Relatório: EP1

André Meneghelli Vale - 4898948
andredalton@gmail.com

Hilder Vitor Lima Pereira - 6777064
vitor_lp@yahoo.com

01 de Setembro

1 Introdução

1.1 Minix

O *Minix* é um sistema operacional semelhante ao *Unix* gratuito e com o código fonte disponível em suas distribuições. É escrito em *Linguagem C* e *assembly. Andrew S. Tanenbaum* criou este sistema para explicar os principios de funcionamento de seu livro *"Operating Systems Design and Implementation"*.

As vantagens deste sistema, além de ser disponibilizado com o seu código fonte, são a necessidade muito reduzida de memória RAM e disco rígido quando comparado aos sistemas operacionais utilizados atualmente e uma arquitetura interessante para o aprendizado. Uma vez que os processos são entidades independentes e estão restritos a camadas, cada processo tem as suas permissões de acesso e algumas propriedades.

É possível encontrar mais informações e baixar as várias versões disponíveis deste sistema em `http://www.minix3.org/`. Para este trabalho foi escolhida a versão 3.1.7.

1.2 Virtualbox

Para facilitar a instalação e distribuição das alterações necessárias para este Exercício foi estipulado o uso do software de virtualização Virtualbox.

Este software é gratuito e compatível com vários sistemas operacionais atuais. Outra característica importante é a capacidade de criação de pastas compartilhadas entre a maquina virtual e o hospedeiro, o que facilita muito a criação de um bom ambiente de programação.

Para realizar a configuração do sistema operacional de maneira mais conveniente optamos por configurar a placa de rede da VM em modo NAT e usamos redirecionamento de portas para fazer conexão ssh. Permitindo desta maneira que todo o Minix fosse editado usando as ferramentas preferidas de cada um dos integrantes do grupo.

1.3 Problema proposto

Modificar o sistema, fazendo com que um resumo da tabela de processos seja mostrada quando a tecla `F5` for acionada. Este resumo deve conter as informações na ordem da lista a seguir:

- PID: identificador do processo;
- Tempo de cpu;
- Tempo de sistema;
- Tempo dos filhos;
- Endereço do ponteiro da pilha e dos segmentos data, bss e text;

2 Códigos alterados

2.1 dmp.c

Este arquivo contém o mapeamento de caracteres, foi usado para poder tratar a captura de interrupção da tecla *F5*.

2.1.1 Localização

Diretório: /usr/src/servers/is/

2.1.2 Alterações [23-27]

Modificando a captura da interrupção da tecla *F5*.

```
23  /*??????????????????????????????????????????????????????*/
24  /*??????????????????????????????????????????????????????*/
25  { F5, custom_proctab_dmp, "Impressao de processos customizada." },
26  /*??????????????????????????????????????????????????????*/
27  /*??????????????????????????????????????????????????????*/
```

dmp.c

2.2 proto.h

Arquivo com os protótipos das funções usadas no arquivo *dmp_kernel.c*.

2.2.1 Localização

Diretório: /usr/src/servers/is/

2.2.2 Alterações [14-18]

Permitindo acesso a função extra custom_proctab_dmp ( ).

```
14 /*??????????????????????????????????????????????????????*/
15 /*??????????????????????????????????????????????????????*/
16 _PROTOTYPE( void custom_proctab_dmp, (void)          );
17 /*??????????????????????????????????????????????????????*/
18 /*??????????????????????????????????????????????????????*/
```

proto.h

2.3 /usr/src/servers/is/dmp_kernel.c

Este arquivo contém a função alterada *custom_proctab_dmp()* que faz a impressão dos processos.

2.3.1 Localização

Diretório /usr/src/servers/is/

2.3.2 Alterações [14-18] [415-481]

A alteração a seguir serve apenas para ter acesso a estrura mproc dentro do escopo do arquivo atual.

```
14 /*??????????????????????????????????????????????????????*/
15 /*??????????????????????????????????????????????????????*/
16 #include "servers/pm/mproc.h"
17 /*??????????????????????????????????????????????????????*/
18 /*??????????????????????????????????????????????????????*/
```

dmp_kernel.c

A função a seguir utiliza acesso as estruturas proc e mproc para poder imprimir as informações necessárias para a tarefa.

```
415 /*??????????????????????????????????????????????????????*/
416 /*??????????????????????????????????????????????????????*/
417 void custom_proctab_dmp(){
418   register struct proc *rp;
419   register struct mproc mproc[NR_PROCS];
420   static int pg = 0;
421   int i, j=0, k;
422
423   /* Pegando uma c pia atualizada da tabela de processos. */
424   if (sys_getproctab(proc) != OK) {
425     printf("IS: warning: couldn't get copy of process table\n");
426     return;
427   }
428
429   /* pegando c pia atualizada da mproc table */
430   if (getsysinfo(PM_PROC_NR, SI_PROC_TAB, mproc) != OK) {
431     printf("Error obtaining table from PM. Perhaps recompile IS?\n");
432     return;
433   }
434
435   printf("\nPID\tCPU\tSYS\tFTIME\tEPILHA\tDATA\tBSS\tTEXT\tNAME");
436
437   for (i=NR_TASKS, j=0; i<(NR_TASKS+NR_PROCS); i++) {
438     if (! isemptyp (&(proc[i]))){
439       /* Imprime quando est  na p gina correta. */
440       if ( j/LINES == pg ) {
441         printf(
442             "\n%03d"
443             "\t%d"
444             "\t%d"
445             "\t%d"
446             "\t0x%X"
447             "\t0x%X"
448             "\t0x%X"
449             "\t0x%X"
```

```
450                 "\t%s",
451                 (int)mproc[i - NR_TASKS].mp_pid,
452
453                 (int)proc[i].p_user_time,
454                 (int)proc[i].p_sys_time,
455                 (int)mproc[i - NR_TASKS].mp_child_stime,
456                 proc[i].p_memmap[S].mem_phys,
457                 proc[i].p_memmap[D].mem_phys,
458                 proc[i].p_memmap[D].mem_phys + proc[i].p_memmap[D].mem_len,
459                 proc[i].p_memmap[T].mem_phys,
460                 proc[i].p_name
461             );
462           }
463           /* Se ultrapassou a p gina atual precisa trocar de p gina e parar o
      la o. */
464           else if ( j/LINES > pg ) {
465             pg++;
466             j--;
467             break;
468           }
469           j++;
470         }
471       }
472
473       /* Aqui est  parte do controle de fluxo do sistema de pagina  o. */
474       while (j%(LINES+1)!=0) {
475         printf("\n");
476         j++;
477       }
478       if ( i >= NR_TASKS+NR_PROCS ) pg = 0;
479     }
480     /*????????????????????????????????????????????????*/
481     /*????????????????????????????????????????????????*/
```

dmp_kernel.c

3 Conclusão

Durante a resolução do problema proposto foi possível verificar superficialmente como o minix gerencia o acesso de múltiplos processos aos recursos da máquina. E também verificar como se dá o endereçamento de memória RAM para este sistema operacional.

O trabalho foi feito dentro da camada de serviços. E não houve necessidade de enviar ou receber mensagens para outros processos.

Para facilitar a visualização se criou um sistema de paginação, permitindo que a página de processos a ser mostrada seja alterada de maneira circular conforme se pressiona a tecla *F5*.

Encontramos algumas dificuldades na instalação, principalmente quanto a instruções de processos que não estão presentes em todas as arquiteturas de processados testadas. E também variações significativas quanto ao que está presente no livro e ao que está presente no minix. Felizmente todas as dúvidas extras foram sanadas ao se analisar a documentação existente dentro do próprio código fonte. Também pudemos contar com uma boa documentação online.

Podemos concluir que, embora rudmentar, este sistema operacional é sem dúvidas uma boa ferramenta para o aprendizado de sistemas operacionais.

Referências

[1] Andrew S. Tanenbaum, "Operating Systems Design and Implementation, 3a. ed.", pp. 112 - 213, January 14, 2006.