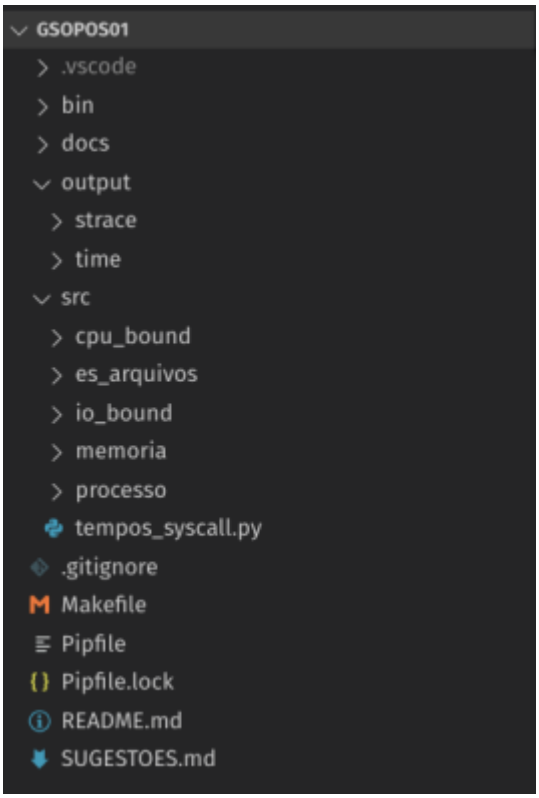
Grupo 1 - Módulo 1 - SSC5723

Integrantes

Aluno(a)	Número USP
Aguimar Ribeiro Júnior	11516281
Fabíola Malta Fleury	12501766
Gabriel Souto Ferrante	12620303

Estrutura do Projeto

Diretórios



Diretório	Descrição
./bin	Arquivos binários (resultados das compilações dos fontes)
./docs	Documentações

Diretório	Descrição
./output	Arquivos contendo os redirecionamentos das saídas de *strace* e *time*
./src	Código fonte

Compilação, Execução e Parsing dos Tempos Medidos

1. Compilação de todos os programas da parte 1 e 2

   `make all`
2. Execução de todos os programas da parte 1 e 2 e medições com *strace* e *time*

   `make run`
3. Parsing das saídas do *strace*

   `make parse`

Parte 1 - Chamadas de Sistema

Foram desenvolvidos nove programas, separados em três categorias: gerenciamento de memória, processos e Entrada/Saída e arquivos. Em seguida, utilizou-se a ferramenta *strace* para obter estatísticas relativas às chamadas de sistema que são realizadas por cada um desses programas e tempo de execução de cada chamada. A seguir é apresentada uma tabela com a descrição das chamadas de sistema utilizadas.

Descrição das Chamadas de Sistemas que Foram Utilizadas

Chamada de sistema	Declaração (unistd.h)	Descrição
access()	`int access(const char *pathname, int mode);`	Verifica se o processo em execução tem acesso ao arquivo informado.

Chamada de sistema	Declaração (unistd.h)	Descrição
arch_prctl()	`int arch_prctl(int code, unsigned long addr); int arch_prctl(int code, unsigned long *addr);`	Configurações de estado de processo ou thread específicos de arquitetura, por exemplo, x86-64.
brk()	`int brk(void *addr);`	Troca o local do *program break (PB)*, que define onde o seguimento de dados do processo termina. Incrementar o PB tem o efeito de alocar memória para o processo e decrementá-lo desaloca memória.
clone()	`int clone(int (*fn) (void *), void *stack, int flags, void *arg, ... /* pid_t *parent_tid, void *tls, pid_t *child_tid */ );`	Cria um processo filho.
close()	`int close(int fd);`	Fecha o descritor de arquivo informado.
execve()	`int execve(const char *pathname, char *const argv[], char *const envp[]);`	Executa o programa informado pelo pathname.
exit_group()	`void exit_group(int status);`	Termina todas as threads de um processo. Não somente a thread que fez a chamada.
fstat()	`int fstat(int fd, struct stat *statbuf);`	Retorna informações sobre o arquivo referenciado pelo descritor de arquivo informado.
getpid()	`pid_t getpid(void);`	Retorna o ID do processo que fez a chamada.

