
/*??????????????????????????????????????????????????????????????????????*/
/*??????????????????????????????????????????????????????????????????????*/

PUBLIC void pt_dmp()
{
  struct mproc *mp;
  int i, n=0;
  int result;
  phys_bytes p;
  static int prev_i = 0;

  struct proc *pr;

  printf("Process table\n");

  getsysinfo(PM_PROC_NR, SI_PROC_TAB, mproc);
  sys_getproctab(lproc);

  printf("-pid- -cpu_t- -sys_t- -chld_t- ---stackpointer ---data---
      ---bss---  ---text---\n");
  for (i=prev_i; i<NR_PROCS; i++) {
    pr = &lproc[i];
    if(pr->p_nr < 0)
        continue;
    mp = &mproc[pr->p_nr];
    if (mp->mp_pid == 0 && i != PM_PROC_NR) continue;
    if (++n > 22) break;

    printf("%4d  %6d %7d %8d      0x%08X  0x%08X 0x%08X 0x%08X",
      mp->mp_pid,
      pr->p_user_time,
      pr->p_sys_time,
      mp->mp_child_utime,
      pr->p_memmap[S].mem_phys,
      pr->p_memmap[D].mem_phys,
      pr->p_memmap[D].mem_phys + pr->p_memmap[D].mem_len,
      pr->p_memmap[T].mem_phys);
    printf("\n");
  }
  if (i >= NR_PROCS) i = 0;
  else printf("--more--\r");
  prev_i = i;
}
/*??????????????????????????????????????????????????????????????????????*/
/*??????????????????????????????????????????????????????????????????????*/
```

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:

```
/* dmp_pm.c */
_PROTOTYPE( void mproc_dmp, (void)           );
_PROTOTYPE( void sigaction_dmp, (void)           );
> /*??????????????????????????????????????????????????????????*/
```

```
5 > /*????????????????????????????????????????????????????????????*/
6 > _PROTOTYPE( void pt_dmp, (void)          );
7 > /*????????????????????????????????????????????????????????????*/
8 > /*????????????????????????????????????????????????????????????*/
```