Relatório: EP2

André Meneghelli Vale - 4898948
andredalton@gmail.com

Hilder Vitor Lima Pereira - 6777064
vitor_lp@yahoo.com

09 de Novembro

1 Introdução

1.1 Minix

O *Minix* é um sistema operacional gratuíto, com o código fonte disponível em suas distribuições e semelhante ao *Unix*. 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 gratuíto e compatível com vários sistemas operacionais atuais. Outra característica importante é a capacidade de criação de pastas compartilhadas entre a máquina 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 acessado e editado usando as ferramentas preferidas de cada um dos integrantes do grupo.

1.3 Problema proposto

Gerar um servidor de semaforos no PM permitindo que processos tenham acesso a ate 128 semaforos.

Sao necessarias a implementacao das seguintes chamadas de sistema:

- int get sem(int valor inicial);
- int p sem(int indice sem);
- int v sem(int indice sem);
- int free sem(int indice sem).

2 Códigos alterados

2.1 table.c

Este arquivo contém o mapeamento para as chamadas de sistema que o *PM* esta apto a responder.

2.1.1 Localização

Diretório: /usr/src/servers/pm/

2.1.2 Alterações [46-51], [55-60], [71-75]:

Criando o mapeamento para as chamadas *get_sem()* e *p_sem()*.

```
46  /*??????????????????????????????????????????????????*/
47  /*??????????????????????????????????????????????????*/
48  do_get_sem, /* 31 = acesso a novo semaforo.        */
49  do_p_sem,   /* 32 = funcao P do semaforo.          */
50  /*??????????????????????????????????????????????????*/
51  /*??????????????????????????????????????????????????*/
```

table.c

Criando o mapeamento para as chamadas *v_sem()* e *free_sem()*.

```
55  /*??????????????????????????????????????????????????*/
56  /*??????????????????????????????????????????????????*/
57  do_v_sem,       /* 34 = funcao V do semaforo.       */
58  do_free_sem,    /* 35 = funcao free do semaforo.    */
59  /*??????????????????????????????????????????????????*/
60  /*??????????????????????????????????????????????????*/
```

table.c

Criando o mapeamento para a chamada *wait_sem()*.

```
71  /*??????????????????????????????????????????????????*/
72  /*??????????????????????????????????????????????????*/
73  do_wait_sem,       /* 44 = funcao wait do semaforo.*/
74  /*??????????????????????????????????????????????????*/
75  /*??????????????????????????????????????????????????*/
```

table.c

2.2 proto.h

Este arquivo contém os prototipos para as funcoes que o *PM* esta apto a responder.

2.2.1 Localização

Diretório: /usr/src/servers/pm/

2.2.2 Alterações [63-72]:

Criando os prototipos *do_get_sem()*, *do_p_sem()*, *do_v_sem*, *do_free_sem* e *do_wait_sem* .

```
63 /*???????????????????????????????????????????????????????????*/
64 /*???????????????????????????????????????????????????????????*/
65 /* Criando os acessos das chamadas do servidor de semaforos. */
66 _PROTOTYPE( int do_get_sem, (void)                              );
67 _PROTOTYPE( int do_p_sem, (void)                               );
68 _PROTOTYPE( int do_v_sem, (void)                               );
69 _PROTOTYPE( int do_free_sem, (void)                            );
70 _PROTOTYPE( int do_wait_sem, (void)                            );
71 /*???????????????????????????????????????????????????????????*/
```

proto.h

2.3 callnr.h

Este arquivo contém as definicoes para as chamadas de sistema que o *PM* esta apto a responder.

2.3.1 Localização

Diretório: /usr/include/minix/
Diretório: /usr/src/include/minix/
Este arquivo esta disponível na fonte do minix e no diretório das bibliotecas, permitindo assim que a utilização das chamadas de sistema por qualquer parte compilada do minix ou qualquer programa que venha a ser compilado no sistema operacional já em funcionamento.

