Relatório do Projeto 2
MC504 - Grupo 12

Matheus Domingues Casanova Nogueira (185135)
André Rodrigues Alves da Silva (231392)
Daniel de Sousa Cipriano (233228)
Guilherme Gomes Gonçalves (170927)

***Resumo*—Neste relatório de projeto, apresentamos detalhes sobre a implementação e execução de uma chamada de núcleo (*kernel call*) no sistema operacional *Minix* 3, em conjunto com a construção de um serviço invocador de chamadas de sistema e integração à um programa manipulador de estado de processos do sistema operacional.**

Inicialmente, mostramos os passos tomados na implementação dos arquivos de cabeçalho, prototipação e configuração, para execução da chamada de núcleo *kpadmon* e para realização do serviço invocador de chamadas de sistema *spadmon*. Logo após, descrevemos a realização de um teste de funcionamento do serviço, integrado com a chamada de núcleo, implementando o código de uma tarefa básica que realiza a chamada de sistema.

Por fim, concluímos o relatório apresentando a implementação do programa *padmon.c* , que é integrado à chamada de núcleo *kpadmon* e ao serviço *spadmon*, para realização de listagem e manipulação no estado de processos do sistema operacional Minix 3. Descrevemos também, os pontos principais de aprendizado que obtivemos na realização do projeto e detalhamos a contribuição individual de cada membro no desenvolvimento do trabalho.

I. Introdução

CHAMADAS DE SISTEMA, também referidas como chamadas de núcleo, são componentes importantes para o funcionamento e integridade de um sistema operacional. Elas são responsáveis por permitir a comunicação entre programas do usuário e o núcleo do S.O, através de funções especiais que acessam componentes privilegiados do sistema. Quando um programa solicita uma chamada de sistema, é gerado uma interrupção e o controle de execução é passado para o sistema operacional, que irá inspecionar parâmetros, executar a chamada no núcleo e devolver o fluxo à sequência da aplicação.

Como as chamadas de núcleo são frequentemente escritas em assembly, existe alta dependência de hardware específico para manutenção desse recurso. Para resolver essa questão, o sistema operacional possui funções de bibliotecas que permitem realizar chamadas de sistema através de linguagens de programação de alto nível.

O *Minix* 3 utiliza chamadas de sistema definidas do padrão *POSIX* (Portable Operating System Interface), um conjunto de normas definidas pela *IEEE* (Institute of Electrical and Electronics Engineers) para manter a compatibilidade entre sistemas baseados em Unix. As chamadas de sistema no Minix 3 estão definidas no arquivo `system.h` localizado no diretório `/usr/src/minix/kernel`.

II. Criação da Kernell Call *kpadmon*

Nesta primeira etapa do projeto, o principal objetivo é criar uma chamada de sistema personalizada chamada *kpadmon*. Para isso, começamos adicionando o protótipo da função que contém a lógica da *syscall*, chamada *do_kpadmon*, no mesmo arquivo do kernel onde outros protótipos de funções que implementam o comportamento de chamadas de sistema são declarados (`/src/minix/kernel/system.h`).

Uma vez que o protótipo está devidamente declarado, a próxima etapa envolve a implementação dessa função, que foi realizada no diretório `/usr/src/minix/kernel/system/`. Nesta fase inicial, a implementação é bastante simples e consiste apenas em dois comandos `printf` para verificar o funcionamento da chamada de sistema. Após isso, procedimentos mais relacionados ao funcionamento interno do sistema operacional são executados. Isso inclui especificar o arquivo recém-criado no Makefile para que ele seja compilado posteriormente, mapeá-lo com seu novo *alias* para a chamada de sistema no sistema e criar o protótipo da própria chamada de sistema `sys_kpadmon` em `/usr/src/minix/include/minix/syslib.h` e `/usr/include/minix/syslib.h`.

Além disso, o número da chamada de sistema é adicionado em `/usr/src/minix/include/minix/com.h` para garantir que ela possa ser invocada posteriormente. É crucial notar que neste arquivo também é necessário incrementar em uma unidade a variável que controla o número de chamadas de sistema disponíveis, garantindo assim que a nova chamada de sistema possa ser efetivamente invocada.

