



# Sistemas Operacionais

Trabalho 01 - Processos, I/O e CPU *bounds*  
[https://github.com/alexsantee/Trabalhos\\_SO](https://github.com/alexsantee/Trabalhos_SO)

Alex Marcelino Santee - 10392226  
Rodrigo Augusto Valeretto - 10684792  
Leonardo Cerce Guioto - 10716640

## Introdução

Nesta apresentação, pretendemos expor os resultados obtidos ao executar os códigos na máquina virtual provida pelo docente. Os códigos tinham o objetivo de realizar determinadas chamadas de sistema, as quais seriam explicitadas com o uso de `strace`.

Dessa forma, os próximos *slides* terão a seguinte estrutura: uma representação do código, seguida pela saída após o uso do `strace` e a explicação de cada chamada de sistema que foi executada. Após isso, serão feitas algumas considerações e será apresentado o próximo código.

Por fim, após os códigos de chamada de sistema, serão apresentados os códigos *I/O bound* e *CPU bound*, seguindo o mesmo formato anterior, usando agora a ferramenta `time`.

Ordem escolhida:

- Gerenciamento de Memória (`brk()`, `mmap()`, `munmap()`);
- Processos (`execve()`, `fork()`, `waitpid()`);
- E/S e Arquivos (`mkdir()`, `open()`, `close()`, `write()`, `rmdir()`);
- CPU-bound;
- I/O-bound;

GitHub: [https://github.com/alexsantee/Trabalhos\\_SO](https://github.com/alexsantee/Trabalhos_SO)

## Como conectar na máquina que utilizamos nos testes

Inicialmente, é necessário acessar a máquina virtual; no *console* do Linux ou Windows, digite o seguinte comando:

```
ssh gso06@andromeda.lasdpc.icmc.usp.br -p 2316
```

Em seguida, basta inserir a senha:

```
742QX5Q0
```

## Como compilar os códigos

Todos os códigos estão disponíveis no github. No caso dos exemplos, compilamos estaticamente os programas para evitar as chamadas que linkam a *libc*. Isso simplifica a análise das *syscalls* pelo *strace*.

O repositório conterá todos os arquivos *.c* exemplos das chamadas de sistema e I/O e CPU *bounds*. Junto a eles está um arquivo *makefile* para simplificar a compilação. Para compilar os todos os arquivos *.c*, digite o comando:

```
make all
```

Para compilar apenas um arquivo *file.c* específico, digite:

```
make file.out
```

Por fim, os códigos devem ser executados com:

```
./file.out
```

## Código: Gerenciamento de Memória - *brk()*

```
#include <stdlib.h>

int main()
{
    char *string;
    string = malloc(sizeof(char)*1024);
    free(string);
    return 0;
}
```

Saída recebida após o uso de *strace*:

```
execve("./exemplo_brk.out", ["/home/gso06/exemplo_brk.out"], [/* 29 vars */]) = 0 <0.000205>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000018>
brk(NULL) = 0x1eaf000 <0.000017>
brk(0x1eb01c0) = 0x1eb01c0 <0.000019>
arch_prctl(ARCH_SET_FS, 0x1eaf880) = 0 <0.000018>
readlink("/proc/self/exe", "/home/gso06/exemplo_brk.out", 4096) = 27 <0.000060>
brk(0x1ed11c0) = 0x1ed11c0 <0.000019>
brk(0x1ed2000) = 0x1ed2000 <0.000018>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000024>
brk(0x1ed1000) = 0x1ed1000 <0.000027>
exit_group(0) = ?
+++ exited with 0 +++
```

## Chamadas de sistema e o que significam

`execve()` : executa o programa apontado pelo nome do arquivo; este deve ser um binário executável.

`uname()` : retorna informações do *kernel*, tais como nome, versão etc.

**`brk()`** : modifica a localização do *program break*; dessa forma, aloca memória para o programa em execução.

`arch_prctl()` : define o estado de um processo ou *thread* de arquiteturas específicas.