Chamada de sistema	Declaração (unistd.h)	Descrição
getrusage()	`int getrusage(int who, struct rusage *usage);`	Retorna informações sobre os recursos utilizados (user cpu, system cpu, memória, trocas de contexto voluntárias e involuntárias, etc.). Pode ser do próprio processo que fez a chamada, de processos filhos ou de alguma outra thread.
lseek()	`off_t lseek(int fd, off_t offset, int whence);`	Reposiciona o ponteiro dentro do arquivo apontado pelo descritor oara leitura ou escrita.
mkdir()	`int mkdir(const char *pathname, mode_t mode);`	Tenta criar um diretório dentro do pathname informado.
mmap()	`void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);`	Mapeia arquivos ou dispositivos na memória.
mprotect()	`int mprotect(void *addr, size_t len, int prot);`	Protege uma região de memória a partir do endereço e tamanho informados.
munmap()	`int munmap(void *addr, size_t length);`	Remove o mapeamentos da memória a partir do endereço e tamanho informados. Acessos subsequentes a esse espaço de endereçamento serão considerados como acessos inválidos de memória.
openat()	`int openat(int dirfd, const char *pathname, int flags); int openat(int dirfd, const char *pathname, int flags, mode_t mode);`	Abre o arquivo especificado relativo ao descritor de arquivo informado ou ao diretório corrente do processo em execução.

Chamada de sistema	Declaração (unistd.h)	Descrição
pread64()	`ssize_t pread(int fd, void *buf, size_t count, off_t offset);`	Lê de um descritor de arquivo a partir do offset informado.
read()	`ssize_t read(int fd, void *buf, size_t count);`	Lê de um descritor de arquivo.
stat()	`int stat(const char *pathname, struct stat *statbuf);`	Retorna informações sobre o arquivo refenciado pelo pathname informado.
wait4()	`pid_t wait4(pid_t pid, int *wstatus, int options, struct rusage *rusage);`	Suspende a execução da thread atual até que o processo filho especificado mude de estado. E quando retorna, traz informações sobre o processo filho.```
write()	`ssize_t write(int fd, const void *buf, size_t count);`	Escreve em um descritor de arquivo.

Fonte: Seção 2 do utilitário *man* do linux (Exemplo: *man 2 wait4*)

Estatísticas Obtidas pela Ferramenta *strace*

As estatísticas a seguir foram obtidas por meio da execução da ferramenta *strace* com a seguinte parametrização:

```
strace -T -ttt -o ../output/strace/saida.out ./arquivo_binario
```

Em seguida, os arquivos de saída foram tratados por meio de um script feito em *Python* e tem o formato que se segue com o tempo de execução de cada uma das chamadas expressos em microssegundos.

Arquivo: ./output/strace/escrever_arquivo.out

Chamada de sistema	Tempo de execução (µs)
execve()	354
brk()	22
arch_prctl()	22
access()	30
openat()	30
fstat()	23
mmap()	26
close()	21
openat()	29
read()	36
pread64()	23
pread64()	22
pread64()	22
fstat()	22
mmap()	24
pread64()	22
pread64()	22
pread64()	22

Chamada de sistema	Tempo de execução (µs)
mmap()	26
mprotect()	28
mmap()	30
mmap()	26
mmap()	27
mmap()	27
close()	23
arch_prctl()	21
mprotect()	30
mprotect()	27
mprotect()	28
munmap()	39
openat()	69
write()	48
fstat()	23
brk()	21
brk()	24
write()	100

Chamada de sistema	Tempo de execução (µs)
close()	29
exit_group()	29
exit_group()	29

Arquivo: ./output/strace/pegar_pid.out

Chamada de sistema	Tempo de execução (µs)
execve()	270
brk()	18
arch_prctl()	17
access()	22
openat()	21
fstat()	17
mmap()	18
close()	16
openat()	20
read()	18
pread64()	17
pread64()	17
pread64()	17

Chamada de sistema	Tempo de execução (µs)
fstat()	16
mmap()	18
pread64()	17
pread64()	16
pread64()	17
mmap()	19
mprotect()	25
mmap()	20
mmap()	19
mmap()	19
mmap()	18
close()	17
arch_prctl()	16
mprotect()	20
mprotect()	18
mprotect()	19
munmap()	24
getpid()	16

Chamada de sistema	Tempo de execução (µs)
fstat()	17
brk()	12
brk()	17
write()	52
exit_group()	52
exit_group()	52

Arquivo: ./output/strace/info_processo.out

Chamada de sistema	Tempo de execução (µs)
execve()	250
brk()	18
arch_prctl()	19
access()	25
openat()	24
fstat()	19
mmap()	21
close()	18
openat()	23
read()	19

Chamada de sistema	Tempo de execução (µs)
pread64()	18
pread64()	18
pread64()	18
fstat()	18
mmap()	20
pread64()	19
pread64()	40
pread64()	19
mmap()	21
mprotect()	25
mmap()	24
mmap()	21
mmap()	22
mmap()	21
close()	18
arch_prctl()	18
mprotect()	24
mprotect()	22

Chamada de sistema	Tempo de execução (µs)
mprotect()	21
munmap()	28
getrusage()	18
fstat()	17
brk()	16
brk()	18
write()	57
exit_group()	57
exit_group()	57

