MAC0422 - Sistemas Operacionais
Bacharelado em Ciência da Computação
IME/USP – Segundo Semestre de 2011
Segundo Exercício-Programa

1 Gravador de system calls

1.1 Introdução

Quando um processo do usuário faz uma chamada ao sistema para requisitar um serviço ao SO ou um dispositivo gera uma interrupção, o hardware muda de modo usuário para modo kernel para que o kernel trate a *system call* ou interrupção apropriadamente e em segurança. No x86, este chaveamento é feito por um mecanismo de hardware chamado de "interrupção" ou "trap". Uma interrupção pausa a execução do processo de usuário atual, salva o contexto, muda de modo usuário para modo kernel, e executa um pedaço de código do kernel chamado de "tratador de interrupção".

No xv6, cada tratador de interrupção constrói uma estrutura de dados chamada de *trapframe* que contém os registradores do processador na hora da interrupção. Subsequentemente, a função em C `trap` definida em `trap.c` é chamada. Esta função verifica o número do trap na estrutura *trapframe* para decidir a causa da chamada e o que precisa ser feito. Se o número de trap for `T_SYSCALL`, `trap` chama o tratador de *syscalls* do SO.

Cada *system call* possui um número ou identificador distinto. No xv6, este número é armazenado no registrador `%eax` antes da chamada. A função `syscall` lê este número que fica armazenado no *trapframe*, faz uma busca na tabela de syscalls e decide qual função `sys_*` chamar. O xv6 não copia os argumentos de uma system call para a pilha de kernel. Na verdade, ele carrega estes argumentos diretamente da pilha do usuário usando as funções auxiliares: `argint`, `argptr`, e `argstr`. Estas funções lêem o registrador de pilha `%esp` do *trapframe*, e assim localizam os argumentos desejados.

1.2 Rastreamento

Para familiarizar-se com este processo, leia os arquivos `vector.S, trapasm.S, trap.c`, e `syscall.c` para entender como o xv6 despacha interrupções e system calls. Em particular, marque um *breakpoint* na função `alltraps` e veja como o xv6, por exemplo, trata as interrupções do relógio. Como sugestão, escreva um programa simples que repetidamente chama `getpid()` e use o **gdb** para acompanhar as chamadas ao sistema.

2 Implementação

2.1 Parte I (70%)

Alguns dos benefícios do rastreamento de system calls é poder entender o funcionamento de programas e facilitar a sua depuração. Neste EP, você modificará o kernel do xv6 para gravar as chamadas de sistema que um processo realiza. Para cada *system call*, você deverá gravar o número da chamada, o valor de retorno, e argumentos, se existirem. Existem três tipos de argumentos para syscalls no xv6:

- **Argumento inteiro**, que você deverá gravar o seu valor numérico;
- **Argumento do tipo ponteiro**, que você deverá gravar o valor do endereço e não o valor apontado (e.g. dado um ponteito `p`, grave `p` e não `*p`), e
- **Argumento string**, que você deverá gravar até 20 caracteres somente.

Para controlar a gravação e recuperar seus resultados, você deverá implementar três seguintes system calls:

- `int startrecording()`
- `int stoprecording()`
- `fetchrecords(struct record *records, int num_records)`

Cada processo poderá estar no modo de gravação ou no modo normal de operação. Inicialmente, um processo está no modo normal. Uma chamada a `startrecording()` coloca este processo no modo de gravação e uma eventual chamada posterior a `stoprecording()` restaura o seu modo normal de operação. Estas duas *syscalls* devem retornar 0 se forem bem-sucedidas ou -1, caso o processo já esteja no modo requisitado pela *syscall*. Se um processo faz um `fork()`, o processo filho herda o modo do pai.

Todas as system calls realizadas por um processo, excetuando as três que você deve implementar, deverão ser gravadas quando este processo estiver em modo de gravação. Você deve gravar tais registros na PCB do processo. Estes registros devem ser removidos quando o processo termina a sua execução. Você também deve tratar apropriadamente (e.g. parar a gravação) o caso de não existir mais memória para gravar registros adicionais. Quando um processo faz um `fork()`, o processo filho deve iniciar com o seu histórico de gravação zerado.

Uma chamada a `fetchrecords()`, recuperará todos os registros de system calls do processo corrente da seguinte forma:

1. Se o seu primeiro argumento for `NULL`, ela retorna o número de registros, e o segundo argumento `num_records` é simplesmente ignorado.
2. Se `records` não for `NULL`, então ele deve apontar para um array prealocado para guardar os registros. Neste caso, `num_records` indica o tamanho do array (o número de registros). Então `fetchrecords` recupera até `num_records` e retorna o número de registros efetivamente lidos.

Para facilitar a sua vida, você pode usar a estrutura abaixo e colocá-la em um arquivo `record.h` para gravar as system calls:

```
enum recordtype { SYSCALL_NO, ARG_INTEGER, ARG_POINTER, ARG_STRING, RET_VALUE };

#define MAX_STR_LEN (21)

struct record {
  enum recordtype type;
  union recordvalue {
    int intval;
    void *ptrval;
    char strval[MAX_STR_LEN];
  } value;
};
```

Note que um registro pode ser um número de system call, um argumento ou um valor de retorno. A enumeração `recordtype` indica o tipo do registro, e a variável `type` grava o seu valor correspondente. Uma única chamada de sistema pode gerar vários registros. Eles devem aparecer na seguinte ordem: número de *syscall*, argumento 0, argumento 1,..., valor de retorno. Por exemplo, a system call `open("README'', 0)` (que retorna valor 3), gerará 4 registros:

```
SYSCALL_NO: SYS_open
ARG_STRING: "README"
ARG_INTEGER: 0
RET_VALUE: 3
```

2.2 Parte II (30%)

Você deverá criar um programa em C para testar a sua implementação das três *syscalls*. Mostre que o seu programa funciona para cada uma das combinações de argumentos, mostrando o histórico das system call realizadas. Faça um `fork()`, e mostre que o histórico iniciado no processo pai sobrevive no filho e no pai. Para facilitar a vida do monitor da disciplina, nomeie esse arquivo `testrecording.c`. Ele será o primeiro a ser testado.

Instruções de entrega

Você deve entregar todos os arquivos fonte que você modificou no xv6, e quaisquer outros arquivos adicionais (e.g. record.h). Lembre-se, se você modificou o Makefile, inclua-o também, senão não poderemos testá-lo. Crie um arquivo compactado `.zip, .tar.gz, .tgz` ou `.bz2` contendo todos os arquivos pertencentes ao seu EP. Não inclua código executável, somente código fonte (.h, .c, .S) e arquivos textos contendo quaisquer descrições suas. Identifique o seu EP de acordo com o exemplo abaixo:

```
EP2-Marcel-Alexandre.zip
```

Não esqueça de adicionar um arquivo `LEIAME.TXT` contendo os nomes dos participantes da equipe (máximo 2 pessoas), números USP e demais instruções necessárias para compilar o seu programa e entender o seu EP. Você é responsável pela clareza destas instruções. Entregue o seu EP eletronicamente no site da disciplina até a data final. Envie somente um EP para a sua equipe.