



UNIVERSIDADE ESTADUAL DE SANTA CRUZ

PATRICK SILVA FERRAZ

**FERRAMENTAS PARA GERENCIAR RECURSOS DO
COMPUTADOR**

ILHÉUS - BA
2018

1. Ferramentas para mensurar o tempo de execução de uma função específica dentro de um código.

Biblioteca <time.h> em C/C++

(Possui no CACAU, biblioteca padrão de C)

Descrição:

Nativo de C/C++

tipo da variável: `clock_t`

Uso:

```
main()
{
    clock_t start, end;
    start = clock();
    //CÓDIGO
    end = clock();
    double tempo = (end - start)/(double)CLOCKS_PER_SEC;
}
```

Obs.: Mede o tempo de uso da CPU e não da aplicação, ou seja, não serve para calcular tempo de cada thread em processamento paralelo.

Mais em: <http://www.cplusplus.com/reference/ctime/>

Biblioteca <sys/time.h> em C/C++

(Possui no CACAU)

Descrição:

Somente para sistemas UNIX e seus derivados.

tipo da variável:

```
struct timeval {
    time_t      tv_sec;    /* seconds */
    suseconds_t tv_usec;   /* microseconds */
};

struct timezone {
    int tz_minuteswest;    /* minutes west of Greenwich */
    int tz_dsttime;        /* type of DST correction */
};
```

protótipo da função:

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Uso:

```
main()
{
    struct timeval t_i, t_f;
    gettimeofday(&t_i, NULL);
    //CÓDIGO
    gettimeofday(&t_f, NULL);
```

```

double tf = (double)t_f.tv_usec + ((double)t_f.tv_sec * (1000000.0));
ti = (double)t_i.tv_usec + ((double)t_i.tv_sec * (1000000.0));
}

```

Biblioteca <omp.h> em C/C++

(Possui no CACAU, suportado por padrão pelo gcc [depende da versão])

Descrição:

Necessário compilar com a diretiva *-fopenmp*.

Uso:

```

main()
{
    double start = omp_get_wtime( );
    //CÓDIGO
    double end = omp_get_wtime( );
}

```

2. Ferramentas para mensurar o tempo de execução de uma aplicação

Terminal Linux:

(Possui no CACAU)

time informa o tempo de execução da aplicação ou comando via terminal

Uso:

\$ time ./aplicacao

(Possui no CACAU)

times informa o tempo de execução da aplicação ou comando via terminal, porém não exibe o resultado da execução, mostrando somente o tempo.

Uso:

\$ times ./aplicacao

Gprof

(Possui no CACAU)

Descrição:

Gprof, em engenharia de software, é uma ferramenta para análise dinâmica (diferente da análise estática) da execução de programas escritos em linguagem C, Fortran e Pascal. O propósito usual desse tipo de análise é determinar o quanto de recurso computacional é consumido por cada parte do código, com o objetivo de otimizar o tempo de execução e diminuir quando possível o consumo de memória. Essa ferramenta pode ser usada em conjunto com o GCC. O Gprof é um projeto GNU, sob a licença GNU GPL.

Uso:

Compilar com gcc com a informação para gerar saída ao gprof

Código.:

```
ferraz@nomade:~/Documentos$ cat test.c
#include <stdio.h>

int main(void)
{
    printf("Hello world");
    return 0;
}
```

\$ **gcc -pg test.c** // o gprof utiliza a.out (executável padrão gerado pelo gcc)

