CS 367 Project 1 - Fall 2024:
StrawHat User Manual

1 Introduction

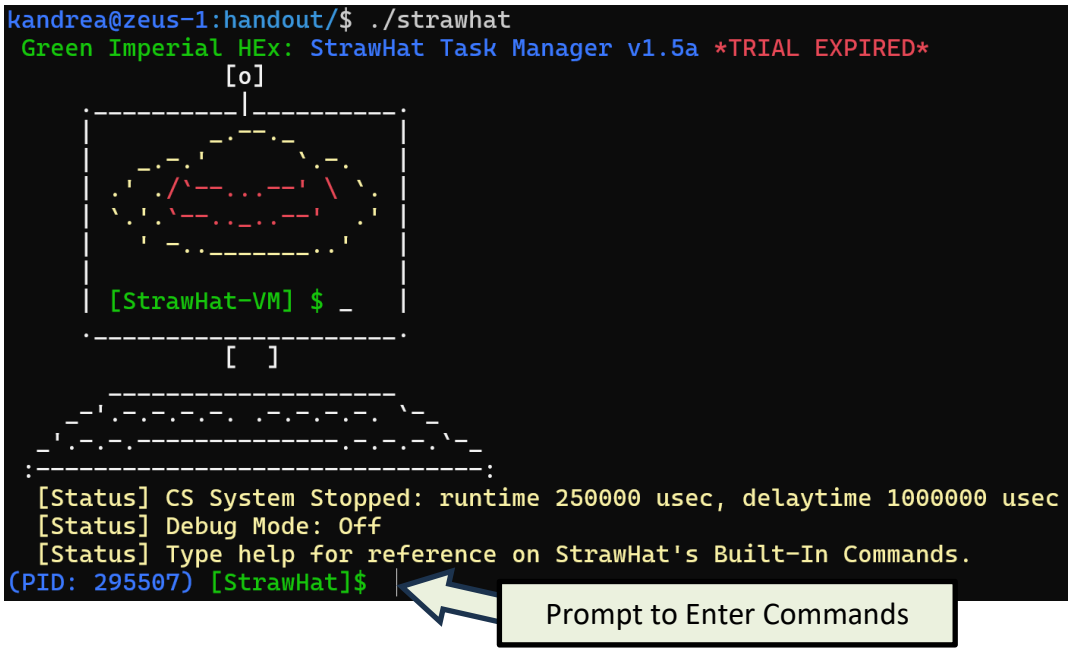
Your code in `egul_sched.c` forms one library in the StrawHat Task Manager (TM). You can test your code entirely separately from StrawHat (see P1_Self_Testing.pdf), but if you want to test the code running on the main TM, this manual will help you understand the various commands to use when running the StrawHat TM (the `strawhat` executable that is generated when you run make).

1.1 StrawHat Details *(Big-Picture Overview)*

StrawHat is a shell that you can use to run normal non-interactive Linux commands like **ls**, **cal**, and **clear** with a custom CPU Scheduler. Programs continually switch back and forth to make it seem that they are all running at once. This is called **Multitasking** in the Operating system. Normally you have no control of when these switches happen.

StrawHat is a shell that runs a special scheduler that will do the switching in a custom manner, which allows program developers to test how their programs will interact with other programs or allows OS developers to test scheduling algorithms to see which performs better.

Like any other shell, StrawHat can run programs with arguments (like **ls -al** or **wc foo.txt**) but it cannot deal with any commands with an interactive interface – you cannot use it run **wc** without arguments, or to run **vim**, for example. Only programs that do not need user input can be run.



When StrawHat starts, you will get a prompt just like it was a Shell in Linux. You can enter two types of commands here: **Built-In StrawHat Commands** or **Programs**.

Built-In Commands are used to control StrawHat (such as '**quit' or 'start**').

Programs are normal programs that you can run from either the current directory (like `slow_door`) or common Linux programs from the /usr/bin directory (like `cal` or `ls`).