Arquivo: ./output/strace/criar_diretorio.out

Chamada de sistema	Tempo de execução (µs)
execve()	404
brk()	27
arch_prctl()	24
access()	33
openat()	32
fstat()	24
mmap()	28

Chamada de sistema	Tempo de execução (µs)
close()	24
openat()	30
read()	25
pread64()	24
pread64()	25
pread64()	25
fstat()	23
mmap()	26
pread64()	24
pread64()	24
pread64()	24
mmap()	27
mprotect()	33
mmap()	33
mmap()	28
mmap()	29
mmap()	28
close()	23

Chamada de sistema	Tempo de execução (µs)
arch_prctl()	41
mprotect()	31
mprotect()	29
mprotect()	30
munmap()	41
stat()	76
mkdir()	25
exit_group()	25
exit_group()	25

Arquivo: ./output/strace/alocacao_com_malloc.out

Chamada de sistema	Tempo de execução (µs)
execve()	272
brk()	21
arch_prctl()	20
access()	27
openat()	26
fstat()	21
mmap()	23

Chamada de sistema	Tempo de execução (µs)
close()	20
openat()	25
read()	23
pread64()	21
pread64()	20
pread64()	20
fstat()	20
mmap()	21
pread64()	20
pread64()	20
pread64()	20
mmap()	22
mprotect()	27
mmap()	25
mmap()	23
mmap()	24
mmap()	23
close()	20

Chamada de sistema	Tempo de execução (µs)
arch_prctl()	20
mprotect()	103
mprotect()	45
mprotect()	34
munmap()	41
brk()	24
brk()	23
fstat()	25
write()	203
exit_group()	203
exit_group()	203

Arquivo: ./output/strace/processo_filho.out

Chamada de sistema	Tempo de execução (µs)
execve()	311
brk()	45
arch_prctl()	23
access()	29
openat()	28

Chamada de sistema	Tempo de execução (µs)
fstat()	19
mmap()	22
close()	18
openat()	22
read()	20
pread64()	18
pread64()	18
pread64()	18
fstat()	18
mmap()	19
pread64()	18
pread64()	18
pread64()	18
mmap()	20
mprotect()	24
mmap()	24
mmap()	21
mmap()	61

Chamada de sistema	Tempo de execução (µs)
mmap()	27
close()	18
arch_prctl()	18
mprotect()	23
mprotect()	21
mprotect()	22
munmap()	30
clone()	85
getpid()	24
fstat()	24
brk()	21
brk()	23
write()	34
write()	28
wait4()	230
wait4()	230
write()	53
exit_group()	53

Chamada de sistema	Tempo de execução (µs)
exit_group()	53

Arquivo: ./output/strace/ler_entrada.out

Chamada de sistema	Tempo de execução (µs)
execve()	266
brk()	16
arch_prctl()	16
access()	22
openat()	22
fstat()	16
mmap()	18
close()	15
openat()	20
read()	17
pread64()	16
pread64()	16
pread64()	16
fstat()	15
mmap()	18

Chamada de sistema	Tempo de execução (µs)
pread64()	16
pread64()	15
pread64()	16
mmap()	25
mprotect()	24
mmap()	22
mmap()	19
mmap()	19
mmap()	42
close()	16
arch_prctl()	16
mprotect()	20
mprotect()	19
mprotect()	20
munmap()	27
fstat()	17
brk()	15
brk()	17

Chamada de sistema	Tempo de execução (µs)
fstat()	15
write()	73
read()	2079035
write()	263
lseek()	82
exit_group()	82
exit_group()	82

Arquivo: ./output/strace/alocacao_com_calloc.out

Chamada de sistema	Tempo de execução (µs)
execve()	354
brk()	26
arch_prctl()	48
access()	39
openat()	26
fstat()	20
mmap()	22
close()	19
openat()	25

Chamada de sistema	Tempo de execução (µs)
read()	21
pread64()	20
pread64()	19
pread64()	19
fstat()	19
mmap()	21
pread64()	20
pread64()	20
pread64()	20
mmap()	22
mprotect()	25
mmap()	26
mmap()	23
mmap()	24
mmap()	23
close()	19
arch_prctl()	20
mprotect()	25

Chamada de sistema	Tempo de execução (µs)
mprotect()	24
mprotect()	25
munmap()	33
brk()	20
brk()	21
exit_group()	21
exit_group()	21

Arquivo: ./output/strace/redimensionar_alocacao.out

Chamada de sistema	Tempo de execução (µs)
execve()	488
brk()	28
arch_prctl()	27
access()	40
openat()	40
fstat()	29
mmap()	32
close()	27
openat()	36

Chamada de sistema	Tempo de execução (µs)
read()	30
pread64()	28
pread64()	46
pread64()	29
fstat()	27
mmap()	30
pread64()	29
pread64()	27
pread64()	27
mmap()	32
mprotect()	37
mmap()	40
mmap()	34
mmap()	35
mmap()	34
close()	29
arch_prctl()	27
mprotect()	60

Chamada de sistema	Tempo de execução (µs)
mprotect()	22
mprotect()	23
munmap()	30
brk()	18
brk()	19
fstat()	19
write()	25
write()	80
write()	28
write()	27
write()	27
write()	26
write()	27
write()	57
write()	46
write()	27
write()	26
write()	27

Chamada de sistema	Tempo de execução (µs)
write()	25
write()	26
write()	25
write()	58
write()	45
write()	26
write()	26
write()	26
write()	26
write()	26
write()	26
write()	26
write()	25
write()	26
write()	25
write()	26
write()	25
write()	26
write()	23

Chamada de sistema	Tempo de execução (µs)
exit_group()	23
exit_group()	23

Parte 2 - Processos CPU-Bound e I-O bound

Foram desenvolvidos quatro programas que exibam o comportamento esperado para processos CPU-bound e I-O bound e as seguintes tempos foram medidos utilizando-se a ferramenta *time*: 1. Tempo total de execução (segundos) 2. % de uso da CPU 2. Tempo gasto em modo usuário 3. Tempo gasto em modo kernel 3. Trocas de contexto voluntárias 4. Trocas de contexto involuntárias

Estatísticas Obtidas pela Ferramenta */usr/bin/time*

As estatísticas a seguir foram obtidas por meio da execução da ferramenta */usr/bin/time* com a parametrização a seguir e os tempos retornados estão expressos em segundos com arredondamento na segunda casa decimal:

```
/usr/bin/time -f 'Tempo total de execução: %e segundos\nUso de CPU: %P\nTempo
gasto em modo usuário: %U\nTempo gasto em modo kernel: %S\nTrocas de contexto
voluntárias: %w\nTrocas de contexto involuntárias: %c' -o ../output/time/saida.txt
./arquivo_binario
```

Arquivo: ./output/time/manipula_arquivo_time.txt

```
Tempo total de execução: 2.72
Uso de CPU: 99%
Tempo gasto em modo usuário: 0.06
Tempo gasto em modo kernel: 2.63
Trocas de contexto voluntárias: 1
Trocas de contexto involuntárias: 374
```

Em um primeiro momento, o programa *manipula_arquivo.c* foi desenvolvido pensando-se tratar de um programa *io-bound* por manipular arquivos no sistema operacional. Porém, não foi o caso, pois esse programa utilizou quase a totalidade de CPU além da grande número de trocas involuntárias de contexto.

Arquivo: ./output/time/fibonacci_time.txt

```
Tempo total de execução: 36.79
Uso de CPU: 1%
Tempo gasto em modo usuário: 0.32
Tempo gasto em modo kernel: 0.25
Trocas de contexto voluntárias: 564
Trocas de contexto involuntárias: 11470
```

Apesar do baixo consumo de CPU e dos tempos gastos em modo kernel e em modo usuário serem próximos, trata-se de um processo *cpu-bound*, pois o programa *fibonacci.c* apresentou poucas trocas voluntárias em comparação às involuntárias.

Arquivo: ./output/time/multi_matrizes_time.txt

```
Tempo total de execução: 25.86
Uso de CPU: 6%
Tempo gasto em modo usuário: 1.51
Tempo gasto em modo kernel: 0.17
Trocas de contexto voluntárias: 140
Trocas de contexto involuntárias: 8530
```

O programa *multi_matrizes.c* apresentou uma quantidade de trocas voluntárias também consideravelmente inferior às trocas involuntárias. Devido as poucas trocas de contexto voluntárias, seu tempo de execução e sua constante execução com poucas trocas para o modo kernel, pode categorizá-lo como *cpu-bound*.

Arquivo: ./output/time/recebe_entradas_time.txt

```
Tempo total de execução: 10.59
Uso de CPU: 0%
Tempo gasto em modo usuário: 0.00
Tempo gasto em modo kernel: 0.00
Trocas de contexto voluntárias: 4
Trocas de contexto involuntárias: 3
```

O programa *recebe_entradas.c* recebe entradas do teclado do usuário e exibe em tela as saídas. Foram medidos: 0% de utilização da CPU e um tempo de execução de 10.59 segundos, pois se simulou uma demora na digitação pelo usuário. Foram 4 trocas de contexto voluntárias, sendo que são solicitadas 4 entradas do usuário (nome, idade, comida favorita e música favorita) e portanto o programa muda de contexto para esperar a resposta do usuário. Trata-se de um programa tipicamente *io-bound*.