// após o comando acima o a.out é gerado

\$ **./a.out** // execute para gerar a saída gmon.out para o gprof

\$ **gprof** // basta executar o gprof na pasta corrente

```
ferraz@nomade:~/Documentos$ gprof
Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated

%   cumulative   self           self       total
time  seconds    seconds   calls   Ts/call  Ts/call  name
%
time      the percentage of the total running time of the
program used by this function.
cumulative a running sum of the number of seconds accounted
seconds    for by this function and those listed above it.
self       the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
listing.
calls      the number of times this function was invoked, if
this function is profiled, else blank.
self       the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
else blank.
total      the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
function is profiled, else blank.
name       the name of the function.  This is the minor sort
for this listing.  The index shows the location of
the function in the gprof listing.  If the index is
in parenthesis it shows where it would appear in
the gprof listing if it were to be printed.

Copyright (C) 2012-2017 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

Alguns testes podem ser realizados sem otimizações, dependendo da finalidade, com as diretivas abaixo:

\$ **gcc -std=c89 -pedantic -Wall -Werror -O0 -pg test.c**

Outros

Pode-se também utilizar as bibliotecas abaixo:

Biblioteca <time.h> em C/C++

Biblioteca <sys/time.h> em C/C++

Biblioteca <omp.h> em C/C++

Apenas colocando as chamadas das funções, dos exemplos em 1, no início e final do código.

3. Ferramenta para monitorar o consumo de memória de uma aplicação

Valgrind

(Possui no CACAU)

Descrição:

Valgrind é um software livre que auxilia o trabalho de depuração de programas criado por Julian Seward. Ele possui ferramentas que detectam erros decorrentes do uso incorreto da memória dinâmica, como por exemplo os vazamentos de memória, alocação e desalocação incorretas e acessos a áreas inválidas.

O diferencial deste programa está no fato de que usa uma máquina virtual para simular o acesso à memória do programa em teste, eliminando a necessidade de uso de outras bibliotecas auxiliares ou mudanças drásticas no código.

Apesar de estar direcionado para programas codificados em C ou C++, a máquina virtual torna possível o uso do Valgrind com programas que foram codificados em outras linguagens, como o Java.

Instalação linux:

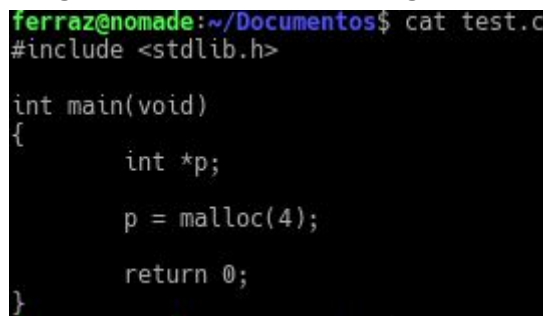
\$ *sudo apt-get install valgrind*

Uso:

\$ *valgrind ./aplicacao*

Obs: Utilizado para verificar memory leak e uso de memória.

Imagem de uma saída do valgrind:

A terminal window with a dark background and light green text. The prompt is 'ferraz@nomade:~/Documentos\$'. The user has entered 'cat test.c'. The content of 'test.c' is displayed: '#include <stdlib.h>', 'int main(void)', '{', ' int *p;', ' p = malloc(4);', ' return 0;', '}'

```
ferraz@nomade:~/Documentos$ cat test.c
#include <stdlib.h>

int main(void)
{
    int *p;

    p = malloc(4);

    return 0;
}
```

```

ferraz@nomade:~/Documentos$ valgrind ./test
==12337== Memcheck, a memory error detector
==12337== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==12337== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==12337== Command: ./test
==12337==
==12337==
==12337== HEAP SUMMARY:
==12337==    in use at exit: 4 bytes in 1 blocks
==12337==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==12337==
==12337== LEAK SUMMARY:
==12337==    definitely lost: 4 bytes in 1 blocks
==12337==    indirectly lost: 0 bytes in 0 blocks
==12337==    possibly lost: 0 bytes in 0 blocks
==12337==    still reachable: 0 bytes in 0 blocks
==12337==    suppressed: 0 bytes in 0 blocks
==12337== Rerun with --leak-check=full to see details of leaked memory
==12337==
==12337== For counts of detected and suppressed errors, rerun with: -v
==12337== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Obs.: Para informações detalhadas do memory leak basta executar com `--leak-check=full`

Splint

(Possui no CACAU)

Verifica o memory leak estaticamente (percorrendo o código fonte)

```

ferraz@nomade:~/Documentos$ cat test.c
#include <stdio.h>

int main(void)
{
    printf("Hello world");
    return 0;
}

ferraz@nomade:~/Documentos$ splint test.c
Splint 3.1.2 --- 01 Dec 2016

Finished checking --- no warnings

```

Comandos nativos do Linux para medir memória

(O SO do CACAU é linux Red Hat, por conta disso, todos os comandos abaixo é possível realizar no CACAU)

\$ **free** // Informa alguns dados sobre a memória disponível e em uso no sistema

```

ferraz@nomade:~/Documentos$ free

