NAME

hs - hierarchical scheduler

SYNOPSIS

```
hs
  [ -d | --debug          ]  [ -f | --freeze-core                ]
  [ -h | --hard           ]
  [ -l | --log            ]  [ -m | --memory-budget-add-on  TERM ]
  [ -p | --poll-llc       ]
  [ -P | --fork-processes  ]
  [ -q | --quiet          ]  [ -Q | --really-quiet               ]
  [ -r | --run            ]  [ -s | --system  TASK-SYSTEM-FILE   ]
  [ -v | --verbose        ]  [ -w | --wait-for-forked-processes  ]
```

DESCRIPTION

hs is a hierarchical scheduler. It executes either hard-coded software, or forked processes, according to a polled-preemptive global EDF algorithm acting on the real-time parameters of the sporadic task model.

OPTIONS

-d, **--debug**

Output various hard-coded debug information.

-f, **--freeze-core**

Do freeze the best-effort core when it exceeds its DRAM budget. To do this, **perf_event_open(2)** is used along with a Linux cgroup. The use of *-f* implies *--poll-llc* because that is how DRAM fetches are booked. **Note:** For this to work, either run hs with ’sudo’; or, set the owner of hs to root, and then set the SUID bit; or, do something else that amounts to the same.

-h, **--hard**

Exit the scheduler with error code -1 immediately if a task is delayed.

-l, **--log** Log the time in nanoseconds at every tick to the file *tick_times.log* in the same directory as hs.

-m, **--memory-budget-add-on***TERM*

Add TERM to all memory budgets.

-p, **--poll-llc**

Every tick, poll the DRAM last-level-cache (LLC) to find out how many non-cached DRAM accesses the best-effort core has made.

-P, **--fork-processes**

Don’t use hard-coded mock software; fork processes. This means the system file must consist of commands (including their arguments) that are executable on the underlying system. E.g., to do the equivalent of *echo hello fool* put */bin/echo(hello fool)* in the system file. (At this point hs cannot mix mock software and real processes; and, absolute paths to executables are required.) The easiest way to use this is to put all commands in a script, and then use that script in the system file. (When freezing and thawing, the process group id (PGID) is used, as to affect offsprings of the script as well.)

-q, **--quiet**
Don't output task state transitions. Hard-coded task software should typically be quiet as well although that has to be coded explicitly in hs.

-Q, **--really-quiet**
As **--quiet** only shut up hard-coded task software as well.

-r, **--run**
Run the system when it is loaded without confirmation. (Sometimes though it is useful to have the system only loaded, not executed, to be triggered exactly when needed.)

-s, **--system***TASK-SYSTEM-FILE*
Load the system from the specified file. Creating systems interactively is just fun and games: it is much better to exclusively use files. Use this with **--run** to execute a system from a file.

-v, **--verbose**
Every tick, output the state of the entire system.

-w, **--wait-for-forked-processes**
At the end of the execution of hs, *wait(2)* for all forked processes to terminate. Use with care: with non-terminating processes this makes hs non-terminating as well. This option overrides the "Global lifetime" parameter as long as there are children left. **-w** implies **--fork-processes** because otherwise there are none to wait for.

TASK SYSTEM

A task system is defined in a task-system text file. There are a couple of examples in *./hs-linux/sys* - otherwise, run hs interactively to see how a system is expressed, then put the exact same in a text file. If need be, later modify the selfsame text file to fine-tune the system, rather that creating one anew interactively.

DOCUMENTATION AND CREDITS

There is an ambitious PDF document that describes this project: *./hs-linux/docs/report.pdf*

QUESTIONS AND FEEDBACK

Written by Emanuel Berg <*embe8573@student.uu.se*> for Uppsala University, 2014.

SEE ALSO

fork(2), **signal**(2), **wait**(2), **perf_event_open**(2)