

Sistemas Operacionais - Turma TW

Implementação do Comando time em xv6

Professor Daniel Fernandes Macedo
Otávio de Meira Lima
2019054900

1- Introdução

O desafio do trabalho foi implementar no Sistema Operacional xv6, que consiste em uma reimplementação do Unix, o comando “time”. Quando executado, o comando solicita um processo a ser executado e seus parâmetros e retorna o tempo de execução dividido em três categorias:

- Tempo real: aquele gasto por todo o processo
- Tempo de sistema: aquele gasto dentro do kernel
- Tempo de usuário: aquele gasto fora do kernel

```
CIANDT\otavio1@lnb028130bhz:~/Documents/UFMG/SO/xv6-public$ time firefox
real    0m6,193s
user    0m3,611s
sys     0m0,875s
```

Exemplo de chamada time no Linux.

2- implementação

Para criar a nova chamada “time” dentro do xv6, é preciso registrar a chamada em alguns arquivos.

syscall.h -> arquivo onde a chamada é enumerada.

```
#define SYS_time 23
```

Syscall.c -> arquivo onde a chamada terá um ponteiro e será inserida em um vetor de chamadas.

```
[SYS_time]    sys_time,
extern int sys_time(void);
```

Sysproc.c -> arquivo onde a chamada será implementada.

```
101  int sysTime;
102
103  int
104  sys_time(void) {
105      return sysTime;
106  }
```

usys.S -> contém a interface para o programa acessar a system call.

```
SYSCALL(time)
```

user.h -> onde é definido a função que o programa irá chamar.

```
int time(void);
```

Makefile -> onde é introduzida a chamada ao sistema.

```
UPROGS=\
_cat\
_echo\
_forktest\
_grep\
_init\
_kill\
_ln\
_ls\
_mkdir\
_rm\
_sh\
_stressfs\
_usertests\
_wc\
_zombie\
_date\
_time\
```

Também deve-se criar um arquivo com o nome “time.c” para que o Makefile consiga alcançar o arquivo e executá-lo.

```

22 int
23 main(int argc, char *argv[])
24 {
25
26     int sysTimeStart = time();
27     int startTicks = uptime();
28     int pid = fork();
29
30     if (pid < 0){
31         printf(2, "Error: Invalid PID!\n");
32         exit();
33     }
34
35     if (pid > 0) wait();
36
37     if (pid == 0) {
38         if (exec(argv[1], argv + 1) < 0) {
39             printf(2, "Error: Exec fails!\n");
40             exit();
41         }
42     }
43
44     int sysTimeEnd = time();
45     int endTicks = uptime();
46
47     int realTime = (endTicks - startTicks)*10;
48     int sysTime = (sysTimeEnd - sysTimeStart)*10;
49     int userTime = realTime - sysTime;
50
51     printf(stdout, "real ");
52     printTimeMillisecondsPrecision(realTime);
53     printf(stdout, "user ");
54     printTimeMillisecondsPrecision(userTime);
55     printf(stdout, "sys ");
56     printTimeMillisecondsPrecision(sysTime);
57
58     exit();
59
60 }

```

No arquivo time.c, deve-se iniciar duas contabilizações de tempo: uma para o sistema e outra para o tempo real. Nesse mesmo arquivo, deve-se criar um processo filho com a função fork() para que, na hora da execução da chamada com a função exec(), o terminal não seja finalizado.

O tempo real é obtido apenas contabilizando os ticks desde o início da função.

O tempo de sistema é obtido pelo retorno da função time(), definida no arquivo sysproc.c como sys_time(). A variável que é retornada é uma variável declarada como externa no arquivo syscall.c, onde está implementada a função syscall.

```

141 | extern int sysTime;
142 |
143 | void
144 | syscall(void)
145 | {
146 |     int num;
147 |     struct proc *curproc = myproc();
148 |     int sysStartTime, sysEndTime;
149 |
150 |     num = curproc->tf->eax;
151 |     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
152 |         sysStartTime = ticks;
153 |         curproc->tf->eax = syscalls[num]();
154 |         sysEndTime = ticks;
155 |         if (num != 3) sysTime += sysEndTime - sysStartTime;
156 |
157 |     } else {
158 |         cprintf("%d %s: unknown sys call %d\n",
159 |             curproc->pid, curproc->name, num);
160 |         curproc->tf->eax = -1;
161 |     }
162 | }

```

As chamadas de sistema são realizadas na linha 153 ao acessar o vetor de chamadas com o número da chamada a ser realizada. Os ticks de início e de fim da operação são contabilizados. Como cada system call pode ter outras chamadas dentro de sua execução, é necessário incrementar uma variável sysTime que armazena o tempo gasto, em ticks, dentro do kernel.