`readlink()` : coloca os conteúdos do link simbólico *path* em um buffer.

`access()` : checa se o processo que o chamou tem permissão para acessar e/ou modificar determinado arquivo.

`exit_group()` : encerra todas as threads de um processo.

## Considerações

No caso do `malloc` da `glibc`, pequenas alocações de memória acontecem na *heap*, que fica dentro da memória do programa. Para aumentar o tamanho da *heap* ele faz o uso da *syscall* `brk()`.

Na saída do `strace` apresentada, os `brks` associados ao `malloc` e `free` são os dois últimos. O `malloc` aumenta a *heap* para alocar memória e o `free` a diminui. Ambos pela chamada `brk`.

## Código: Gerenciamento de Memória - `mmap()` e `munmap()`

```
#include<stdlib.h>

int main(){
    char *string;
    string = malloc(sizeof(char)*(64*1024*1024));
    free(string);
    return 0;
}
```

Saída recebida após o uso de `strace`:

```
execve("./exemplo_mmap_munmap.out", ["/home/gso06/exemplo_mmap_munmap.out"], [/* 29 vars */])
= 0 <0.000075>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000007>
brk(NULL) = 0x2185000 <0.000006>
brk(0x21861c0) = 0x21861c0 <0.000007>
arch_prctl(ARCH_SET_FS, 0x2185880) = 0 <0.000006>
readlink("/proc/self/exe", "/home/gso06/exemplo_mmap_munmap."..., 4096) = 35 <0.000018>
brk(0x21a71c0) = 0x21a71c0 <0.000007>
brk(0x21a8000) = 0x21a8000 <0.000006>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000008>
mmap(NULL, 67112960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd1c4bca000 <0.000007>
munmap(0x7fd1c4bca000, 67112960) = 0 <0.000010>
exit_group(0) = ?
+++ exited with 0 +++
```



## Chamadas de sistema e o que significam

**mmap( )** : mapeia arquivos ou dispositivos na memória, mapeando no espaço de endereçamento virtual do processo chamado.

**munmap( )** : apaga os mapeamentos nos endereços específicos e gera referências de memória inválidas nesses endereços.

## Considerações

No caso do `malloc` da `glibc`, grandes alocações de memória acontecem fora da *heap* e são mapeadas em uma nova região com a chamada do `mmap( )`. O comando `free` desfaz a alocação da memória e faz chamada para o `munmap( )`.

Código: Processos - execve()

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

int main(){
    int res = 0;
    res = execve("/bin/bash", 0, 0);
    if(res != -1)
    {
        printf("O programa executou corretamente!\n");
    }else{
        printf("Ocorreu um erro durante a execução!\n");
    }
}
```

Saída após o uso de strace:

```
execve("/.exemplo_execve.out", ["/.exemplo_execve.out", /* 29 vars */] = 0 <0.000070>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000006>
brk(NULL) = 0x2607000 <0.000006>
brk(0x26081c0) = 0x26081c0 <0.000007>
arch_prctl(ARCH_SET_FS, 0x2607880) = 0 <0.000006>
readlink("/proc/self/exe", "/home/gso06/exemplo_execve.out", 4096) = 30 <0.000018>
brk(0x26291c0) = 0x26291c0 <0.000007>
brk(0x262a000) = 0x262a000 <0.000005>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000009>
execve("/bin/bash", NULL, NULL) = 0 <0.000064>
brk(NULL) = 0x1309000 <0.000005>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000008>
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory) <0.000008>
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3 <0.000009>
fstat(3, {st_mode=S_IFREG|0644, st_size=74000, ...}) = 0 <0.000006>
mmap(NULL, 74000, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fedb122b000 <0.000007>
close(3) = 0 <0.000006>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000007>
open("/lib/x86_64-linux-gnu/libtinfo.so.5", O_RDONLY|O_CLOEXEC) = 3 <0.000009>
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\3\0\0\1\0\0\0p\310\0\0\0\0\0"... , 832) = 832 <0.000006>
fstat(3, {st_mode=S_IFREG|0644, st_size=167240, ...}) = 0 <0.000006>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fedb122a000 <0.000006>
mmap(NULL, 2264256, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fedb0df0000 <0.000007>
mprotect(0x7fedb0e15000, 2093056, PROT_NONE) = 0 <0.000008>
mmap(0x7fedb1014000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7fedb1014000 <0.000008>
close(3) = 0 <0.000006>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000008>
open("/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3 <0.000009>
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\3\0\0\1\0\0\0r\240\0\0\0\0\0"... , 832) = 832 <0.000006>
fstat(3, {st_mode=S_IFREG|0644, st_size=14608, ...}) = 0 <0.000006>
mmap(NULL, 2109680, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fedb0bec000 <0.000009>
mprotect(0x7fedb0bef000, 2093056, PROT_NONE) = 0 <0.000011>
mmap(0x7fedb0dee000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7fedb0dee000 <0.000009>
close(3) = 0 <0.000006>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000007>
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3 <0.000008>
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\3\0\0\1\0\0\0p\t\2\0\0\0\0\0"... , 832) = 832 <0.000006>
fstat(3, {st_mode=S_IFREG|0755, st_size=1868984, ...}) = 0 <0.000006>
mmap(NULL, 3971488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fedb0822000 <0.000007>
mprotect(0x7fedb09e2000, 2097152, PROT_NONE) = 0 <0.000009>
mmap(0x7fedb0be2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c0000) = 0x7fedb0be2000 <0.000009>
mmap(0x7fedb0be8000, 14752, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fedb0be8000 <0.000008>
close(3) = 0 <0.000006>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fedb1229000 <0.000007>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fedb1228000 <0.000006>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fedb1227000 <0.000006>
arch_prctl(ARCH_SET_FS, 0x7fedb1228700) = 0 <0.000006>
```