```

	total	used	free	shared	buff/cache	available
Mem:	5556472	3240976	532220	726048	1783276	1345304
Swap:	5722108	826648	4895460			

\$ cat /proc/meminfo // detalhes adicionais não mostrados por free

```
ferraz@nomade:~$ cat /proc/meminfo
MemTotal:        5556472 kB
MemFree:         500776 kB
MemAvailable:    1451780 kB
Buffers:         134172 kB
Cached:          1710084 kB
SwapCached:      67608 kB
Active:          3079128 kB
Inactive:        1621656 kB
Active(anon):    2318864 kB
Inactive(anon):  1257836 kB
Active(file):    760264 kB
Inactive(file):  363820 kB
Unevictable:     32 kB
Mlocked:         32 kB
SwapTotal:       5722108 kB
SwapFree:        4900944 kB
Dirty:           192 kB
Writeback:       0 kB
AnonPages:       2840620 kB
Mapped:          623848 kB
Shmem:           720164 kB
Slab:            171356 kB
SReclaimable:    78344 kB
SUnreclaim:     93012 kB
KernelStack:    16000 kB
PageTables:     87932 kB
NFS_Unstable:    0 kB
Bounce:         0 kB
WritebackTmp:    0 kB
CommitLimit:    8500344 kB
Committed_AS:   11404440 kB
VmallocTotal:    34359738367 kB
VmallocUsed:     0 kB
VmallocChunk:    0 kB
HardwareCorrupted: 0 kB
AnonHugePages:   2048 kB
ShmemHugePages:  0 kB
ShmemPmdMapped:  0 kB
HugePages_Total: 0
HugePages_Free:  0
HugePages_Rsvd:  0
HugePages_Surp:  0
Hugepagesize:    2048 kB
DirectMap4k:     438816 kB
DirectMap2M:     5285888 kB
```

Informações de memória

\$ ps

\$ top

Uso do /proc

\$ cat /proc/<pid>/statm

Saída:

tamanho total do programa |

tamanho do conjunto de residentes |

páginas compartilhadas |

texto (código) |

dados / pilha |

biblioteca |

páginas sujas |

\$ cat /proc/<pid>/status

Saída:

Tamanho de Vm: 2772 kB

Vm Lck: 0 kB - ???

Vm RSS: 1624 kB

Dados Vm: 404 kB

Vm Stk: 24 kB

Exe Vm: 608 kB

Vm Lib: 1440 kB

\$ cat /proc/<pid>/maps // mostra as áreas de memória que foram mapeadas

Obs.: O *<pid>* do processo pode ser obtido através do comando **top** informando anteriormente.

4. Ferramenta de profiling de aplicações

SLURM

(Possui no CACAU)

O Slurm Workload Manager (anteriormente conhecido como Simple Linux Utility para Resource Management ou SLURM), ou Slurm, é um agendador de tarefas gratuito e de código aberto para Linux e kernels semelhantes ao Unix, usado por muitos dos supercomputadores e clusters de computadores do mundo. Ele fornece três funções principais. Primeiro, ele atribui acesso exclusivo e/ou não exclusivo a recursos (nós de computador) aos usuários por algum tempo para que eles possam executar o trabalho. Em segundo lugar, ele fornece uma estrutura para iniciar, executar e monitorar o trabalho (normalmente, um trabalho paralelo, como o MPI) em um conjunto de nós alocados. Finalmente, ele arbitra a contenção de recursos gerenciando uma fila de tarefas pendentes.

Comandos:

```
$ sinfo [-p partition_name ou -M cluster_name] // Informações sobre nós e partições
$ squeue --user=nome_de_usuario // verifica o estado dos trabalhos de um usuário
$ srun // executar um comando nos nós de computação alocados
$ snodes [nó do cluster/ estado da partição] // exibir informações do nó
$ fisbatch // lançar um trabalho interativo
$ sranks // listar prioridade de tarefas enfileiradas
$ sueff user-name // obter a eficiência de um trabalho em execução
$ suacct data-inicio user-name // obter informações de contabilidade do SLURM
para tarefas de um usuário desde a data de início até agora
$ slist <jobid> // obter informações sobre contabilidade e nó do SLURM para um
trabalho
$ slogs data-inicio lista-usuarios // obter informações sobre contabilidade de uso de
recursos para as tarefas de um usuário desde a data de início até a data atual
$ stimes // obter as horas de início estimadas para tarefas enfileiradas
$ scontrol [-options] // utilizado para visualizar ou modificar configurações do
SLURM
$ scontrol show job <jobid> // relatório detalhado para o trabalho
$ smap // tela interativa com mapa de processos
$ sacct // histórico dos processos já concluídos e a contabilidade
```

