Administração de processos em Minix
Relatório do Projeto 2

Diogo T. Sant'Anna, *RA 169966,* Luciano G. Zago, *RA 182835,*
e Maria A. Paulino, *RA 183465*

Resumo—Este projeto teve por objetivo promover a familiarização com os modos de operação do sistema operacional e o funcionamento dos processos dentro desse. Para tal, foi requerido projetar e implementar uma solução de software que seja capaz de administrar os estados dos processos de um sistema operacional. A aplicação deveria consistir de duas partes: a chamada padmon - nível usuário - e spadmon - nível kernel. A execução do trabalho exigiu conhecimentos como a utilização de chamadas de sistema, passagem de parâmetros para essas e criação de servidor e syscalls. Por fim, foi possível desenvolver uma solução capaz de executar todas as funções exigidas no enunciado deste projeto.

1 Introdução

As soluções implementada durante este projeto foram o `padmon` e `spadmon`.

O `padmon` deve gerenciar as opções exigidas no enunciado e foi idealizado como sendo o programa responsável por intermediar a comunicação entre o usuário e o serviço `spadmon` responsável por realizar os serviços solicitados pelo usuário. Sendo assim, o `padmon` é responsável por:

1) Interpretar o requisitado pelo usuário, identificando possíveis erros e reportando-os diretamente.
2) Devidamente redirecionar as requisições válidas para o serviço `spadmon` através de syscalls.
3) Identificar os retornos das syscalls, redirecionando os resultados ao usuário, possivelmente descrevendo os erros decorridos.

Já o `spadmon`, consistia na criação de um server e syscalls que podem ser chamadas no programa do usuário citado anteriormente, e tiveram toda sua estrutura embasada no apresentado em [2] e [9] com pequenas alterações que podem ser observadas ao decorrer do desenvolvimento deste relatório.

2 Desenvolvimento

Como dito no enunciado deste projeto, a aplicação foi desenvolvida em duas partes, assim como explicitas a seguir.

2.1 Padmon: Aplicação Nível Usuário

2.1.1 Projeto e implementação

Os programas de usuário são colocados no diretório `usr.bin`, logo possui o arquivo base da implementação `/usr.bin/padmon/padmon.c`. Além disso, este diretório se transforma em `/usr/bin/`, local padrão que permite a execução dos programas pelas chamadas da linha de comando, bastando chamar o programa diretamente pelo nome.

Observamos também que é importante que o próprio programa `padmon` reporte os resultados ao usuário para que este tenha a liberdade de direcionar as saídas para onde desejar, por exemplo.

2.1.2 Leitura de Argumentos

Para processamento dos argumentos recebidos, utilizou-se da função `getopt()`, proveniente do header file `unistd.h`[1] , para o tratamento de opções curtas. Esta função recebe como parâmetro um inteiro com a quantidade de argumentos, um um vetor de strings com os argumentos e uma string com a definição das opções aceitas pelo programa e seus respectivos formatos (flags ou não) [1]. Seu retorno é: -1 quando todas as opções foram conferidas, o carácter correspondente à opção desejada ou '?' quando a opção selecionada é inválida.

Como `getopt()` é usado somente para comando simples, foi necessário verificar os argumentos recebidos em busca de instruções `-ps` e `-help` e simplificá-los a um único carácter. Além disso, para evitar problemas com a verbosidade, junto com a simplificação de argumentos citada acima, verificou-se a presença do carácter. Quando encontrado, este é retirado da string e uma flag que indica o uso de verbosidade é setada.

2.1.3 Chamadas de sistema

O `padmon` se comunica com o `spadmon` através de 6 diferentes syscalls: uma delas solicitando a listagem dos processos no sistema (`padmon -ps`) e as outras cinco correspondendo à solicitação para mudar o estado de um dado processo. As syscalls foram criadas no serviço do `spadmon` e recebem parâmetros por meio das messages, estruturas usadas pelo Minix para passagem de dados entre processos.

2.1.4 Passagem de parâmetros (message)

A messages do Minix correspondem a uma union de structs que definem messages contendo componentes particulares para serem usados em diferentes contextos dentro do Minix. Para a implementação das syscalls utilizamos duas structs diferentes, uma simples já implementada no Minix (mess_1) e outra implementada por nós para melhor intercambiar os resultados do `padmon -ps`.