```
mprotect(0x7fedb0be2000, 16384, PROT_READ) = 0 <0.000009>
mprotect(0x7fedb0dee000, 4096, PROT_READ) = 0 <0.000008>
mprotect(0x7fedb1014000, 16384, PROT_READ) = 0 <0.000007>
mprotect(0x6f3000, 4096, PROT_READ) = 0 <0.000008>
mprotect(0x7fedb123e000, 4096, PROT_READ) = 0 <0.000007>
munmap(0x7fedb122b000, 74000) = 0 <0.000011>
open("/dev/tty", O_RDWR|O_NONBLOCK) = 3 <0.000012>
close(3) = 0 <0.000008>
brk(NULL) = 0x1309000 <0.000006>
brk(0x130a000) = 0x130a000 <0.000007>
brk(0x130b000) = 0x130b000 <0.000006>
getuid() = 1148 <0.000006>
getgid() = 63000 <0.000005>
geteuid() = 1148 <0.000006>
getegid() = 63000 <0.000006>
rt_sigprocmask(SIG_BLOCK, NULL, [], 8) = 0 <0.000006>
brk(0x130c000) = 0x130c000 <0.000006>
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0 <0.000008>
ioctl(2, TCGETS, 0x7ffdbf2b7c00) = -1 ENOTTY (Inappropriate ioctl for device) <0.000006>
sysinfo({uptime=575701, loads=[192, 1984, 2976], totalram=1997692928, freeram=1002250240, sharedram=0, bufferram=159244288, totalswap=1070592000, freeswap=1070592000, procs=128, totalhigh=0, freehigh=0, mem_unit=1}) = 0 <0.000009>
brk(0x130d000) = 0x130d000 <0.000006>
rt_sigaction(SIGCHLD, {SIG_DFL, [], SA_RESTORER|SA_RESTART, 0x7fedb08574b0}, {SIG_DFL, [], 0}, 8) = 0 <0.000006>
rt_sigaction(SIGCHLD, {SIG_DFL, [], SA_RESTORER|SA_RESTART, 0x7fedb08574b0}, {SIG_DFL, [], SA_RESTORER|SA_RESTART, 0x7fedb08574b0}, 8) = 0 <0.000006>
rt_sigaction(SIGINT, {SIG_DFL, [], SA_RESTORER, 0x7fedb08574b0}, {SIG_DFL, [], 0}, 8) = 0 <0.000006>
rt_sigaction(SIGINT, {SIG_DFL, [], SA_RESTORER, 0x7fedb08574b0}, {SIG_DFL, [], SA_RESTORER, 0x7fedb08574b0}, 8) = 0 <0.000006>
rt_sigaction(SIGQUIT, {SIG_DFL, [], SA_RESTORER, 0x7fedb08574b0}, {SIG_DFL, [], 0}, 8) = 0 <0.000005>
rt_sigaction(SIGQUIT, {SIG_DFL, [], SA_RESTORER, 0x7fedb08574b0}, {SIG_DFL, [], SA_RESTORER, 0x7fedb08574b0}, 8) = 0 <0.000006>
rt_sigprocmask(SIG_BLOCK, NULL, [], 8) = 0 <0.000005>
rt_sigaction(SIGQUIT, {SIG_IGN, [], SA_RESTORER, 0x7fedb08574b0}, {SIG_DFL, [], SA_RESTORER, 0x7fedb08574b0}, 8) = 0 <0.000006>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000006>
brk(0x130f000) = 0x130f000 <0.000007>
getcwd("/home/gso06", 4096) = 12 <0.000006>
getpid() = 21654 <0.000006>
getppid() = 21652 <0.000006>
stat(".", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0 <0.000007>
stat("/usr/local/sbin/bash", 0x7ffdbf2b7900) = -1 ENOENT (No such file or directory) <0.000008>
stat("/usr/local/bin/bash", 0x7ffdbf2b7900) = -1 ENOENT (No such file or directory) <0.000007>
stat("/usr/sbin/bash", 0x7ffdbf2b7900) = -1 ENOENT (No such file or directory) <0.000007>
stat("/usr/bin/bash", 0x7ffdbf2b7900) = -1 ENOENT (No such file or directory) <0.000007>
stat("/sbin/bash", 0x7ffdbf2b7900) = -1 ENOENT (No such file or directory) <0.000007>
stat("/bin/bash", {st_mode=S_IFREG|0755, st_size=1037528, ...}) = 0 <0.000007>
stat("/bin/bash", {st_mode=S_IFREG|0755, st_size=1037528, ...}) = 0 <0.000007>
getuid() = 1148 <0.000006>
getgid() = 63000 <0.000006>
geteuid() = 1148 <0.000006>
getgid() = 63000 <0.000006>
```