2.3.2 Alterações [34-39], [41-46], [55-59]:

Criando as definições *GET_SEM* e *P_SEM*.

```
34 /*???????????????????????????????????????????????????????????????????????*/
35 /*???????????????????????????????????????????????????????????????????????*/
36 #define GET_SEM         31 /* Requisicao para um novo semaforo.            */
37 #define P_SEM           32 /* Operacao P do semaforo.                      */
38 /*???????????????????????????????????????????????????????????????????????*/
39 /*???????????????????????????????????????????????????????????????????????*/
```

callnr.h

Criando o mapeamento para as chamadas *V_SEM* e *FREE_SEM*.

```
41 /*???????????????????????????????????????????????????????????????????????*/
42 /*???????????????????????????????????????????????????????????????????????*/
43 #define V_SEM         34 /* Operacao V do semaforo.                       */
44 #define FREE_SEM      35 /* Liberacao de um semaforo.                     */
45 /*???????????????????????????????????????????????????????????????????????*/
46 /*???????????????????????????????????????????????????????????????????????*/
```

callnr.h

Criando o mapeamento para a chamada *WAIT_SEM*.

```
55 /*???????????????????????????????????????????????????????????????????????*/
56 /*???????????????????????????????????????????????????????????????????????*/
57 #define WAIT_SEM       44 /* Aguarda que descendentes liberem o semaforo. */
58 /*???????????????????????????????????????????????????????????????????????*/
59 /*???????????????????????????????????????????????????????????????????????*/
```

callnr.h

2.4 main.c

Este arquivo contém a função principal do *PM*, é aqui que devemos verificar se um processo que morreu esta deixando algum semaforo inicializado.

2.4.1 Localização

Diretório: /usr/src/servers/pm/

2.4.2 Alterações [38-43] e [131-136]:

Inserindo o cabeçalho com o protótipo da unica função contida no *misc.c* que o PM deve acessar.

```
38 /*???????????????????????????????????????????????????????????????????????*/
39 /*???????????????????????????????????????????????????????????????????????*/
40 /* Incluindo cabecalho da misc.c para permitir acesso a funcao terminator(). */
41 #include "misc.h"
42 /*???????????????????????????????????????????????????????????????????????*/
43 /*???????????????????????????????????????????????????????????????????????*/
```

main.c

Chamando a função *terminator()* contida em *misc.c* quando o *PM* esta processando o término de um processo.

```
131
    /*????????????????????????????????????????????????????????????????????????*/
132
    /*????????????????????????????????????????????????????????????????????????*/
133         /* Chamando a funcao terminator() para todos os processos finalizados.
    */
134         if ( call_nr==PM_EXIT_REPLY ) terminator( _ENDPOINT_P(m_in.m1_i1));
135
    /*????????????????????????????????????????????????????????????????????????*/
136
    /*????????????????????????????????????????????????????????????????????????*/
```

main.c

2.5 fila.c

Este novo arquivo contém funções de manipulação de uma fila.

