Measuring Program Resource Use

The GNU `time` Command
Edition 1.7, for version 1.7
12 June 1996

David MacKenzie

Copyright © 1991, 92, 93, 96 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

1 Measuring Program Resource Use

The `time` command runs another program, then displays information about the resources used by that program, collected by the system while the program was running. You can select which information is reported and the format in which it is shown (see Section 1.1 [Setting Format], page 1), or have `time` save the information in a file instead of displaying it on the screen (see Section 1.3 [Redirecting], page 4).

The resources that `time` can report on fall into the general categories of time, memory, and I/O and IPC calls. Some systems do not provide much information about program resource use; `time` reports unavailable information as zero values (see Section 1.5 [Accuracy], page 6).

The format of the `time` command is:

```
time [option...] command [arg...]
```

`time` runs the program *command*, with any given arguments *arg*.... When *command* finishes, `time` displays information about resources used by *command*.

Here is an example of using `time` to measure the time and other resources used by running the program `grep`:

```
eg$ time grep nobody /etc/aliases
nobody:/dev/null
etc-files:nobody
misc-group:nobody
0.07user 0.50system 0:06.69elapsed 8%CPU (0avgtext+489avgdata 324maxresident)k
46inputs+7outputs (43major+251minor)pagefaults 0swaps
```

Mail suggestions and bug reports for GNU `time` to `bug-gnu-utils@prep.ai.mit.edu`. Please include the version of `time`, which you can get by running '`time --version`', and the operating system and C compiler you used.

1.1 Setting the Output Format

`time` uses a *format string* to determine which information to display about the resources used by the command it runs. See Section 1.2 [Format String], page 2, for the interpretation of the format string contents.

You can specify a format string with the command line options listed below. If no format is specified on the command line, but the `TIME` environment variable is set, its value is used as the format string. Otherwise, the default format built into `time` is used:

```
%Uuser %Ssystem %Eelapsed %PCPU (%Xtext+%Ddata %Mmax)k
%Iinputs+%Ooutputs (%Fmajor+%Rminor)pagefaults %Wswaps
```

The command line options to set the format are:

`-f` *format*
`--format=`*format*
: Use *format* as the format string.

`-p`
`--portability`
: Use the following format string, for conformance with POSIX standard 1003.2:

```
real %e
user %U
sys %S
```

`-v`
`--verbose`
: Use the built-in verbose format, which displays each available piece of information on the program's resource use on its own line, with an English description of its meaning.

1.2 The Format String

The *format string* controls the contents of the `time` output. It consists of *resource specifiers* and *escapes*, interspersed with plain text.

A backslash introduces an *escape*, which is translated into a single printing character upon output. The valid escapes are listed below. An invalid escape is output as a question mark followed by a backslash.

`\t`	a tab character
`\n`	a newline
`\\`	a literal backslash

`time` always prints a newline after printing the resource use information, so normally format strings do not end with a newline character (or '`\n`').

A resource specifier consists of a percent sign followed by another character. An invalid resource specifier is output as a question mark followed by the invalid character. Use '`%%`' to output a literal percent sign.

The resource specifiers, which are a superset of those recognized by the `tcsh` builtin `time` command, are listed below. Not all resources are measured by all versions of Unix, so some of the values might be reported as zero (see Section 1.5 [Accuracy], page 6).

1.2.1 Time Resources

`E`	Elapsed real (wall clock) time used by the process, in [hours:]minutes:seconds.
`e`	Elapsed real (wall clock) time used by the process, in seconds.
`S`	Total number of CPU-seconds used by the system on behalf of the process (in kernel mode), in seconds.
`U`	Total number of CPU-seconds that the process used directly (in user mode), in seconds.
`P`	Percentage of the CPU that this job got. This is just user + system times divied by the total running time.

1.2.2 Memory Resources

`M`	Maximum resident set size of the process during its lifetime, in Kilobytes.
`t`	Average resident set size of the process, in Kilobytes.
`K`	Average total (data+stack+text) memory use of the process, in Kilobytes.
`D`	Average size of the process's unshared data area, in Kilobytes.
`p`	Average size of the process's unshared stack, in Kilobytes.
`X`	Average size of the process's shared text, in Kilobytes.
`Z`	System's page size, in bytes. This is a per-system constant, but varies between systems.

1.2.3 I/O Resources

`F`	Number of major, or I/O-requiring, page faults that occurred while the process was running. These are faults where the page has actually migrated out of primary memory.
`R`	Number of minor, or recoverable, page faults. These are pages that are not valid (so they fault) but which have not yet been claimed by other virtual pages. Thus the data in the page is still valid but the system tables must be updated.
`W`	Number of times the process was swapped out of main memory.
`c`	Number of times the process was context-switched involuntarily (because the time slice expired).
`w`	Number of times that the program was context-switched voluntarily, for instance while waiting for an I/O operation to complete.
`I`	Number of file system inputs by the process.
`O`	Number of file system outputs by the process.
`r`	Number of socket messages received by the process.
`s`	Number of socket messages sent by the process.
`k`	Number of signals delivered to the process.