```

access("/bin/bash", X_OK) = 0 <0.000007>
stat("/bin/bash", {st_mode=S_IFREG|0755, st_size=1037528, ...}) = 0 <0.000007>
geteuid() = 1148 <0.000006>
getegid() = 63000 <0.000006>
getuid() = 1148 <0.000005>
getgid() = 63000 <0.000006>
access("/bin/bash", R_OK) = 0 <0.000007>
stat("/bin/bash", {st_mode=S_IFREG|0755, st_size=1037528, ...}) = 0 <0.000007>
stat("/bin/bash", {st_mode=S_IFREG|0755, st_size=1037528, ...}) = 0 <0.000007>
geteuid() = 1148 <0.000006>
getegid() = 63000 <0.000006>
getuid() = 1148 <0.000005>
getgid() = 63000 <0.000005>
access("/bin/bash", X_OK) = 0 <0.000007>
stat("/bin/bash", {st_mode=S_IFREG|0755, st_size=1037528, ...}) = 0 <0.000007>
geteuid() = 1148 <0.000006>
getegid() = 63000 <0.000006>
getuid() = 1148 <0.000005>
getgid() = 63000 <0.000006>
access("/bin/bash", R_OK) = 0 <0.000007>
brk(0x131000) = 0x131000 <0.000006>
socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3 <0.000011>
connect(3, {sa_family=AF_LOCAL, sun_path="/var/run/nscd/socket"}, 110) = 0 <0.000035>
sendto(3, "\2\0\0\0\v\0\0\0\7\0\0\0passwd\0", 19, MSG_NOSIGNAL, NULL, 0) = 19 <0.000034>
poll([{fd=3, events=POLLIN|POLLERR|POLLHUP}], 1, 5000) = 1 ([{fd=3, revents=POLLIN|POLLHUP}]) <0.000007>
recvmsg(3, {msg_name(0)=NULL, msg_iov(2)=[{"passwd\0", 7}, {"\3100\3\0\0\0\0", 8}], msg_controllen=24, [{cmsg_len=20, cmsg_level=SOL_SOCKET, cmsg_type=SCM_RIGHTS, [4]}]},
msg_flags=MSG_CMSG_CLOEXEC, MSG_CMSG_CLOEXEC) = 15 <0.000008>
mmap(NULL, 217032, PROT_READ, MAP_SHARED, 4, 0) = 0x7fedb11f2000 <0.000008>
close(4) = 0 <0.000006>
close(3) = 0 <0.000008>
getpgrp() = 21648 <0.000006>
rt_sigaction(SIGCHLD, {0x447b10, [], SA_RESTORER|SA_RESTART, 0x7fedb08574b0}, {SIG_DFL, [], SA_RESTORER|SA_RESTART, 0x7fedb08574b0}, 8) = 0 <0.000006>
getrlimit(RLIMIT_NPROC, {rlim_cur=31393, rlim_max=31393}) = 0 <0.000006>
brk(0x1311000) = 0x1311000 <0.000007>
rt_sigprocmask(SIG_BLOCK, NULL, [], 8) = 0 <0.000006>
fcntl(0, F_GETFL) = 0x8402 (flags O_RDWR|O_APPEND|O_LARGEFILE) <0.000006>
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0 <0.000006>
lseek(0, 0, SEEK_CUR) = -1 EPIPE (Illegal seek) <0.000006>
read(0, "e", 1) = 1 <1.998143>
read(0, "x", 1) = 1 <0.000008>
read(0, "i", 1) = 1 <0.000006>
read(0, "t", 1) = 1 <0.000006>
read(0, "\n", 1) = 1 <0.000006>
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0 <0.000007>
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 <0.000005>
exit_group(0) = ?
+++ exited with 0 +++

```

## Chamadas de sistema e o que significam

**execve()** - executa o programa apontado pelo nome do arquivo; este deve ser um binário executável.

**open()** - cria ou abre um arquivo/dispositivo, retornando um descritor de arquivo.

**read()** - tenta ler um número determinado de *bytes* através de um descritor de arquivo.

**fstat()** - retorna informações sobre um arquivo usando um descritor de arquivo.

**mprotect()** - ativa a proteção na região de memória em que atua.

**close()** - fecha um descritor de arquivo.

**getuid()** - retorna o ID real do *user* do processo que está sendo chamado.

**getgid()** - retorna o ID real do grupo do processo que está sendo chamado.

**geteuid()** - retorna o ID efetivo do *user* do processo que está sendo chamado.

**getegid()** - retorna o ID efetivo do grupo do processo que está sendo chamado.

`rt_sigprocmask()` - examina e muda a máscara de sinais do processo que está sendo chamado.

`ioctl()` - manipula parâmetros subjacentes de arquivos especiais de dispositivos.

`sysinfo()` - retorna informações sobre estatísticas gerais do sistema.

`rt_sigaction()` - examina e muda a ação de um processo perante um sinal específico.

`get_cwd()` - retorna o diretório de execução do processo.

`getpid()` - retorna o ID do processo que fez a chamada.

`getppid()` - retorna o ID do parente do processo que fez a chamada.

`stat()` - retorna informações de um arquivo sem fazer uso de um descritor.

`socket()` - cria um ponto de comunicação, retornando um descritor.

`connect()` - inicia a conexão com um *socket*.

`sendto()` - envia uma mensagem através de um *socket*.

`poll()` - espera um evento ocorrer num descritor de arquivo.

`recvmsg()` - recebe mensagens/informações de um *socket*.

`getpgrp()` - retorna o PGID do processo especificado.

`getrlimit()` - retorna o limite de recursos do sistema.

`fcntl()` - manipula um descritor de arquivo.

`lseek()` - reposiciona o *offset* de um arquivo aberto referenciado por um descritor de arquivo.

## Considerações

Houve um grande número de *syscalls* por conta da execução do */bin/bash*, que interage bastante com o sistema.

O `print` que deveria ser realizado pelo código acaba não acontecendo pois o programa executado pelo `execve` toma o lugar do programa que estava sendo executado anteriormente e o fluxo de execução dele se perde.



## Código: Processos - *fork()*

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

int main(){
    pid_t pid;
    pid = fork();
    if(pid == 0)
        printf("sou o filho\n");
    else{
        usleep(1000);
        printf("sou o pai\n");
    }
    return 0;
}
```