Porém, é preciso ignorar a chamada de wait pois ela faz parte da execução do processo pai, já que é necessário esperar a chamada do filho terminar para que seja possível contabilizar o tempo de execução. Essa filtragem é feita com a condição `if(num != 3)` onde num é o número do processo e 3 é o número da chamada wait. Esses números são definidos previamente no arquivo `syscall.h`

O tempo de usuário é obtido subtraindo o tempo real pelo tempo de sistema, já que:

$$\text{real time} = \text{sys time} + \text{user time}$$

3- Testes

Como forma de teste, foram implementadas 3 novas chamadas que balanceiam tempo de sistema e usuário.

```
C systimetest.c U x
xv6-public > C systimetest.c > main(int, char * [])
1  #include "types.h"
2  #include "user.h"
3  #include "date.h"
4
5  int
6  main(int argc, char *argv[])
7  {
8
9      for (int i = 0; i < 10; i++) {
10         sleep(1);
11     }
12
13     exit();
14 }
```

O systimetest possui um programa com foco em chamadas de sistema, fazendo com que sua execução passe mais tempo dentro do kernel, o que significa que o tempo de sistema será relevante. A chamada sleep() é um bom exemplo.

```

C usertimetest.c U ×
xv6-public > C usertimetest.c > main(int, char *[])
1  #include "types.h"
2  #include "user.h"
3  #include "date.h"
4
5  int stdout = 1;
6  int stderr = 2;
7
8  int
9  main(int argc, char *argv[])
10 {
11
12     float x = 0.0;
13     float pi = 3.141573;
14
15     for (int i = 0; i < 99999; i++){
16         for (int j = 0; j < 100; j++){
17             x += pi*pi;
18         }
19     }
20
21     printf(stdout, "%f\n", x);
22
23     exit();
24 }

```

Já o `usertimetest` faz várias contas, fazendo com que sua execução passe mais tempo fora do kernel. Dois loops aninhados realizam diversas tarefas, aumentando o tempo de usuário.

```
yscall.c M  C sysproc.c M  [as] usys.S M  h user.h M  Makefile M  C realltimetest.c U x
xv6-public > C realltimetest.c > main(int, char * [])
1  #include "types.h"
2  #include "user.h"
3  #include "date.h"
4
5  int stdout = 1;
6  int stderr = 2;
7
8  int
9  main(int argc, char *argv[])
10 {
11
12     float x = 0.0;
13     float pi = 3.141573;
14
15     for (int i = 0; i < 10; i++){
16         sleep(1);
17     }
18
19     for (int i = 0; i < 99999; i++){
20         for (int j = 0; j < 100; j++){
21             x += pi*pi;
22         }
23     }
24
25     printf(stdout, "%f\n", x);
26
27     exit();
```

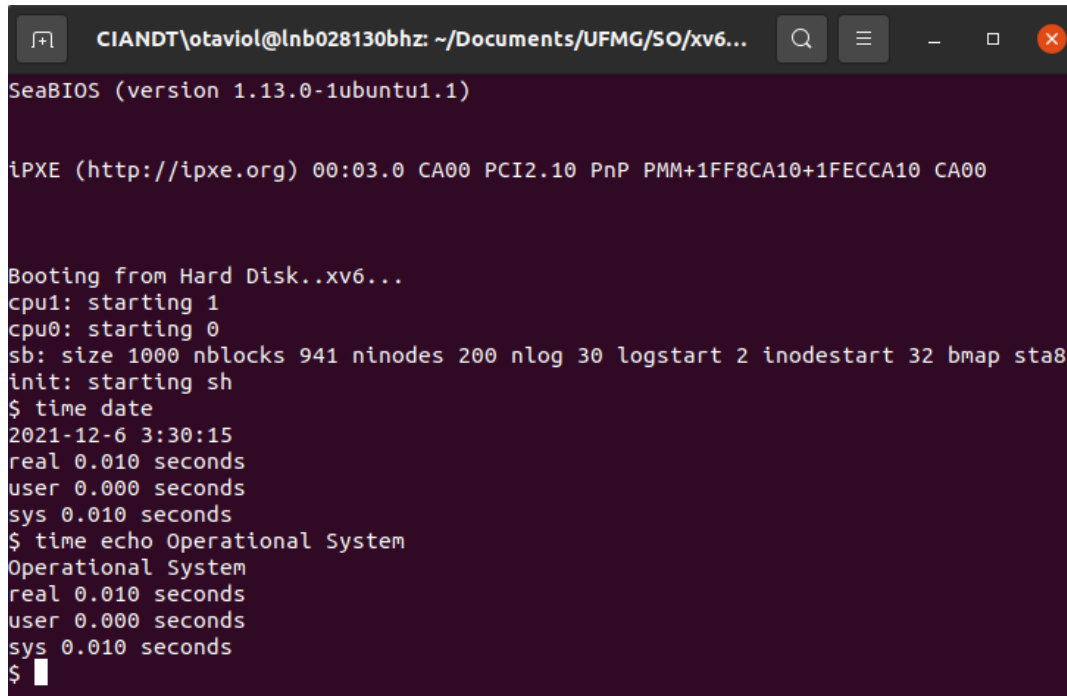
O realltimetest possui uma implementação mista. Dessa forma, os tempos de sistema e usuário serão balanceados.

```
CIANDT\otaviol@lnb028130bh: ~/Documents/UFMG/SO/xv6...
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00

Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
$ time systimetest
real 1.010 seconds
user 0.000 seconds
sys 1.010 seconds
$ time usertimetest
real 1.070 seconds
user 1.060 seconds
sys 0.010 seconds
$ time realltimetest
%f
real 2.060 seconds
user 1.050 seconds
sys 1.010 seconds
$
```