1.2.4 Command Info

`C`	Name and command line arguments of the command being timed.
`x`	Exit status of the command.

1.3 Redirecting Output

By default, `time` writes the resource use statistics to the standard error stream. The options below make it write the statistics to a file instead. Doing this can be useful if the program you're running writes to the standard error or you're running `time` noninteractively or in the background.

`-o` *file*
`--output=`*file*
Write the resource use statistics to *file*. By default, this *overwrites* the file, destroying the file's previous contents.

`-a`
`--append` *Append* the resource use information to the output file instead of overwriting it. This option is only useful with the '`-o`' or '`--output`' option.

1.4 Examples

Run the command '`wc /etc/hosts`' and show the default information:

```
eg$ time wc /etc/hosts
      35     111    1134 /etc/hosts
0.00user 0.01system 0:00.04elapsed 25%CPU (0avgtext+0avgdata 0maxresident)k
1inputs+1outputs (0major+0minor)pagefaults 0swaps
```

Run the command '`ls -Fs`' and show just the user, system, and wall-clock time:

```
eg$ time -f "\t%E real,\t%U user,\t%S sys" ls -Fs
total 16
1 account/      1 db/           1 mail/         1 run/
1 backups/      1 emacs/        1 msgs/         1 rwho/
1 crash/        1 games/        1 preserve/     1 spool/
1 cron/         1 log/          1 quotas/       1 tmp/
        0:00.03 real,   0.00 user,      0.01 sys
```

Edit the file '`.bashrc`' and have `time` append the elapsed time and number of signals to the file '`log`', reading the format string from the environment variable `TIME`:

```
eg$ export TIME="\t%E,\t%k" # If using bash or ksh
eg$ setenv TIME "\t%E,\t%k" # If using csh or tcsh
eg$ time -a -o log emacs .bashrc
eg$ cat log
        0:16.55,        726
```

Run the command '`sleep 4`' and show all of the information about it verbosely:

```
eg$ time -v sleep 4
        Command being timed: "sleep 4"
        User time (seconds): 0.00
        System time (seconds): 0.05
        Percent of CPU this job got: 1%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:04.26
        Average shared text size (kbytes): 36
        Average unshared data size (kbytes): 24
        Average stack size (kbytes): 0
        Average total size (kbytes): 60
        Maximum resident set size (kbytes): 32
        Average resident set size (kbytes): 24
        Major (requiring I/O) page faults: 3
        Minor (reclaiming a frame) page faults: 0
        Voluntary context switches: 11
        Involuntary context switches: 0
        Swaps: 0
        File system inputs: 3
        File system outputs: 1
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 1
        Page size (bytes): 4096
        Exit status: 0
```

1.5 Accuracy

The elapsed time is not collected atomically with the execution of the program; as a result, in bizarre circumstances (if the `time` command gets stopped or swapped out in between when the program being timed exits and when `time` calculates how long it took to run), it could be much larger than the actual execution time.

When the running time of a command is very nearly zero, some values (e.g., the percentage of CPU used) may be reported as either zero (which is wrong) or a question mark.

Most information shown by `time` is derived from the `wait3` system call. The numbers are only as good as those returned by `wait3`. Many systems do not measure all of the resources that `time` can report on; those resources are reported as zero. The systems that measure most or all of the resources are based on 4.2 or 4.3BSD. Later BSD releases use different memory management code that measures fewer resources.

On systems that do not have a `wait3` call that returns status information, the `times` system call is used instead. It provides much less information than `wait3`, so on those systems `time` reports most of the resources as zero.

The '`%I`' and '`%O`' values are allegedly only "real" input and output and do not include those supplied by caching devices. The meaning of "real" I/O reported by '`%I`' and '`%O`' may be muddled for workstations, especially diskless ones.

1.6 Running the `time` Command

The format of the `time` command is:

`time` [`option`...] *command* [*arg*...]

`time` runs the program *command*, with any given arguments *arg*.... When *command* finishes, `time` displays information about resources used by *command* (on the standard error output, by default). If *command* exits with non-zero status or is terminated by a signal, `time` displays a warning message and the exit status or signal number.

Options to `time` must appear on the command line before *command*. Anything on the command line after *command* is passed as arguments to *command*.

`-o` *file*
`--output=`*file*
: Write the resource use statistics to *file*.

`-a`
`--append`
: *Append* the resource use information to the output file instead of overwriting it.

`-f` *format*
`--format=`*format*
: Use *format* as the format string.

`--help`
: Print a summary of the command line options to `time` and exit.

`-p`
`--portability`
: Use the POSIX format.

`-v`
`--verbose`
: Use the built-in verbose format.

`-V`
`--version`
: Print the version number of `time` and exit.

Table of Contents