Assim como fizemos com `do_kpadmon`, também implementamos `sys_kpadmon`, que por sua vez apenas realiza a chamada de sistema para que `do_kpadmon` seja executada. Em seguida, novamente, especificamos esse arquivo no *Makefile* para que ele seja compilado posteriormente.

Por fim, foi realizada a reconstrução do *Kernel* do *Minix* através dos comandos `make includes` e `make hdboot` no diretório `/user/src/releasetools`.

Neste ponto, a chamada de sistema *kpadmon* foi devidamente implementada e está pronta para ser invocada por outros serviços e aplicativos do sistema, como será descrito na próxima seção.

III. Criação do Serviço *spadmon*

Para realizar a chamada da kernel call `kpadmon`, implementada anteriormente, criamos um serviço invocador de chamadas de sistema, nomeado de `spadmon`. Para a implementação desse serviço, primeiramente criamos dentro do diretório `/usr/src/minix/servers` do Minix, a pasta `spadmon`.

Incluímos nesse diretório o arquivo `Makefile`(figura 1), que automatiza a configuração do serviço.

Em seguida, incluímos nesse mesmo diretório, a implementação de código do arquivo `spadmon.c`, responsável por realizar a ligação entre código de usuário e a *kernel call*, através da comunicação por um tipo específico de mensagem definido no código do *Minix* (mais detalhes na seção IV). Essa mensagem é passada por parâmetro para a função `sys_kpadmon`, que como dito anteriormente, irá executar a chamada de sistema necessária.

Após isso, criamos o arquivo `spadmon.conf` (figura 2), dentro do diretório `/usr/src/minix/servers/spadmon`. Esse arquivo estabele configurações de permissão para execução de *kernel calls* e comunicação inter-processos, garantindo o funcionamento correto do serviço `spadmon`.

Por fim, adicionamos a referência ao diretório `/usr/src/servers/spadmon` no arquivo `Makefile` (figura 3) em `/usr/src/minix/servers`. Executamos `make` e `make install` em `/usr/src/minix/servers/spadmon`. Nesse ponto o serviço está configurado e pronto para ser executado. Para execução utilizamos o gerenciador de serviços do *Minix* através do comando `minix-service up /service/spadmon` (figura 4).

IV. Criação de uma tarefa simples integrando *kpadmon* e *spadmon*

Para testarmos e entendermos melhor o funcionamento das *kernel calls* e dos serviços do Minix, criamos uma tarefa simples relacionando a chamada *kpadmon* que criamos com o serviço *spadmon*. A tarefa que criamos, via terminal, lê dois inteiros e chama o serviço *spadmon* passando esses inteiros como parâmetros. O serviço, por sua vez, recebe esses parâmetros e se comunica com a *kernel call kpadmon*, que soma eles e retorna a soma de volta para *spadmon*, que imprime ela. A seguir, descreveremos com mais detalhes as implementações que fizemos para essa tarefa.

Para escrevermos a tarefa que recebe os parâmetros e chama o serviço *spadmon*, criamos um programa em C (`tarefa_3-3.c`) que lia dois inteiros utilizando a função `scanf` e convertia eles em *strings* (`char[]`), para assim poder chamar o serviço. A conversão foi realizada com a função `sprintf`. Para chamarmos o serviço, criamos uma string com o comando que queríamos executar no terminal, que é o comando padrão do Minix para chamar serviços, o `minix-service run /service/spadmon -args "[...]"`. Após concatenarmos essa string do comando com a string dos números lidos (utilizando a função `strcat`), finalmente executamos a string final no *shell* utilizando a função `system`. O código desse programa pode ser visto na figura 7.

O serviço *spadmon* recebe então esses parâmetros (números lidos em `tarefa_3-3.c`) no formato de um vetor de strings (`char ** argv`), e os converte para inteiros utilizando a função `strtol`. A fim de evitarmos erros simples, checamos antes da conversão se o número de argumentos (que é recebido pelo sistema no parâmetro `int argc`) é 3. Apesar de recebermos apenas 2 inteiros, sempre temos um argumento em `argv[0]` contendo o nome do executável que chamou a função (o seu caminho desde o diretório da raiz).