1. http://pubs.opengroup.org/onlinepubs/7908799/xsh/unistd.h.html

Para a syscall correspondente ao `padmon -ps` utilizamos a message implementada como a seguir:

```
typedef struct {
    vir_bytes pid_array;
    vir_bytes status_array;
    int length;
    int fun_return;
    uint8_t padding[40];
} mess_spadmon_processes;
```

Onde em `pid_array` e `status_array` o `padmon` insere o endereço para um array que o `spadmon` popula com os PIDs e status, respectivamente, dos processos encontrados no sistema operacional. O campo `length` é preenchido pelo `spadmon` com o número total de processos encontrados, e o campo `fun_return` é utilizado para passar os retornos das syscalls. Para as outras syscalls, foi usado uma struct de mensagem que possui 3 campos para inteiros, dos quais um foi usado para passar o PID do processo de que se deseja alterar o estado.

2.1.5 Tratamento de erros: Retornos padrões

Como já mencionado, o `padmon` é responsável por devidamente chamar as syscall, interpretar seus retornos e direcionar o resultado ao usuário. Para isso convencionamos inteiros para representar e indicar possíveis erros nas syscalls, que portanto retornam um inteiro indicando como a chamada finalizou. Os números convencionados e seus respectivos significados são:

Código	Significado
0	Syscall terminou sem nenhum erro
1	Não foi possível obter alguma das tabelas de processos
2	Foi solicitado a mudança de estado para o estado em que o processo já se encontra.
400	Houve uma solicitação inválida
404	Não foi encontrado o processo com o PID dado
500	Aconteceu algum erro imprevisto

Tabela 1: Tabela de Retornos

2.2 Spadmon: aplicação nível kernel

2.2.1 Serviço do Minix

Para implementar um novo serviço no Minix, foi seguido um tutorial[2] à risca. O server `spadmon` localiza-se na pasta `/minix/servers/spadmon`. Ele se baseou na estrutura do server `ds`, para receber e enviar as messages das syscalls. Após a remoção de funções desnecessárias, o protótipo base do server `spadmon` foi criado. Foi necessário incluí-lo no Makefile da pasta `/minix/servers/`.

Após isso, para adicionar o server no Minix, foi incluído no arquivo `/minix/include/com.h` a linha `#define SPADMON_PROC_NR ((endpoint_t) 11)`, antes de `LAST_SPECIAL_PROC_NR`, e alterado seu valor de 11 para 12.

Para adicionar o server ao kernel, foi necessário incluir no arquivo `/kernel/table.c` a linha `{SPADMON_PROC_NR, "spadmon"}`, no final da lista. E em `/servers/rs/table.c` adicionada abaixo de `PFS_PROC_NR` nas duas primeiras listas desse arquivo.

Em `/etc/system.conf`, para darmos permissão ao nosso server, foi incluído o seguinte bloco:

```
service spadmon{
    uid 0;
    ipc ALL_SYS;
    system  ALL:
    vm  BASIC;
    io  NONE;
    irq NONE;
    sigmgr    rs;
    scheduler KERNEL;
    priority  4;
    quantum   500;
};
```

Para compilar o `spadmon`, foi necessário incluir no Makefile de `/releasetools/`, na lista de programas, após o `init`, a seguinte linha `PROGRAMS+= ${PROGROOT}/servers/spadmon/spadmon`. Para o `padmon` ter permissão para enviar uma syscall ao `spadmon`, foi necessário remover em `/kernel/proc.c` o bloco de código referente à `if (call_nr != RECEIVE)`

2.2.2 Criação das syscalls

Para implementar as syscalls do `spadmon`, foi seguido um tutorial[3], porém foi necessário adaptá-lo para o nosso server, já que ele se referia à criação de syscalls para o `pm`. O `/minix/server/spadmon/table.c` mapeia os números das syscalls para os ponteiros das funções criadas em `/minix/server/spadmon/spadmon.c`, através de um vetor `call_vec` e as constantes. Exemplo: `int (*call_vec[])(void) = {do_spadmon_ps, do_spadmon_r};`

Os números das syscalls foram escolhidos em `/minix/include/minix/callnr.h`, criando uma syscall para cada mudança de estado do `padmon` e para o `-ps`, seguindo o seguinte modelo: `#define SPADMON_PS (SPADMON_BASE + 0)`, com `#define SPADMON_BASE 0x1B00`.

