ECE 252: Systems Programming and Concurrency Spring 2019

Lecture 17 — Hogwarts School of Witchcraft and Wizardry

Jeff Zarnett *2018-09-18*

In Class Exercise: Harry Potter and the pthread House Elves

Background. At Hogwarts School of Witchcraft and Wizardry, undesirable tasks are done by House Elves, magical creatures who apparently are not covered under any sort of employment standards or workers rights legislation. They work as a team under the direction of Dobby, who is a free elf. At the beginning of the day, the headmaster, Dumbledore, tells Dobby how many tasks there are to do today. Dobby will communicate with Dumbledore about what the specific tasks to do are, and posts a partial list of items for the other elves to do. The elves take tasks from the list and do them. When the list is empty, Dobby asks for more directions and posts more tasks. This continues until the house elves have finished all the tasks for the day.

Primary Objective. The primary objective of this exercise is to practice using synchronization constructs such as semaphores and mutexes in a simple C program.

Secondary Objective(s). You will gain some experience in designing and debugging a multithreaded program. You may also improve your ability to work with version control (git) and gitlab.

Starter Code. The starter code can be found at `https://git.uwaterloo.ca/jzarnett/ece252/ece252-e2` – fork this repository to your own space. Set permissions for this repository to be private, but add the group for the course staff with read access so your code can be evaluated.

Submitting Your Code. When you're done, use the command `git commit -a` to add all files to your commit and enter a commit message. Then `git push` to upload your changes to your repository. You can commit and push as many times as you like; we'll look at whatever was last pushed. And check that you gave the course staff permissions!

Grading. Binary grading: 1 if you have made a meaningful attempt at implementing the code; 0 otherwise.

Description of Behaviour. The goal is to implement the program so that the following behaviour occurs:

- At program start, the first argument provided to the program represents the total number of tasks to do today. So if, for example, the program is invoked with `./dobby 100`, there are 100 tasks to be done today.
- The `init` and `cleanup` functions should initialize and clean up, respectively, any global variables.
- Your program should create five (5) house elves (`house_elf()`) and one (1) Dobby thread (`dobby()`).
- Worker elves take tasks using the function `take_task()` (which cannot be safely run in parallel) and then do the work by calling `do_work()` (which can be done in parallel). If the list of tasks is empty, an elf cannot call `take_task()` and must instead wake up Dobby, who will then post the next 10 tasks before going to sleep again; meanwhile the elf is blocked until the next group of tasks is ready.
- When woken up, Dobby posts (up to) 10 tasks to do by calling `post_tasks()` and updates the number of available tasks. After that, Dobby sleeps (is blocked). If there are fewer than 10 tasks remaining for the day then Dobby cannot post 10, but instead posts however many remain.
- Note that Dobby and the elves are responsible for managing the count of tasks available.

- House elves don't know that their day is over until they are told so using `pthread_cancel`. Dobby must dismiss the other elves using this function, if Dobby gets woken up and the remaining tasks for the day is zero. After cancelling the other threads, Dobby should use `pthread_join` to wait for each of them to finish before he can go home for the day (exit).
- Your program should not leak memory; be sure to destroy/deallocate anything initialized/allocated.
- There should not be any race conditions in your program either.

Hints & Debugging Guidance. This problem is very similar to what is called the "Dining Savages Problem" [Dow08], but is probably less offensive. Because who does not like Harry Potter themed material? Also, I really hope J. K. Rowling does not sue me. I don't have time for that...

Some general guidance is below. If you're having trouble, try running through these steps and it may resolve your problem. If you're still stuck you can ask a neighbour or the course staff.

- Check the documentation for how functions work if you are unfamiliar with them (google is your friend!)
- Have you initialized all variables? It is easy to forget; `malloc` does not initialize the value...
- If a variable is going to be used in more than one function, make sure it is allocated on the heap (ie with `malloc`) and not on the stack.
- Is there a missing or extra $*$ (dereference) on a pointer somewhere?
- Check carefully what values semaphores are initialized to.
- Is the number of `wait` and `post` statements on a given semaphore balanced?
- Does every memory allocation have a matching deallocation?
- Does every code path in a critical section protected by a mutex lead to an unlock statement?
- Are accesses to shared variables protected by a mutex or semaphore appropriately?
- Remember that both reads and writes of shared variables need to be inside critical sections.
- It may be helpful to put `printf()` statements to follow along what the program is doing and it may help you narrow down where the issue is.
- Don't be shy about asking for help; the TAs and instructor are here to help you get it done and will help you as much as is reasonable.

References

[Dow08] Allen B. Downey. *The Little Book of Semaphores (2nd Edition)*. Green Tea Press, 2008.