Após isso, o serviço imprime uma mensagem para indicar sucesso na conversão dos números, e logo em seguida passa eles para a *kernel call kpadmon*. Para passarmos esses inteiros, precisamos conhecer um pouco da estrutura de comunicação entre processos do Minix. A forma mais comum de nos comunicarmos com outro processo (nesse caso, com *kpadmon*), é usarmos uma estrutura de mensagens criada pelo próprio S.O. definida como `message`. A definição dessa `struct` pode ser encontrada no arquivo `/usr/src/include/minix/ipc.h`. Ela é uma estrutura contendo uma união de vários tipos de mensagem possíveis, que também foram criados pelo S.O. No total, existem nove formatos de mensagem, `mess_1` a `mess_9`, cada um com seus próprios usos e suas especificidades. Além disso, uma mensagem sempre tem um campo `m_source`, informando quem enviou a mensagem, e um campo `m_type`, indicando o tipo da mensagem. Seguindo a bibliografia [1], do Minix, decidimos utilizar a mensagem `m_m1` para enviarmos os dois inteiros para a chamada *kpadmon*.

Após isso, executamos a chamada `sys_kpadmon`, que executa a função da *kernel call kpadmon*, a função `do_kpadmon`. Nessa função, recebemos os números através mensagem `*m_ptr` (que é a mensagem `m_m1` que enviamos em *spadmon*), e os imprimimos para mostrar que lemos eles com sucesso. Após isso, simplesmente retornamos a soma deles usando a *keyword* padrão `return`. O código desse procedimento do *kpadmon* pode ser visto na figura 5. O serviço *spadmon* então recebe a soma desses números que foi retornada e a imprime no terminal, terminando finalmente o fluxo de execução da tarefa inteira. O código da função modificada do *spadmon* que fizemos para essa seção se encontra na figura 6.

Quando executamos a tarefa criada junto com as modificações do *kpadmon* e do *spadmon* obtemos o resultado algorítmico esperado (as chamadas estavam sendo executadas e estavam somando corretamente as entradas). No entanto, ao final da execução, estávamos recebendo uma mensagem de erro (ou um *warning*) dizendo *"Request 0x700 to RS failed: specified endpoint is not alive (error 215)"*. Ao lermos a documentação do Minix [2] [3], descobrimos que novos serviços se comunicam com um outro servidor do Minix chamado de *Reincarnation Server (RS)*. Esse servidor, além de periodicamente enviar mensagens para os serviços, define protocolos de inicialização para os serviços receberem e processarem. Uma maneira simples de recebermos esse protocolo e administrarmos melhor o serviço é usarmos uma própria componente da biblioteca do sistema operacional chamada de *System Event Framework (SEF)*. Ao usarmos esse *framework* através da chamada `sef_startup()` no começo do serviço *spadmon*, fomos capazes de resolver o erro 215 descrito acima, e conseguimos fazer nossos serviços funcionarem perfeitamente. A figura 8 mostra o resultado final de uma execução da tarefa que criamos nessa seção.

Como nosso repositório do projeto 2 (projeto-2) é apenas um *fork* do Minix, resolvemos criar um novo repositório para guardarmos o programa `tarefa_3-3.c`. Esse novo repositório se chama projeto2-aux, e pode ser encontrado no GitLab do grupo. Os *commits* que fizemos para implementarmos essa tarefa, editando o *kpadmon* e o *spadmon*, podem ser encontrados

no repositório normal projeto-2, e o código da tarefa em C `tarefa_3-3.c` pode ser encontrada nesse novo repositório que criamos.

V. Criação do Programa *padmon*

Para última etapa do projeto, construímos o programa *padmon*, que funciona através do terminal, onde pode executar comandos, como mostrar o estado de todos os processos que atualmente se encontram no sistema e mudar o estado de um processo. O programa recebe os comandos do usuário através da entrada padrão. Ele então analisa os comandos e inicia o serviço spadmon passando para este as ações a serem executadas. *Spadmon* por sua vez, foi implementado de maneira similar ao administrador de processos, sendo responsável por fazer o devido tratamento dos argumentos recebidos, executar alguns comandos e se comunicar com a kernel call *kpadmon* quando necessário. Já *kpadmon* foi utilizado quando os comandos recebidos por *spadmon* precisam ser verbosos, sinalizando as possíveis alterações de estados dos processos.

