superhelperschedulerspy

with positional argument of an executable program, runs offline analysis; without, runs online analysis
--length={15s,Forever} # logs program for given length of time, defaults to
--quiet # doesn't write to log file in pwd

online mode

- resource monitor of sorts
    - how to differentiate from top, sar?
    - https://www.redhat.com/magazine/011sep05/features/tools/#sar <- nice
    - top is more primitive, sar is a little more helpful but only measures physical devices and dumb stuff like page faults
- graphical program window (maybe ASCII graphs)
- graphs of currently running processes and how many timeslices they've ran for, how many voluntary/involuntary ctx switches, group by scheduling class, average time run in timeslice

offline mode

- forensic analysis (for given period of time)
- can view state of rb tree
    - only mildly cool on its own, needs more
- race condition finder
    - use scheduler internals to determine race conditions between threads
- flag for length of querying
- primitives + pthread variables:
    - expose a variable/remove variable (this was anna, she was here, and wrote this thing, which is now here, on this page. here HI THIS IS ANNA)
        - record when the variable is changed/accessed (and which thread)
        - record value (only usable for primitives)
    - expose/remove mutex
        - see which threads are waiting on mutex
        - see which thread currently holds mutex
    - expose/remove cond var
        - see which threads are waiting on cond var
        - see which thread signals/broadcasts cond var
        - see which thread is woken up from a given signal

- if we're adding context switch code, we need to make our timeslices longer to compensate
- start_logging function
- have per-cpu buffer of mutex values, primitives, and cond vars and record values at end of timeslices
- have kthread reading from those buffers and outputting to proc
- want timestamp on every report which comes at conclusion of every timeslice
    - need to use clock_gettime, unless this blocks in which case we won't

ARCHITECTURE:

run in online mode
it will write a 1 to a node in /proc and will begin reading from the logs there

run in offline mode
it will fork a process. this process will have #included our header library and registered mutexes, cond vars, and primitive variables, and called start_logging, which will create ring buffers (run in offline mode, one per cpu) and fill them every ctx switch (with information like which thread changed which tracked variables), and create a proc node in the child process's /proc. whenever a read from the /proc node is instigated, it will flush the ring buffers from cpus appropriately. each element in the ring buffers will be:

- a timestamp
- values of all exposed variables, mutexes

the context switcher will record the values of all exposed variables and other stuff every context switch and add them to the ring buffer. the context switcheroo will check if there is space in the ring buffer and if not, just don't log anything (hopefully will never happen). every time a read is instigated, values are popped off end of the ring buffer.

problems/questions:

- how to read variable values every context switch?
    - can do in syscall, because translates virtual addresses, but this does not happen in context switcher, i think
- if we are going to record both read and write, how to implement this?
    - DDL?
    - kernel support?
- how to deal with threads running truly concurrently (at the exact same time, on different cpus)?
    - it won't detect that variables have been changed - how to track if during a timeslice a variable was read/written if at the end the value is exactly the same