2.5.1 Localização

Diretório: /usr/src/servers/pm/

2.5.2 Conteudo:

```
1  /*????????????????????????????????????????????????????????????????????????*/
2  /*????????????????????????????????????????????????????????????????????????*/
3  #include "fila.h"
4
5  /* Inicia uma nova fila. */
6  fila *novaFila(void) {
7      fila *tmp = (fila*) malloc( sizeof(fila) );
8
9      if ( tmp != NULL ) {
10         tmp->cabeca = NULL;
11         tmp->fim = NULL;
12         tmp->n = 0;
13     }
14
15     return tmp;
16 }
17
18 /* Insere uma nova informacao no fim da fila. */
19 int entra ( fila *f, int info ) {
20     if ( f->n == 0 ) {
21         f->fim =  (nof*) malloc(sizeof(nof));
22         f->cabeca = f->fim;
23         f->fim->prox = NULL;
24         f->fim->info = info;
25         f->n = 1;
26
27         return 1;
28     }
29
30     f->fim->prox =  (nof*) malloc(sizeof(nof));
31
```

```
32      if ( f->fim->prox != NULL ) {
33          f->fim = f->fim->prox;
34          f->fim->info = info;
35          f->fim->prox = NULL;
36          f->n++;
37          return 1;
38      }
39      return 0;
40  }
41
42  /* Busca o proximo da fila. */
43  int proximo(fila *f) {
44      int info = f->cabeca->info;
45      nof *tmp = f->cabeca;
46
47      if ( !f || !f->cabeca )
48          return -1;
49
50      f->cabeca = f->cabeca->prox;
51      f->n--;
52      free(tmp);
53
54      return info;
55  }
56
57  /* Libera a fila para a memoria. */
58  int fechou(fila *f) {
59      nof *atu;
60      nof *prox;
61
62      if ( f == NULL )
63          return 0;
64
65      atu = f->cabeca;
66
67      while ( atu != NULL ) {
68          prox = atu->prox;
69          free(atu);
70          atu = prox;
71      }
72      free(f);
73
74      return 1;
75  }
76
77  /* Retorna o tamanho da fila. */
78  int tamanho(fila *f) {
79      if ( f==NULL )
80          return -1;
81      return f->n;
82  }
83  /*????????????????????????????????????????????????????????????????????*/
84  /*????????????????????????????????????????????????????????????????????*/
```

fila.c

2.6 fila.h

Este novo arquivo contém os prototipos das funções contidas em fila.c.

2.6.1 Localização

Diretório: /usr/src/servers/pm/

2.6.2 Conteudo:

```
/*????????????????????????????????????????????????????????????????????????????*/
/*????????????????????????????????????????????????????????????????????????????*/
#include <stdio.h>
#include <stdlib.h>

/* Definindo os nos utilizados na fila. */
typedef struct No {
    int info;           /* Informacao a ser guardada na fila (PIDs). */
    struct No* prox;    /* Ponteiro para o proximo no da fila.       */
} nof;

/* Definindo a estrutura da fila. */
typedef struct Fi {
    nof *cabeca;  /* Inicio da fila.  */
    nof *fim;     /* Fim da fila.     */
    int n;        /* Tamanho da fila. */
} fila;

fila *novaFila();
int entra( fila *f, int info );
int proximo( fila *f );
int fechou( fila *f );
int tamanho( fila *f );
/*????????????????????????????????????????????????????????????????????????????*/
/*????????????????????????????????????????????????????????????????????????????*/
```

fila.h

2.7 misc.h

Este novo arquivo contém o prototipo para a unica função que será usada no *main.c* que esta contida em *misc.c*.

2.7.1 Localização

Diretório: /usr/src/servers/pm/

2.7.2 Conteudo:

```
1  /*????????????????????????????????????????????????*/
2  /*????????????????????????????????????????????????*/
3  /* Prototipo para a funcao terminator().          */
4  void terminator(int ppid);
5  /*????????????????????????????????????????????????*/
6  /*????????????????????????????????????????????????*/
```

misc.h

2.8 misc.c

Este arquivo contém a implementação do servidor de semáforos assim como a estrutura de dados utilizada por ele.

2.8.1 Localização

Diretório: /usr/src/servers/pm/

2.8.2 Alterações [16-24] e [636-848]:

Criando algumas definições para melhorar a visualização e também permitindo alterar facilmente o servidor entre modo de produção e depuração. Aqui também acontece a inclusão do cabeçalho das funções que tratam a fila.

```
16 /*????????????????????????????????????????????????????????????????????????*/
17 /*????????????????????????????????????????????????????????????????????????*/
18 #define MAX_SEM 128      /* Definindo o numero maximo de semaforos.     */
19 #define DB 0             /* Define se esta em modo de depuracao.        */
20 #define DEBUG if (DB)    /* Monta condicao de depuracao.                */
21
22 #include "fila.h"        /* Insercao das funcoes de controle de fila.   */
23 /*????????????????????????????????????????????????????????????????????????*/
24 /*????????????????????????????????????????????????????????????????????????*/
```

