**EXPERIMENTAÇÃO COM O KERNEL 3.16.0 DO LINUX**

**Introdução**

Este experimento descreve os passos realizados na elaboração de um programa para instrumentalizar funções do Kernel 3.16.0 do sistema operacional Linux. Mais especificamente, o programa contabiliza quantas vezes uma determinada função do código do Kernel é utilizada por um processo (aplicação do sistema ou usuário). Para isto, o programa receberá como parâmetros: a identificação da aplicação (*Process Identification* – PID), a identificação de uma determinada função (representada por um índice de 0 à 22) e um valor 0 ou 1, para zerar ou ler o valor do contador, respectivamente.

O primeiro passo foi identificar quais funções do Kernel seriam instrumentadas. A Tabela 1 apresenta as 23 funções escolhidas.

|  |  |  |
| --- | --- | --- |
| **Função Instrumentada** | **Localização** | **Índice** |
| tcp\_sendmsg() | net/ipv4/tcp.c | 0 |
| kbd\_event() | drivers/tty/vt/keyboard.c | 1 |
| do\_fork() | kernel/fork.c | 2 |
| set\_page\_dirty() | mm/page-writeback.c | 3 |
| do\_path\_lookup() | fs/namei.c | 4 |
| \_\_iget() | fs/inode.c | 5 |
| \_\_ide\_do\_rw\_disk() | drivers/ide/ide-disk.c | 6 |
| \_\_alloc\_pages\_high\_priority() | mm/page\_alloc.c | 7 |
| \_\_free\_pages() | mm/page\_alloc.c | 8 |
| request\_dma() | kernel/dma.c | 9 |
| free\_dma() | kernel/dma.c | 10 |
| sock\_sendmsg() | net/socket.c | 11 |
| kernel\_sendmsg() | net/socket.c | 12 |
| tcp\_read\_sock() | net/ipv4/tcp.c | 13 |
| tcp\_recvmsg() | net/ipv4/tcp.c | 14 |
| tcp\_init\_sock() | net/ipv4/tcp.c | 15 |
| balance\_dirty\_pages() | mm/page-writeback.c | 16 |
| \_\_bdi\_update\_bandwidth() | mm/page-writeback.c | 17 |
| add\_wait\_queue() | kernel/sched/wait.c | 18 |
| msleep() | kernel/timer.c | 19 |
| add\_timer\_on() | kernel/timer.c | 20 |
| put\_pid() | kernel/pid.c | 21 |
| eth\_header() | net/ethernet/eth.c | 22 |

**Alterações no Kernel**

Primeiramente, foi inserido na estrutura de cada processo do sistema operacional um *buffer* de memória com 23 posições, correspondentes a cada uma das funções escolhidas. Para isto foi inserido um array de 23 posições dentro da estrutura *struct task\_struct* no arquivo *include/linux/sched.h*, como apresentado no seguinte trecho de código:

*struct task\_struct*

*{*

*volatile long state; /\* -1 unrunnable, 0 runnable, >0 stopped \*/*

*void \*stack;*

*atomic\_t usage;*

*unsigned int flags; /\* per process flags, defined below \*/*

*unsigned int ptrace;*

*/\* Buffer de 23 posições para os contadores das funcoes instrumentadas \*/*

*unsigned long contador[23]; ///*

*#ifdef CONFIG\_SMP*

***.***

***.***

***.***

Em seguida, foram inseridas as linhas de código abaixo em cada uma das funções instrumentadas, para o incremento do contador. Neste exemplo, o índice 0 do array significa que este código foi inserido na função *tcp\_sendmsg()*.

*struct task\_struct \*tarefa = current;*

*tarefa->contador[0]++; /\* incrementa o contador da funcao tcp\_sendmsg() (indice 0) \*/*

Para que o programa pudesse interagir com as funções do Kernel foi necessária a criação de uma *system call*. O processo é simples e consiste de passos bem definidos:

* Primeiro, como usuário *root*, extraiu-se o código fonte do Kernel no diretório */usr/src/* usando o seguinte comando:

*tar -xvf linux-3.16.tar.xz -C/usr/src/*

* Criou-se um diretório “*scontador*” no diretório fonte do Kernel que foi modificado, localizado em */usr/src/linux-3.16/*.

*mkdir scontador*

* Dentro do diretório *scontador*, com o nome *scontador.c* foi escrito o seguinte código fonte:

*#include <linux/kernel.h>*

*#include <linux/sched.h>*

*asmlinkage long sys\_scontador(pid\_t pid, int indice, int condicao)*

*{*

*struct task\_struct \*tarefa;*

*tarefa = find\_task\_by\_pid(pid);*

*if (condicao)*

*return tarefa->contador[indice]; /\* ler o contador \*/*

*else*

*{*

*tarefa->contador[indice] = 0; /\* zera o contador \*/*

*return tarefa->contador[indice];*

*}*

*}*

* Além do arquivo *scontador.c* foi necessário criar um Makefile, de nome “*Makefile*”, com a seguinte linha:

*obj-y := scontador.o*

Assegurando que quando o Kernel fosse compilado a nossa *system call* fosse incluida.

* Voltando para o diretório fonte do Kernel que foi modificado (*/usr/src/linux-3.16/*) e acessando o arquivo, já existente, de nome Makefile, foi necessário mudar na linha 844 a instrução “*core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/*” para

“*core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ scontador/*”

Essa modificação referencia para o compilador que os arquivos fontes de nossa *system call* estão presentes no diretório scontador.

* Foi necessário adicionar a nova *system call scontador* na tabela de system calls (*syscall\_64.tbl*) contida no diretório */usr/src/linux-3.16/arch/x86/syscalls/*.

*syscall\_64.tbl* para sistemas de 64 bits e *syscall\_32.tbl* para sistemas de 32 bits.

No arquivo *syscall\_64.tbl* na linha 326 foi adicionada a seguinte linha:

“*317 64 scontador sys\_scontador*”

Onde 317 é o número da nossa *system call* *scontador*, note que a system call anterior a nossa é a de número 316, esta ordem deve ser satisfeita.

* Em seguida, adicionamos a nova *system call* no arquivo de cabeçalhos de system calls, *syscalls.h*, no diretório */usr/src/linux-3.16*/*include/linux/*.

Adicionamos a seguinte linha no final do arquivo, porém, antes do *#endif*.

*asmlinkage long sys\_scontador(pid\_t pid, int indice, int condicao);*

Definindo o protótipo na nossa *system call*.

**Compilar o Kernel**

Para compilar o Kernel foi necessário instalar:

1. A última versão do GCC.

2. Pacote de desenvolvimento NCURSES.

3. Os pacotes do sistema devem estar atualizados.

Comandos como *root*:

1. *apt-get install gcc*

2. *apt-get install libncurses5-dev*

3. *apt-get update*

4. *apt-get upgrade*

Para configurar o Kernel foram necessários os seguintes comandos:

*cd /usr/src/linux-3.16/*

*cat /boot/config-<versão do kernel instalado na máquina> > .config*

Após a execução do segundo comando várias opções apareceram no terminal, 'n' para todas as opções.

Em seguida, compilamos o kernel com o comando:

*make*

Após a compilação foi necessário instalar o kernel com o seguinte comando:

*make modules\_install install*

O comando acima instala o Kernel do Linux 3.16 no sistema. Ele cria alguns arquivos dentro do diretório */boot/* e automatimente configura o *grub.cfg*. Logo, os arquivos:

1. System,map-3.16

2. vmlinuz-3.16

3. initrd.img-3.16

4. config-3.16

Foram criados no diretório */boot/* após o comando *make modules\_install install*.

E por fim, o comando *shutdown -r now* foi dado para o reboot do sistema com o kernel modificado (pode ser necessário escolher, no grub (opções avançadas), entre o kernel antigo e o modificado).

**Teste da System Call**

Após a instalação e o reboot no Kernel modificado foi necessário testar a *system call* através de um programa:

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <linux/kernel.h>*

*#include <sys/syscall.h>*

*#include <unistd.h>*

*#include <errno.h>*

*int main(int argc, char \*argv[])*

*{*

*int pid, indice, condicao;*

*pid = atoi(argv[1]);*

*indice = atoi(argv[2]);*

*condicao = atoi(argv[3]);*

*if (condicao)*

*{*

*long int rest\_sys = syscall(317, pid, indice, condicao);*

*printf("Qtd = %ld\n", ret\_sys);*

*}*

*else*

*{*

*long int rest\_sys = syscall(317, pid, indice, condicao);*

*printf("Contador da funcao %d zerado (%ld)\n", indice, ret\_sys);*

*}*

*return 0;*

*}*

A utilização do programa é bem simples. Basta passar como parâmetros o PID de um processo do sistema, o indice da função que deseja-se verificar (de acordo com a Tabela 1) e um parâmetro para indicar se o programa deve ler o contador ou zerá-lo. Foi definido que este parâmetro deve ser 0 para zerar o contador e diferente de 0 para ler o contador. Dessa forma, a utilização do programa possui a seguinte sintaxe:

*#./nome\_do\_programa [PID] [INDICE] [CONDICAO]*

**Resultados**

No primeiro exemplo ilustrado abaixo, temos a verificação de quantas vezes o processo *init* (PID 1) utilizou a função do Kernel *do\_fork()* (índice 2). Em seguida, foi zerado o contador desta função dentro do processo *init*.

*debian@debian:/home/debian/experimento-so# ./a.out 1 2 1*

*Qtd = 72*

*debian@debian:/home/debian/experimento-so# ./a.out 1 2 0*

*Contador da funcao 2 zerado (0)*

*debian@debian:/home/debian/experimento-so# ./a.out 1 2 1*

*Qtd = 0*

O próximo exemplo ilustra quantas vezes o interpretador de comandos *bash* utilizou a função *do\_fork()*. Nota-se que, após zerado o contador desta função, o resultado é 1 porque já está contabilizando a utilização do programa *a.out*.

*debian@debian:**/home/debian/experimento-so# ps*

*PID TTY TIME CMD*

*642 tty1 00:00:00 login*

*717 tty1 00:00:00 su*

*718 tty1 00:00:00 bash*

*893 tty1 00:00:00 ps*

*debian@debian:/home/debian/experimento-so# ./a.out 2917 2 1*

*Qtd = 96*

*debian@debian:/home/debian/experimento-so# ./a.out 2917 2 0*

*Contador da funcao 2 zerado (0)*

*debian@debian:/home/debian/experimento-so# ./a.out 2917 2 1*

*Qtd = 1*