Saída recebida após o uso de *strace*:

```
execve("./exemplo_fork.out", ["/.exemplo_fork.out"], [/* 29 vars */]) = 0 <0.000076>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000006>
brk(NULL) = 0x9e4000 <0.000006>
brk(0x9e51c0) = 0x9e51c0 <0.000006>
arch_prctl(ARCH_SET_FS, 0x9e4880) = 0 <0.000006>
readlink("/proc/self/exe", "/home/gso06/exemplo_fork.out", 4096) = 28 <0.000026>
brk(0xa061c0) = 0xa061c0 <0.000006>
brk(0xa07000) = 0xa07000 <0.000005>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000008>
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x9e4b50) = 21658 <0.000023>
nanosleep({0, 1000000}, NULL) = ? ERESTART_RESTARTBLOCK (Interrupted by signal) <0.000080>
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=21658, si_uid=1148, si_status=0, si_utime=0, si_stime=0} ---
restart_syscall(<... resuming interrupted nanosleep ...>) = 0 <0.001108>
fstat(1, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0 <0.000022>
ioctl(1, TCGETS, 0x7ffe6b770180) = -1 ENOTTY (Inappropriate ioctl for device) <0.000019>
write(1, "sou o pai\n", 10) = 10 <0.000019>
exit_group(0) = ?
+++ exited with 0 +++
```

## Chamadas de sistema e o que significam

**clone()** - chamado por *fork()*, cria um processo filho e permite que este compartilhe parte de seu contexto de execução com o processo que realizou a chamada (processo-pai).

*nanosleep()* - suspende a execução de um *thread* por um tempo determinado.

*restart\_syscall()* - reinicia uma chamada de sistema.

## Considerações

Usamos a função *fork()* do C, mas ela por algum motivo é um wrapper para a *syscall clone()* ao invés de *fork()*.

A *syscall write("sou o filho")* não aparece no *strace* porque o *strace* não segue o processo filho, que foi gerado na *syscall clone()*.

## Código: Processos - `waitpid()`

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    pid_t pid;
    pid = fork();

    if(pid == 0)
    {
        usleep(1000);
        printf("Processo filho finaliza\n");
    }else{
        waitpid(0, NULL, 0);
        printf("Processo Pai finaliza após esperar o filho finalizar\n");
    }
}
```

Saída recebida após o uso de `strace`:

```
execve("../exemplo_waitpid.out", ["../exemplo_waitpid.out"], [/ 29 vars */]) = 0 <0.000072>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000006>
brk(NULL) = 0x2259000 <0.000006>
brk(0x225a1c0) = 0x225a1c0 <0.000007>
arch_prctl(ARCH_SET_FS, 0x2259880) = 0 <0.000006>
readlink("/proc/self/exe", "/home/gso06/exemplo_waitpid.out", 4096) = 31 <0.000018>
brk(0x227b1c0) = 0x227b1c0 <0.000006>
brk(0x227c000) = 0x227c000 <0.000006>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000008>
clone(child_stack=0, flags=CLONE_CHILD_CLEARPID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x2259b50) = 21674 <0.000021>
wait4(0, NULL, 0, NULL) = 21674 <0.001414>
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=21674, si_uid=1148, si_status=0, si_utime=0, si_stime=0} ---
fstat(1, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0 <0.000006>
ioctl(1, TCGETS, 0x7ffe6617e730) = -1 ENOTTY (Inappropriate ioctl for device) <0.000006>
write(1, "Processo Pai finaliza ap\303\263s espe"..., 54) = 54 <0.000006>
exit_group(0) = ?
+++ exited with 0 +++
```

## Chamadas de sistema e o que significam

**wait4()** - chamado por *waitpid()*, espera o processo-filho mudar de estado; adicionalmente, retorna o uso de recursos do processo-filho.

## Código: E/S e Arquivos - `mkdir()`

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

#define SUCESSO 0
#define ERRO -1

int main()
{
    int err;

    err = mkdir("Pasta");

    if(err == ERRO)
    {
        printf("Erro ao criar pasta.");
        return ERRO;
    }

    return SUCESSO;
}
```

Saída recebida após o uso de `strace`:

```
execve("./exemplo_mkdir.out", [ "./exemplo_mkdir.out" ], [/* 29 vars */]) = 0 <0.000209>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000018>
brk(NULL) = 0x2205000 <0.000017>
brk(0x22061c0) = 0x22061c0 <0.000018>
arch_prctl(ARCH_SET_FS, 0x2205880) = 0 <0.000017>
readlink("/proc/self/exe", "/home/gso06/exemplo_mkdir.out", 4096) = 29 <0.000070>
brk(0x22271c0) = 0x22271c0 <0.000019>
brk(0x2228000) = 0x2228000 <0.000017>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000025>
mkdir("Pasta", 03777600202462050) = 0 <0.000091>
exit_group(0) = ?
+++ exited with 0 +++
```

## Chamadas de sistema e o que significam

**mkdir()** - tenta criar um novo diretório no diretório de trabalho.

## Considerações

Esse programa exemplifica a chamada de sistema `mkdir(const char* pathname, mode_t mode)`. Se o segundo argumento não é dado, cria-se um diretório com configurações padrões do sistema operacional.

## Código: E/S e Arquivos - *open()*, *close()*, *write()*

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

#define SUCESSO 0
#define ERRO -1
#define TAM 5

int main()
{
    FILE * fp;
    int err;
    fp = (FILE *) fopen("Arquivo.txt", "wt");
    if(fp == NULL)
    {
        printf("Erro ao abrir arquivo.");
        return ERRO;
    }
    err = fwrite("Hello", 1, TAM, fp);
    if(err != TAM)
    {
        printf("Erro ao escrever.");
        return ERRO;
    }
    fclose(fp);
    return SUCESSO;
}
```

Saída recebida após o uso de `strace`:

```
execve("./exemplo_open_close_write.out", ["/.exemplo_open_close_write.out"], [/* 29 vars */]) = 0 <0.000080>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000006>
brk(NULL) = 0xf20000 <0.000006>
brk(0xf211c0) = 0xf211c0 <0.000006>
arch_prctl(ARCH_SET_FS, 0xf20880) = 0 <0.000006>
readlink("/proc/self/exe", "/home/gso06/exemplo_open_close_w"... , 4096) = 40 <0.000018>
brk(0xf421c0) = 0xf421c0 <0.000006>
brk(0xf43000) = 0xf43000 <0.000006>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000008>
open("Arquivo.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3 <0.000024>
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0 <0.000006>
write(3, "Hello", 5) = 5 <0.000011>
close(3) = 0 <0.000039>
brk(0xf42000) = 0xf42000 <0.000008>
exit_group(0) = ?
+++ exited with 0 +++
```

Chamadas de sistema e o que significam

**open()** - cria ou abre um arquivo/dispositivo, retornando um descritor de arquivo.

**write()** - escreve um número determinado de *bytes* no arquivo apontado pelo descritor de arquivos.

**close()** - fecha um descritor de arquivo.



## Código: E/S e Arquivos - `rmdir()`

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

#define SUCESSO 0
#define ERRO -1

int main()
{
    int err;

    err = rmdir("Pasta");

    if(err == ERRO)
    {
        printf("Erro ao remover pasta.");
        return ERRO;
    }

    return SUCESSO;
}
```

Saída recebida após o uso de `strace`:

```
execve("./exemplo_rmdir.out", [ "./exemplo_rmdir.out" ], [ /* 29 vars */ ]) = 0 <0.000071>
uname({sysname="Linux", nodename="tau01-vm6", ...}) = 0 <0.000006>
brk(NULL) = 0xcd0000 <0.000005>
brk(0xcd11c0) = 0xcd11c0 <0.000006>
arch_prctl(ARCH_SET_FS, 0xcd0880) = 0 <0.000006>
readlink("/proc/self/exe", "/home/gso06/exemplo_rmdir.out", 4096) = 29 <0.000018>
brk(0xcf21c0) = 0xcf21c0 <0.000007>
brk(0xcf3000) = 0xcf3000 <0.000006>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000008>
rmdir("Pasta") = 0 <0.000031>
exit_group(0) = ?
+++ exited with 0 +++
```

Chamadas de sistema e o que significam

**rmdir()** - tenta deletar um diretório vazio no diretório especificado, ou no diretório de trabalho.

Considerações

Esse código exemplifica a chamada de sistema `rmdir(const char* pathname)`.

## Código - CPU-bound

```
#include<stdlib.h>

#define MATRIX_SIZE 500
#define MAX_NUMBER 1024

int main(){
    srand(0);
    float A[MATRIX_SIZE][MATRIX_SIZE];
    float B[MATRIX_SIZE][MATRIX_SIZE];
    float C[MATRIX_SIZE][MATRIX_SIZE];

    //inicializacao das matrizes A e B
    for(int i = 0; i < MATRIX_SIZE; i++){
        for(int j = 0; j < MATRIX_SIZE; j++){
            A[i][j] = rand() % MAX_NUMBER;
            B[i][j] = rand() % MAX_NUMBER;
        }
    }

    //calcula da matriz C = A*B
    for(int i = 0; i<MATRIX_SIZE; i++){
        for(int j = 0; j<MATRIX_SIZE; j++){
            C[i][j] = 0.0f;
            for(int k = 0; k < MATRIX_SIZE; k++){
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

Saída após o uso de `time`

```
real 0.83
user 0.83
sys 0.00

voluntary 1
involuntary 16
```

## Considerações

O programa *cpu-bound* apresenta quase todo o seu tempo de execução em espaço de usuário realizando operações. Durante sua execução, o kernel retirou o processo involuntariamente várias vezes, o que indica que o processo deve ter preenchido todo o quantum do escalonador várias vezes.

Como não foram realizadas muitas *syscalls*, o tempo do em espaço de *kernel* foi mínimo.

## Código - I/O-bound

```
#include<stdio.h>

int main(){
    int c;
    while( (c = getchar()) != EOF ){
        putchar(c);
    }
    return 0;
}
```

Saída após o uso de `time`

```
real 5.18
user 0.00
sys 0.00

voluntary 5
involuntary 0
```

## Considerações

O processo não gasta praticamente nenhum tempo de CPU porque fica maior parte do tempo aguardando a interrupção de entrada/saída, num estado bloqueado.

Também observa-se que todas as vezes o processo cedeu voluntariamente o processador, provavelmente quando aguardava a entrada de usuário.

Isso indica fortemente a natureza *IO-bound* do processo.