Em `/minix/servers/spadmon/proto.h` foi definido os protótipos das funções utilizadas no `main.c`, `spadmon.c` e `table.c`. Todos esses arquivos foram incluídos no Makefile de `/minix/servers/spadmon/`.

As novas syscalls podem ser chamadas no programa de usuário (`padmon`) através do seguinte comando `_syscall(SPADMON_PROC_NR, modo, &m)`, em que `modo` deve ser substituído pelas constantes definidas em `callnr.h` e `m` deve ser

instanciado como uma das messages descritas anteriormente, que possibilitam a passagem de parâmetros para a syscall, como os argumentos do `padmon` ou retornos do `spadmon`.

2.2.3 Coleta de informações dos processos

Para o processamento do `padmon -ps`, precisamos obter as informações da tabela de processos que, no Minix, é dividida em três partes: uma parte é definida e atualizada no kernel, outra no serviço PM e outra no VFS. As três tabelas são implementadas como arrays de structs(`struct mproc` no `PM`, `fproc` no `VFS` e `proc` no kernel) e as do PM e do kernel têm atributos relevantes para determinar os estados de processo solicitados nesse projeto portanto precisamos de acesso às tabelas. Esse acesso foi garantido pelo uso de uma syscall do Minix chamada `getsysinfo` que permite solicitar dados ao sistema. Utilizamos essa syscall para solicitar uma cópia de cada uma das tabelas necessárias.

Tendo acesso aos processos do sistema, o `spadmon` itera sobre a tabela de processos do PM e, através da leitura de cada `struct mproc`, avalia se o slot do array está em uso e, em caso afirmativo, avalia o estado do processo e insere o `pid` do processo e seu estado em arrays auxiliares.

Ao final da iteração pelo array de `mprocs`, o `spadmon` deve escrever o que foi salvo nos arrays auxiliares nos endereços de memória passados pelo `padmon` através das mensagens, como detalhado anteriormente. Um problema é que o `spadmon` não tem permissão para escrever num endereço de memória reservado para outro processo, então para isso precisamos chamar outra syscall que cuida dessa escrita. A syscall em questão é a `sys_datacopy`, que recebe como parâmetros o que é para ser escrito, onde é para ser escrito e os dois processos envolvidos (no caso o `spadmon` e o `padmon`).

2.2.4 Mudança de estado dos processos

Para mudar os estados dos processo já não é suficiente uma cópia da tabela de processos, precisamos alterar a tabela de fato. Para tal, criamos uma nova syscall dentro do serviço `PM`. O processo de criação foi equivalente a criação das syscalls do `spadmon` já descritas:

1) Adicionamos o número da syscall em `minix/include/minix/callnr.h`;
2) Adicionamos o arquivo no qual definimos a syscall (`spadmon.c`) no makefile do `pm` `minix/servers/pm/makefile`;
3) Adicionamos a assinatura da função em `minix/servers/pm/proto.h`;
4) Relacionamos o número da syscall com a sua função em `minix/servers/pm/table.c`
5) Desenvolvemos a função de fato no arquivo `spadmon.c`

Para a chamada dessa syscall utilizamos a mesma mensagem utilizada na chamada do `spadmon (mess_1)`, mas desta vez utilizamos um inteiro para armazenar o PID do processo que desejamos mudar o estado, e outro inteiro para armazenar o estado para o qual queremos mudar o processo.

Dentro da syscall no PM a primeira coisa que fazemos é iterar sobre o array de `mprocs` buscando o processo de PID dado. Se o processo não for encontrado a syscall já retorna um inteiro indicando que não foi encontrado nenhum processo com PID passado.

Se a função encontra o processo desejado, ela então deve verificar o estado do processo. Para isso utilizamos um procedimento semelhante ao descrito para o processamento do `padmon -ps`: por estarmos no PM já temos acesso direto a sua tabela de processos, então requisitamos uma cópia da tabela do kernel e usamos as informações das duas para identificar o estado do processo.