misc.c

Aqui esta o código do servidor de semáforos.

```
636 /*????????????????????????????????????????????????*/
637 /*????????????????????????????????????????????????*/
638
639 /*
640  * Definindo a estrutura do semaforo.
641  */
642 typedef struct semaforo {
643     /* quantidade de processos que podem passar pelo semaforo          */
644     int NMEM;           /* Tamanho do semaforo.                        */
645     int N;              /* Posicoes livres do semaforo.                */
646     unsigned int ppid;  /* PID do processo que pediu este semaforo.    */
647     short int espera;   /* Processo pai esperando o termino dos filhos. */
648     fila *f;            /* Fila de acesso ao semaforo.                 */
649 } SEM;
650
651
652 /* Declarando variaveis globais.                                        */
653 static SEM vet_sem[MAX_SEM];        /* Vetor de semaforos.              */
654 static short int qnt_sem = -1;      /* Quantidade de semaforos disponiveis. */
655
656 /*
657  * Funcao utilizada para iniciar os semaforos na primeira vez que o sistema
658  * rodar. Para isso a variavel que contem os semaforos livres, que foi
659  * iniciada com -1 servira de contador para a inicializacao e terminara com
660  * MAX_SEM.
661  */
662 void inicializa_sem(void) {
663     DEBUG printf("\nIniciando os semaforos.\n");
664     for( qnt_sem=0; qnt_sem<MAX_SEM; qnt_sem++) {
665         vet_sem[qnt_sem].ppid = NO_PID;
666         vet_sem[qnt_sem].f = novaFila();
667         vet_sem[qnt_sem].espera = 0;
668     }
669 }
670
671 /*
672  * Funcao que retorna 1 caso parent seja ancestral de ppid.
673  */
674 int ancestral (int ppid, int parent) {
675     int prn = ppid;
676     int mem = NO_PID;
677
678     DEBUG printf("\n");
679     while ( mem!=prn && prn>parent ) {
680         DEBUG printf("name: %s\tpid1: %d\tpid2: %d\tparent: %d\n", mproc[prn].
    mp_name, prn, mproc[prn].mp_pid, mproc[prn].mp_parent );
681         mem = prn;
682         prn = mproc[prn].mp_parent;
683     }
684     if (prn==parent) return 1;
685     return 0;
686 }
687
688 /*
689  * Funcao que retorna uma nova posicao de semaforo valida.
690  * Retorna -1 indicando erro quando o numero de semaforos
691  * maximo e alcancado.
692  */
693 PUBLIC int do_get_sem() {
```

```
694     int i;
695     int n = m_in.m1_i1;      /* Tamanho do semaforo. */
696     int ppid = who_p;        /* Pid do pai.          */
697 
698     /* Verificando se vetor de semaforos precisa ser inicializado. */
699     if ( qnt_sem == -1 ) inicializa_sem();
700 
701     /* nao ha mais semaforos dispon veis */
702     if (qnt_sem==0 || n<=0){
703         DEBUG printf("\nSemaforos indisponiveis, matando o processo.\n");
704         return -1;
705     } else {
706         DEBUG printf("\nProcurando por uma posicao livre no vetor de semaforos.\
    n");
707         for( i=0; i<MAX_SEM; i++) {
708             if ( vet_sem[i].ppid == NO_PID ) {
709                 qnt_sem--;
710                 vet_sem[i].N = n;
711                 vet_sem[i].NMEM = n;
712                 vet_sem[i].ppid = ppid;
713                 DEBUG printf("\nAlocando espaco na posicao %d\n", i);
714                 return i;
715             }
716         }
717         DEBUG printf("\nErro no vetor de semaforos.\n");
718         return -2;
719     }
720 }
721 
722 /*
723  * Funcao que faz um P para o semaforo, caso o processo que faz o pedido nao
    seja descendente
724  * do processo que pediu o semaforo sera -1 indicando erro. Caso tenha acesso
    garantido sera
725  * retornado 1 e caso seja necessario esperar retorna SUSPEND.
726  */
727 PUBLIC int do_p_sem()
728 {
729     int ppid = _ENDPOINT_P(m_in.m_source);        /* Pid de quem chamou a call.
                */
730     int sid = m_in.m1_i1;                         /* Sid para a qual o proceso
    pede acesso. */
731     SEM *sem = vet_sem+sid;                       /* Auxiliar para o caminho do
    semaforo.    */
732 
733     DEBUG printf("\nProcesso %d tentando acessar o semaforo %d.\n", ppid, sid);
734 
735     if ( !ancestral(ppid, sem->ppid) ) {