Padmon suporta os seguintes comandos:

-ps - Exibe o estado de todos os processos.
-r [pid/endpoint] - Altera o estado do processo especificado para R (executável).
-s [pid/endpoint] - Altera o estado do processo especificado para S (dormindo).
-t [pid/endpoint] - Altera o estado do processo especificado para T (detido).
-z [pid/endpoint] - Altera o estado do processo especificado para Z (zumbi).
-e [pid/endpoint] - Encerra o processo especificado.
-v - Exibe mensagens de depuração.
-help - Exibe uma lista de comandos.

Ele captura o que foi digitado pelo usuário e envia os argumentos para *spadmon* usando o comando `minix-service run /service/spadmon -args` e os argumentos a serem enviados.

Spadmon por outro lado, recebe os argumentos, verifica se são válidos e faz os tratamentos necessários para extrair a ação que deve executar. Após esse processo, se o comando digitado pelo usuário for *./padmon -ps*, o serviço irá utilizar a função *getsysinfo* para obter a tabela do gerenciador de processos e imprimir no terminal o PID/Endpoint e o estado de cada processo (figura 9). Já para o comando *./padmon -help* o serviço imprime no terminal um guia de uso:

Uso: spadmon [comando] [argumento]
Comandos disponíveis:
-ps: Mostra a lista de processos
-r: Coloca o processo no estado executável
-s: Coloca o processo no estado dormindo
-t: Coloca o processo no estado detido
-z: Coloca o processo no estado zumbi
-e: Termina o processo
-help: Mostra este help

Por fim, para os comandos de mudança de estado dos processos, *spadmon* utiliza a kernel call *sys_kill* para enviar o sinal necessário para alterar o processo escolhido pelo usuário. Além disso, é enviada uma mensagem para kpadmon através da kernel call *sys_kpadmon* contendo o comando digitado, pid do processo e um indicador de que o comando deve ou não ser verboso.

Exemplos:

```
sys_kill(m.m_m1.m1i1, SIGKILL);

m.m_m1.m1p1 = "r";
m.m_m1.m1p2 = verbose;
m.m_m1.m1i1 = pid;
ret = sys_kpadmon(SYS_KPADMON, SELF, &m);
```

O programa padmon foi feito utilizando a linguagem de programação C, e os *commits* e os códigos finais do *padmon*, *spadmon* e *kpadmon* implementados para essa seção podem ser encontrados no repositório do Gitlab do grupo12. Como mais ilustrações do funcionamento do *padmon*, temos as figuras 10 e 11. Elas mostram a execução do programa para mudar o estado de um processo para executável (10) e encerrado (11). As duas figuras também mostram a flag v para exibir mensagens de depuração.

VI. Conclusão

O trabalho apresentado neste artigo teve como objetivo principal desenvolver uma chamada de sistema personalizada no Minix 3 e integrar essa chamada com um serviço do sistema operacional, bem como criar uma programa a nível de usuário que possa interagir com os dois. Os resultados obtidos mostram que foi possível implementar com sucesso uma chamada de sistema personalizada no Minix 3 e integrá-la com um serviço do sistema operacional. A tarefa simples desenvolvida também funciona corretamente, mostrando que a chamada de sistema kpadmon e o serviço spadmon funcionaram corretamente. Por fim, com este trabalho foi possível ver na prática o que são chamadas de sistema e sua importância para o funcionamento de um sistema operacional, como integrar uma chamada de sistema personalizada com um serviço do sistema operacional, como funciona o gerenciador de processos e como tudo isso pode ser integrado em uma aplicação de usuário.

VII. Divisão do trabalho

- A equipe inteira colaborou com discussões gerais sobre o andamento do projeto e erros que encontrávamos pelo caminho
- Matheus: criação da chamada de sistema *kpadmon* e seções I e II do relatório.
- Daniel: criação do serviço *spadmon* e seção III do relatório.
- André: criação da tarefa simples integrando *kpadmon* e *spadmon* e seção IV do relatório.
- Guilherme: criação do programa *padmon*, alterações em *spadmon* e *kpadmon* e seção V do relatório.
- Mais detalhes podem ser encontrados nos *commits* do Gitlab do grupo12.

APÊNDICE A
FIGURAS

Figura 1: Implementação do arquivo `Makefile` que automatiza a configuração do serviço `spadmon`.