Para o caso de o processo já estar no estado para o qual a mudança foi solicitada, a syscall retorna um valor indicando esse erro. Caso contrário, o serviço deve mudar o processo para o estado solicitado.

Alterar o valor dos estados traz outro problema, pois não é possível alterar a tabela de processos do kernel com cópia da mesma. Para resolver o problema criamos uma kernelcall para de fato alterar os valores da tabela. Descreveremos a seguir a criação da kernelcall.

2.2.5 Criação da Kernel Call

Seguindo um tutorial [9], foi possível criar uma kernel call para alterar a tabela proc do kernel, que contém informações sobre os estados dos processos, com flags e uma fila de "runeables". Foi necessário criar uma função que realiza a mudança das flags e da fila. Caso as `flags == 0`, o programa é runeble e está na fila. Se não, o oposto.

O protótipo dessa função deve ser adicionado em `/minix/kernel/system.h` e a função dentro de `/kernel/system/`, com a inserção de seu nome no `/kernel/system/Makefile.inc`.

O mapeamento da kernal call é feito em `/minix/kernel/system.c` , incluindo a seguinte linha: `map(SYS_SPADMON_RUN_SWAP, do_spadmon_run_swapl);` O protótipo de `SYS_SPADMON_RUN_SWAP` é inserido em `/minix/include/minix/syslib.h`.

O número da chamada é incluído em `/minix/include/minix/com.h` com `# define SYS_SPADMON_RUN_SWAP (KERNEL_CALL + 58)` e incrementando o valor de `NR_SYS_CALLS`.

A implementação da função que chama a kernel call foi feita em `/minix/lib/libsys/sys_spadmon_run_swap.c`, passando os parâmetros em uma message e chamando a kernel call por `_kernel_call(SYS_SPADMON_RUN_SWAP, &m)`. Coloque também o novo arquivo no Makefile da pasta.

2.3 Testes de Validação

Para auxiliar na validação do implementado, criou-se os testes:

1) Ao utilizar-se `padmon -ps` antes e depois de executar `vi &` e verificar se o número de processos é incrementado;
2) Criou-se um processo e obteve-se seu PID, usando `padmon` alterou-se o estado deste processo para todos os estados disponíveis e verificou-se a mudança efetiva de status, por fim finalizou-se o processo com `padmon -e` e verificou-se se este deixou a lista de processos;
3) Verificou-se o uso da verbosidade comparando a saída de todas as funções fornecidas com e sem a flag `v` setada;
4) Verificou-se se `padmon -v -t $pid`, `padmon -vt $pid` e `padmon -t $pid -v` possuem o mesmo retorno;
5) Confirma se `padmon` chama a syscall do`spadmon`;
6) Verifica se `padmon -help` retorna algo.

Além disso, criou-se testes que esperam erro, como opções e argumentos inválidos.

3 Conclusão

O projeto foi concluido implementando todas as funcionalidades exigidas pelo enunciado. Desde a listagem até a mudança de estado dos processos. Essas tarefas foram realizadas através da criação de um server (spadmon), uma syscall do PM e uma kernelcall. Ainda ocorrem erros inesperados algum tempo depois de alterar os estados dos processos (como vfs error e kernel panic) pois somente a mudança das flags afetam o funcionamento do sistema operacional. Configuramos também o CI para rodar os testes criados pelo grupo.

Referências

[1] https://daemoniolabs.wordpress.com/2011/10/07/usando-com-as-funcoes-getopt-e-getopt_long-em-c/
[2] https://blog.heabuh.com/jekyll/update/2016/11/13/minix-service-tutorial.html
[3] http://homepages.cs.ncl.ac.uk/nick.cook/csc2025/minix/syscall-exercise1.html
[4] http://wiki.minix3.org/doku.php?id=developersguide:kernelapi#sys_getinfo
[5] https://blog.heabuh.com/jekyll/update/2016/11/29/minix-getsysinfo.html
[6] www.pcs.usp.br/~jkinoshi/2007/r2-6.doc
[7] http://www.cis.syr.edu/~wedu/seed/Documentation/Minix2/Process_Tables.pdf
[8] http://condor.depaul.edu/glancast/443class/docs/lecOct05.html
[9] http://wiki.minix3.org/doku.php?id=developersguide:newkernelcall