Script modelo:

```
#!/bin/sh
#SBATCH --time=1
/bin/hostname
```

```
$ sbatch script-file // envia um script para execução posterior
```


Parâmetros:

comando	abreviação	significado
--time	-t	tempo máximo de execução ("minutos", "minutos:segundos", "horas:minutos:segundos", "dias-horas")
--cpus-per-task	-c	número de processos por servidor
--nodes	-N	número mínimo de servidores
--ntasks	-n	número total de processos
--output	-o	arquivo com o <i>stdout</i>
--error	-e	arquivo com o <i>stderr</i>

\$ salloc // Alocar nós de computação para uso interativo
\$ scancel <jobid> // Cancelar um trabalho pendente ou em execução

Obs.: SLURM é utilizado em 60% dos computadores do TOP500 (https://en.wikipedia.org/wiki/Slurm_Workload_Manager).

Mais em: <https://slurm.schedmd.com/>

OpenHPC

O OpenHPC é um conjunto de ferramentas FOSS (software gratuito e de código aberto) conduzidas pela comunidade para HPC (computação de alto desempenho) baseada em Linux. O OpenHPC não possui requisitos específicos de hardware.

O OpenHPC fornece uma coleção integrada e testada de componentes de software que, junto com uma distribuição Linux padrão suportada, pode ser usada para implementar um cluster de computação completo.

Enduro / X

Enduro/X é uma plataforma de middleware de código aberto para processamento de transações distribuídas. Ele é construído sobre APIs comprovadas, como XATMI e XA do X/Open. A plataforma é projetada para criar aplicativos baseados em microsserviços em tempo real com opção de clusterização. O Enduro/X funciona como uma alternativa de substituição estendida para o Oracle (R) Tuxedo (R). A plataforma usa filas do Kernel POSIX na memória que garantem uma alta taxa de transferência de comunicação entre processos .

HTCondor

O HTCondor é uma estrutura de software de computação de alto rendimento de código aberto para paralelização distribuída de granulação grosseira de tarefas computacionalmente intensivas. Ele pode ser usado para gerenciar a carga de trabalho em um cluster dedicado de computadores ou para distribuir o trabalho para computadores desktop ociosos - a chamada eliminação de ciclos. O HTCondor é executado nos sistemas operacionais Linux, Unix, Mac OS X, FreeBSD e Microsoft Windows. O HTCondor pode integrar tanto recursos dedicados (clusters montados em rack) quanto máquinas desktop não dedicadas (eliminação de ciclo) em um ambiente de computação.

Moab Cluster Suite

O Moab Cluster Suite é um pacote de gerenciamento de carga de trabalho de cluster, disponível na Adaptive Computing, Inc., que integra o agendamento, gerenciamento, monitoramento e relatório de cargas de trabalho de cluster. O Moab Cluster Suite simplifica e unifica o gerenciamento em um ou vários ambientes de hardware, sistema operacional, armazenamento, rede, licença e gerenciador de recursos.

Outros: OpenLava (GNU), Grid MP (Univa), Plataforma LSF (GNU, substituído por OpenLava), Oracle Grid Engine (Oracle), Xgrid (Apple).

Referências

<https://pt.wikipedia.org/wiki/Valgrind>

<https://pt.wikipedia.org/wiki/Gprof>

https://elinux.org/Runtime_Memory_Measurement

<https://ncc.unesp.br/its/projects/gridunesp/wiki/SLURM>

https://en.wikipedia.org/wiki/Comparison_of_cluster_software

<https://en.wikipedia.org/wiki/OpenHPC>

<https://en.wikipedia.org/wiki/Enduro/X>

<https://en.wikipedia.org/wiki/HTCondor>

https://en.wikipedia.org/wiki/Moab_Cluster_Suite