

Sistemas Operacionais - TP1

getcnt syscall on xv6

Matheus Flávio Gonçalves Silva - 2020006850

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

matheusfgs@ufmg.br

1 Introdução

Neste trabalho é utilizado sistema operacional de aprendizado **XV6**. O sistema em questão é feito de modo a abranger diversos conceitos chaves **UNIX**, e, para utilizá-lo, é necessário compilar os arquivos fontes com a utilização do processador emulado **QEMU**

2 Preparação

Para executar o sistema e realizar o trabalho, é necessário seguir o seguinte passo-a-passo:

2.1 Instalação de dependências

A instalação das dependências necessária é feita em ambiente Linux de base Ubuntu (Linux Mint) com a execução do seguinte comando no terminal:

```
1 sudo apt-get install git build-essential gdb-multiarch qemu-system-misc  
   ↪ gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu
```

2.2 Download do XV6

O Download dos arquivos do XV6 é simplesmente feito por meio da cópia ou fork do repositório do projeto original do MIT. Para isso, basta simplesmente executar o comando a seguir caso prefira clonar o repositório

```
1 # Clone  
2 git clone https://github.com/mit-pdos/xv6-riscv.git
```

Caso prefira fazer um fork, basta selecionar a opção na página do Github.

2.3 Execução

A execução do sistema, que é feita de forma similar à execução de um terminal padrão do linux, é feita por meio do comando:

```
1 make qemu
```

Ao executar o comando acima, é aberto um emulador de terminal que aceita os comandos definidos no sistema.

3 Implementação

3.1 Estrutura de Dados

A Estrutura de Dados para manter a quantidade de chamadas de cada função é simplesmente um vetor de 22 posições inicializado com 0 em cada posição, cada índice armazenando a quantidade de execuções do comando com o pid correspondente ao índice + 1.

```
1 //proc.c
2 int sys_cnt[22] = {0};
```

3.2 Atualização da Estrutura de Dados

Tendo em mente a Estrutura de Dados, essa variável é simplesmente utilizada como sendo uma variável global que pode ser acessada por todo código que importe "proc.h":

```
1 //Exemplo do sysfile.c atualizando as chamadas do comando "dup"
2 extern int sys_cnt[22];
3
4 // Fetch the nth word-sized system call argument as a file descriptor
5 // and return both the descriptor and the corresponding struct file.
6 static int
7 @@ -57,6 +59,8 @@ sys_dup(void)
8 struct file *f;
9 int fd;
10
11 sys_cnt[SYS_dup - 1]++; //Atualização da quantidade de chamadas do processo de pid
    ↳ referente ao dup
```

Todos os processos são atualizados da mesma forma, apenas alterando o valor atribuído à posição do vetor que é manipulada.

Foi feito também o include da lib syscall.h para utilizar os defines lá declarados tornando mais fácil entender os pids ao não utilizar os pids numéricos diretamente.

3.3 Implementação da função

A implementação da função em si segue os padrões definidos anteriormente no projeto, levando como base, por exemplo, a chamada "argint" para pegar o pid passado como parâmetro e utilizando manipulações simples de inteiros com o retorno da posição do vetor referente à quantidade de chamadas do processo passado como parâmetro.

Essa implementação é feita como segue em **sysfile.c**:

```
1  uint64
2  sys_getcnt(void)
3  {
4      sys_cnt[SYS_getcnt - 1]++;
5
6      int sys_ID;
7      argint(0, &sys_ID); //argfd mostra como pegar um argumento int
8      sys_ID--;
9      return sys_cnt[sys_ID];
```

3.4 Impressão do comando

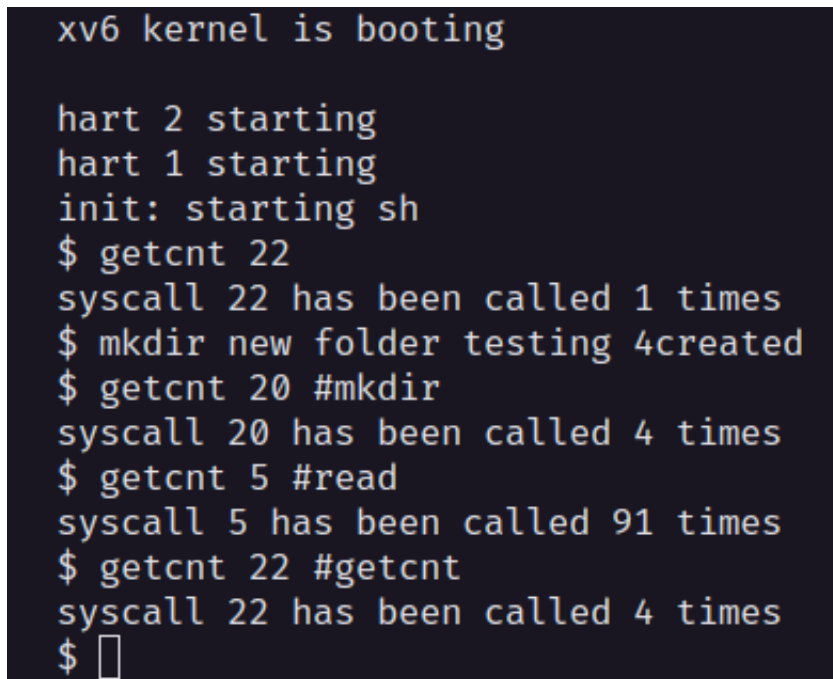
Por fim, deve-se tratar da impressão do comando feita de acordo com o passado na especificação do TP. Isso é feito um arquivo criado à parte **getcnt.c**:

```
1  #include "kernel/types.h"
2  #include "kernel/stat.h"
3  #include "user/user.h"
4
5  extern int sys_cnt[22];
6
7  int
8  main(int argc, char *argv[])
9  {
10     if(argc < 2){
11         fprintf(2, "usage: getcnt pid...\n");
12         exit(1);
13     }
14
15     int sys_ID = atoi(argv[1]);
16     int id_cnt = getcnt(sys_ID);
17
18     fprintf(2, "syscall %s has been called %d times\n", argv[1], id_cnt);
```

```
19
20     exit(0);
21 }
```

4 Teste

O teste da implementação foi feito seguindo uma rotina simulando a utilização simples de um usuário interagindo com o terminal, sendo realizados uma verificação de contagem, a criação de 4 pastas, a contagem do comando mkdir, a contagem do comando read, e a contagem do getcnt finalizando a rotina de testes.

A terminal window with a dark background and light-colored text. The text shows the booting process of xv6, followed by the starting of hart 2 and hart 1, and the init process starting a shell. The user then runs a series of commands to test system calls: getcnt 22, mkdir new folder testing 4created, getcnt 20 #mkdir, getcnt 5 #read, and getcnt 22 #getcnt. The output shows the number of times each system call has been called: 1 for syscall 22, 4 for syscall 20, 91 for syscall 5, and 4 for syscall 22. The prompt is a dollar sign followed by a space.

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ getcnt 22
syscall 22 has been called 1 times
$ mkdir new folder testing 4created
$ getcnt 20 #mkdir
syscall 20 has been called 4 times
$ getcnt 5 #read
syscall 5 has been called 91 times
$ getcnt 22 #getcnt
syscall 22 has been called 4 times
$ 
```

Figura 1: Rotina de testes simulando usuário