736         DEBUG printf("\nProcesso %d acessando indevidamente semaforo %d. Matando
     processo. \n", ppid, sid);
737         return -1;
738     }
739 
740     --sem->N;
741     /* se  puder passar */
742     if (sem->N >= 0) {
743         DEBUG printf("\nProcesso %d acessando diretamente o semaforo %d.\n",
    ppid, sid);
744         return 1;
745     }
```

```
746
747     DEBUG printf("\nProcesso %d aguardando o semaforo %d.\n", ppid, sid);
748     entra ( vet_sem[sid].f , ppid );
749     return (SUSPEND);
750 }
751
752 /*
753  * Funcao que faz o P do semaforo. Caso exista alguem esperando na fila de
        acesso acorda este processo.
754  * Se o processo gerador deste semaforo estiver esperando os filhos terminarem e
        o processo que chamou
755  * for o ultimo na fila avisa o processo gerador que pode terminar.
756  * Caso algum processo tente liberar um semaforo que nao e descendente retorna
        -1 indicando erro.
757  */
758 PUBLIC int do_v_sem()
759 {
760     int ppid = _ENDPOINT_P(m_in.m_source);
761     int npid;
762     int sid = m_in.m1_i1;
763     SEM *sem = vet_sem+sid;
764
765     DEBUG printf("\nProcesso %d liberando o semaforo %d.\n", ppid, sid);
766
767     if ( !ancestral(ppid, sem->ppid) ) {
768         DEBUG printf("\nProcesso %d liberando indevidamente semaforo %d. Matando
        processo. \n", ppid, sid);
769         return -1;
770     }
771
772     ++sem->N;
773     /* Acordando o proximo a entrar no semaforo. */
774     if ( tamanho(sem->f) > 0) {
775         npid = proximo(sem->f);
776         DEBUG printf("\nProcesso %d acordando processo %d no semaforo %d.\n",
        ppid, npid, sid);
777         setreply (npid, 1);
778     }
779     /* Liberando o pai caso ele esteja esperando a conclusao dos filhos. */
780     else {
781         if ( sem->N==sem->NMEM && sem->espera ){
782             DEBUG printf("\nProcesso %d avisando pai %d do final da fila do
        semaforo %d.\n", ppid, vet_sem[sid].ppid, sid);
783             setreply(vet_sem[sid].ppid, 1);
784         }
785     }
786
787     return 0;
788 }
789
790 /*
791  * Funcao que libera um semaforo. Pode ser usada diretamente pelo processo
        gerador
792  * como quando o processo gerador morre. Caso um processo que nao tenha gerado
        este
793  * semaforo tente cancela-lo a funcao retorna -1 indicando erro.
794  */
795 int free_sem(int sid, int ppid)
796 {
797     if ( vet_sem[sid].ppid==ppid ) {
```

```
798             vet_sem[sid].ppid = NO_PID;
799             if ( fechou(vet_sem[sid].f) )
800                 vet_sem[sid].f = novaFila();
801             vet_sem[sid].espera = 0;
802             qnt_sem++;
803             return 0;
804         }
805         return -1;
806 }
807 /*
808  * Funcao que trata a call para liberar um semaforo.
809  * Apenas uma interface de acesso para a funcao free_sem().
810  */
811 PUBLIC int do_free_sem()
812 {
813         int sid = m_in.m1_i1;
814         int ppid = who_p;
815 
816         DEBUG printf("\nProcesso %d pede cancelamento do semaforo %d.\n", ppid, sid)
        ;
817 
818         return free_sem(sid, ppid);
819 }
820 
821 /*
822  * Funcao chamada pelo PM para verificar se um processo morto gerou algum
        semaforo.
823  * Se sim libera o semaforo para outros processos.
824  */
825 void terminator(int ppid) {
826         int i;
827         for ( i=0; i<MAX_SEM; i++ ) {
828             if ( vet_sem[i].ppid == ppid ) {
829                 DEBUG printf("\nProcesso %d morreu, cancelando semaforo %d.\n", ppid
        , i);
830                 free_sem( i, ppid);
831             }
832         }
833 }
834 
835 /*
836  * Funcao que permite que o processo gerador do semaforo espere que todos os
        seus descendentes
837  * liberem o acesso para o semaforo.
838  */
839 PUBLIC int do_wait_sem(void) {
840         int ppid = who_p;
841         int sid = m_in.m1_i1;
842         if ( vet_sem[sid].ppid == ppid ) {
843             vet_sem[sid].espera = 1;
844             return (SUSPEND);
845         }
846         return -1;
847 }
848 /*????????????????????????????????????????????????????*/
```