Figura 2: Implementação do arquivo `spadmon.conf` que configura o serviço `spadmon.conf`.

Figura 3: Adição do serviço `spadmon` ao arquivo `Makefile` do diretório `/usr/src/minix/servers`.

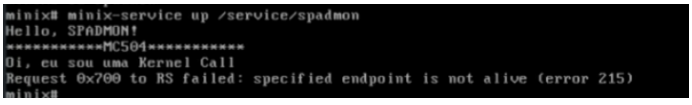

Figura 4: Execução do serviço spadmon

```c
#include "kernel/system.h"
#include <minix/endpoint.h>
#include <stdio.h>

int do_kpadmon(struct proc *caller_ptr, message *m_ptr)
{
    // printf("**********MC504**********\n");
    // printf("Oi, eu sou uma Kernel Call\n");

    // implementacao de uma tarefa aqui
    int a = m_ptr->m_m1.m1i1;
    int b = m_ptr->m_m1.m1i2;
    printf("KPADMON: Somar os parametros recebidos pela mensagem: %d e %d\n", a, b);

    int soma = a + b;
    return soma;
}
```

Figura 5: Implementação da função `do_kpadmon`, do *kpadmon* realizada para a seção 3.3.

```c
#include <stdio.h>
#include <stdlib.h>
#include <minix/syslib.h>

int main(int argc, char **argv){
        if (argc != 3) {
            // Checar se os argumentos estao corretos
            return EXIT_FAILURE;
        }

        int a = strtol(argv[1], &argv[1], 10);
        int b = strtol(argv[2], &argv[2], 10);

        printf("SPADMON: Parametros a serem somados foram recebidos: %d e %d\n", a, b);

        message m; // usado para se comunicar com a kernel call
                   // envia e recebe valores diversos
        // Passar mensagem mess_1
        m.m_m1.m1i1 = a;
        m.m_m1.m1i2 = b;

        // printf("Hello, SPADMON!\n");

        int soma = sys_kpadmon(SYS_KPADMON,SELF,&m);
        printf("SPADMON: A soma calculada por KPADMON foi %d\n", soma);

        return EXIT_SUCCESS;
}
```

Figura 6: Implementação do serviço *spadmon* realizada para a seção 3.3.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <minix/syslib.h>

int main() {
  int a, b;
  scanf("%d %d", &a, &b);

  // Converter para string
  char a_st[12];
  char b_st[12];
  sprintf(a_st, "\"%d ", a);
  sprintf(b_st, "%d\"", b);

  // Comando para executar o servico
  char command[] = "minix-service run /service/spadmon -args ";

  // String final com o comando e os parametros convertidos
  char final[80]; strcpy(final, command);
  strcat(final, a_st); strcat(final, b_st);

  system(final);

  return 0;
}
```

Figura 7: Código da tarefa simples da seção 3.3.

```
minix# ./tarefa_3-3
153 245
SPADMON: Parametros a serem somados foram recebidos: 153 e 245
KPADMON: Somar os parametros recebidos pela mensagem: 153 e 245
SPADMON: A soma calculada por KPADMON foi 398
minix# _
```

Figura 8: Exemplo de execução da tarefa simples da seção 3.3.

```
----------------------
PID/Endpoint State
----------------------
005 R
007 R
004 R
008 R
006 R
009 R
003 R
010 R
011 R
012 R
001 R
018 R
020 R
----------------------
```

Figura 9: Exemplo de execução do comando -ps do `padmon`.

```
minix# ./padmon -vr 020
KPADMON: Received command "r". Executing r command...
KPADMON: Alterando estado do processo 020
 para executavel.minix#
```

Figura 10: Exemplo de execução do comando -vr do `padmon`.

```
minix# ./padmon -ve 020
KPADMON: Received command "e". Executing e command...
KPADMON: Encerrando o processo 020
.minix#
```

Figura 11: Exemplo de execução do comando -ve do `padmon`.

REFERÊNCIAS

[1] Tanenbaum, Andrew S.; Woodhull, Albert S. (1987). Operating Systems: Design and Implementation. ISBN 9780131429383.

[2] Minix 3 developers guide, url = https://wiki.minix3.org/doku.php?id=developersguide:start

[3] Minix 3 developers guide - driver programming, url = https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming