



## Sistemas Operacionais

Rodrigo Rubira Branco  
rodrigo@kernelhacking.com  
rodrigo@fgp.com.br

# Chamadas ao Sistema (modo usuario)

```
#include <sys/types.h>
#include <unistd.h>

main(){
    if ( setuid(1001) ) {
        perror("\n Mudando de usuario\n");
        exit(-1);
    }
    printf("\n UserID: %d\n",getuid());
    if ( chdir("/teste") ) {
        perror("\n Mudando para o diretorio /teste\n");
        exit(-1);
    }
    printf("\n Tive permissoes\n");
}
```

# Chamadas ao Sistema (modo usuario)

```
#include <sys/syscall.h>
```

```
/* SYS_exit = NR_exit = 1 (definido em  
/usr/include/asm-i386/unistd.h */
```

```
#define sys_mycall(x) syscall(SYS_exit,x);
```

```
main()
```

```
{
```

```
    sys_mycall(10);
```

```
}
```

# Chamadas ao Sistema (modo usuario) - strace

execve("./syscall", ["./syscall"], [/\* 19 vars \*/]) = 0

uname({sys="Linux", node="notedell", ...}) = 0

brk(0) = 0x8049600

■■■

munmap(0x40017000, 82007) = 0

\_exit(10) = ?

# fork() e execve()

```
#include <sys/types.h>
#include <unistd.h>

main() {
    int status;
    if ( fork() ) {
        /* Pai */
        printf("\n codigo pai executando normalmente\n");
        wait(&status); // espera o filho terminar
        printf("\n continuando apos termino do filho\n");
    } else {
        /* Filho */
        execve("/bin/bash", NULL, NULL);

        printf("\n nao devera executar\n");
    }
}
```



# Comunicacao Inter-Processos – pipe()

```
#include <unistd.h>
```

```
main()
```

```
{
```

```
    int childpid, mypid;
```

```
    int pipecom[2]; /* pipecom[0] para leitura e pipecom
```

```
    mypid=getpid();
```

```
    if ( pipe(pipecom) )
```

```
    {
```

```
        perror("\n Criando pipe\n");
```

```
        exit(-1);
```

```
    }
```

... continua

```

if ( (childpid=fork()) ) {
    /*Codigo pai - executara primeiro */
    printf("\n Processo pai - Pid do filho:
%d\n",childpid);
    char *escrever="Escrevi no pipe";
    int status;

    write(pipecom[1],escrever,strlen(escrever));
    wait(&status);
    printf("\n Processo pai - Filho terminou sua
execucao\n");
}else {
    /*Codigo filho */
    printf("\n Processo filho - Pid do pai:
%d\n",mypid);
    char *ler;
    ler=(char *) malloc(sizeof(char) * 100);
    read(pipecom[0], ler, (sizeof(char) * 100));
    printf("\n Processo filho - Lido: %s\n",ler);
}
}

```

# Comunicacao Inter-Processos – popen()

```
#include <stdio.h>
#include <stdlib.h>
main() {
    FILE *pipecom;
    char *saida;

    if ( (pipecom=popen("/bin/echo OK", "r") ) ==NULL) {
        perror("\n Popen\n");
        exit(-1);
    }
    saida=(char *)malloc(1000+1);
    fgets(saida, 1000, pipecom);
    printf("\n lido: %s\n", saida);
    fgets(saida, 1000, pipecom);
    pclose(pipecom);
}
```



# Scheduler – Alarm() e Signal()

```
/*Ideia para fazer um scheduler em modo usuario*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <signal.h>  
  
/* Handler do sinal */  
void hookalarm(int sinal) {  
  
    printf("\n Sinal Recebido \n");  
  
    /* Agenda um novo sinal */  
    alarm(5);  
  
    return ;  
}
```

... continua

[www.fgp.com.br](http://www.fgp.com.br)



```
int main(int argc, char **argv)
{
    /* Adiciona o handler */
    if (signal(SIGALRM, hookalarm) < 0) {
        perror("handler");
        return -1;
    }

    /* Schedules um sinal para daqui 5 segundos */
    alarm(5);

    while(1) {

        printf("loop\r");

    }

    /* Nunca chegara aqui */

    return 1;
}
```





FIM! Será mesmo?

DÚVIDAS?!?

Rodrigo Rubira Branco  
rodrigo@kernelhacking.com