misc.c

2.9 semaforo.h

Este novo arquivo contém o codigo das chamadas para o servidor de semáforos. Qualquer programa que tenha este cabeçalho é capaz de fazer requisições ao servidor de semáforos.

2.9.1 Localização

Diretório: /usr/include/

2.9.2 Conteudo:

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

#include <lib.h>
#include <unistd.h>
#include <sys/cdefs.h>
#include <stdlib.h>
#include <stdio.h>
#include <minix/callnr.h>
#include <minix/endpoint.h>

PUBLIC int get_sem (int n) {
   int sid;
   message m;
   m.m1_i1 = n;

   sid = (_syscall(PM_PROC_NR, GET_SEM, &m) );

   if ( sid==-1 ) exit(0);

   return sid;
}

PUBLIC int p_sem (int sid) {
   message m;
   m.m1_i1 = sid;

   /* Faz a call e pode ser suspenso neste momento. */
   return _syscall(PM_PROC_NR, P_SEM, &m);
}

PUBLIC int v_sem (int sid) {
   int rec;
   message m;
   m.m1_i1 = sid;
   _syscall(PM_PROC_NR, V_SEM, &m);
}

PUBLIC int free_sem (int sid) {
   message m;
   m.m1_i1 = sid;
   return (_syscall(PM_PROC_NR, FREE_SEM, &m) );
}

PUBLIC int wait_sem (int sid) {
   message m;
   m.m1_i1 = sid;
   return (_syscall(PM_PROC_NR, WAIT_SEM, &m) );
}

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

semaforo.h

3 Conclusão

A criação de um sistema de semáforos no minix não e algo conceitualmente complicado, no entanto envolve uma necessidade de grande conhecimento do sistema de mensagens assim como a instalação de uma nova imagem de boot que contenha o servidor.

Para permitir que programas sejam compilados com instrucoes para o uso de semáforos foi necessária a criação de uma nova biblioteca na área */usr/include/*, tal biblioteca monta as chamadas de sistema adequadas para cada uma das tarefas do servidor e também permite que o sistema utilize estas chamadas caso necessário..

A grande maioria das alterações foi feita diretamente no código do *PM*. Sendo que o maior desafio envolvido foi entender como ocorre a comunicação entre processos através das mensagens.

A conclusão final é de que, embora custosa em termos de processamento, uma arquitetura de sistema operacional pode se tornar bastante segura quando baseada em mensagens e protegida em camadas.

Referências

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

[2] Derek Frank, Synchronization

[3] Colin Fowler, src/kernel/proc.c File Reference

[4] Junjie Li, How to Add a System Call in MINIX 3.1.8