Exemplo da execução da chamada time com as chamadas de teste.

A chamada `time` deve funcionar com qualquer outra chamada a ser executada, assim como no Linux. Podemos usar como exemplo a chamada `date` e a chamada `echo`.



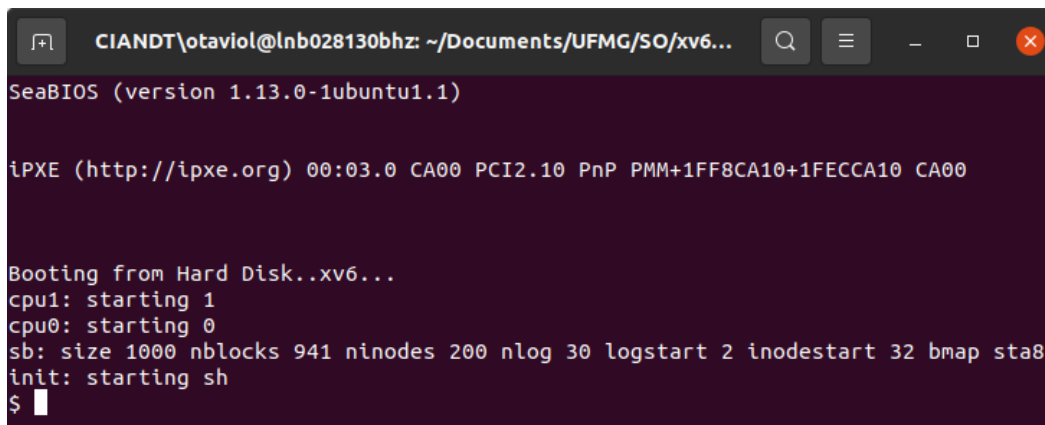
```
CIANDT\otaviol@lnb028130bhz: ~/Documents/UFGM/SO/xv6...  
SeaBIOS (version 1.13.0-1ubuntu1.1)  
  
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00  
  
Booting from Hard Disk..xv6...  
cpu1: starting 1  
cpu0: starting 0  
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8  
init: starting sh  
$ time date  
2021-12-6 3:30:15  
real 0.010 seconds  
user 0.000 seconds  
sys 0.010 seconds  
$ time echo Operational System  
Operational System  
real 0.010 seconds  
user 0.000 seconds  
sys 0.010 seconds  
$
```

4- Execução

Para executar o `xv6`, deve-se rodar a seguinte sequência de comandos dentro da pasta `xv6-public`:

1. `make`
2. `make qemu-nox`

Após a execução desses comandos, seu terminal deve estar semelhante a esse:



```
CIANDT\otaviol@lnb028130bhz: ~/Documents/UFGM/SO/xv6...  
SeaBIOS (version 1.13.0-1ubuntu1.1)  
  
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00  
  
Booting from Hard Disk..xv6...  
cpu1: starting 1  
cpu0: starting 0  
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8  
init: starting sh  
$
```


Para executar os testes, é necessário estar com o xv6 rodando. Cada teste possui sua chamada e podemos testá-las da seguinte forma:

- time systimetest
- time usertimetest
- time realtimetest

Para sair da execução do xv6, basta apenas clicar em Ctrl+A e depois em X.