You can write any program you want to run in StrawHat for testing.

When you start StrawHat, the main execution loop, which is called the **Context Switch (CS)** system, is not running. This is a nice feature that we're going to abuse for our own debugging as well. To start the CS system that will be looping and controlling the programs to be run on the CPU, use the built-in command **start**. You can stop it at any time with **stop**.

The CS system **calls your functions** in this sequence: *(This is the main StrawHat Logic)*

On Startup of StrawHat

- `egul_initialize` Create a new Scheduler and the three Queues

Command Entry: These are called when a new program to run is entered into the prompt.

- `egul_create` Create a new Process Struct and Initialize its Values
- `egul_insert` Inserts that new Process Struct into the Ready Queue

CS Engine Main Loop (active when the CS system is running - start, and paused when not - stop)

- `egul_select` Chooses the Process to Run Next from the Ready Queue
- < At this point if any process was returned by select, we run the selected process for 250ms >
- If the process did not exit during this run...
    a. `egul_insert` Insert this Process Struct back into the Ready Queue
- else, if the process did exit during this run...
    a. `egul_exited` Insert this Process Struct into the Terminated Queue

CS Engine Shutdown

- `egul_deallocate` Deallocate all Allocated Memory.

StrawHat Command-Based Calls (called by the appropriate built-in commands listed on the right)

- `egul_killed` Move a Process from Ready/Suspended into Terminated [kill]
- `egul_suspend` Move a Process Struct from Ready to Suspended [suspend]
- `egul_resume` Move a Process Struct from Suspended to Ready [resume]
- `egul_count` Returns the number of Processes in a given Queue [schedule]
- `egul_get_ec` Returns the exit code of a Terminated Process [schedule]

StrawHat will choose a process to run, then it will run it for a little while (default is 250ms), then it will return the process and repeat those above steps until all processes have finished. The selection algorithm is detailed in the main Project Documentation.

When we run processes, they will cycle in a **priority** manner. Lower priority processes will run less frequently. If a process is **critical** it will run immediately. If a process is **starving**, then it will get to run once and then it'll go back to waiting like normal in the Ready queue again until it's either the highest priority process remaining, or until it becomes starving again.

For very simple processes, like '`cal`', those programs will finish in a single execution on the CPU. For something that runs for a long time, like '`slow_cooker`', you will see it run for a long time on its own, so it will keep getting selected to run on the CPU over and over until it's finished.

This, of course, also only works once your code is written.

1.2 StrawHat Function Pointers

StrawHat is using your Egul Scheduler library, however, if you look through the StrawHat code, you will not find any direct calls to your functions! The reason is that StrawHat can choose any scheduler library for its implementation. It is not hardcoded to just use yours, so how does it call your functions if it doesn't know which library it will be using?

The way we solve this problem is using Function Pointers in C! Just like you saw with using the **qsort** C function and you had to pass in the name of the comparison function to use, StrawHat uses its own function pointers to call functions using its own names! There is a file in StrawHat's code that does provide a mapping by creating those function pointers in the first place.

So, if you want to look through the StrawHat code, here is the list of function pointers you will see it call and which of your functions they refer to:

StrawHat's Function Pointer...	Will Call Your Egul's Function...
`sched.scheduler_initialize`	`egul_initialize`
`sched.scheduler_create`	`egul_create`
`sched.scheduler_enqueue`	`egul_insert`
`sched.scheduler_get_count`	`egul_count`
`sched.scheduler_select`	`egul_select`
`sched.scheduler_suspend`	`egul_suspend`
`sched.scheduler_resume`	`egul_resume`
`sched.scheduler_exited`	`egul_exited`
`sched.scheduler_killed`	`egul_killed`
`sched.scheduler_get_ec`	`egul_get_ec`
`sched.scheduler_cleanup`	`egul_deallocate`

So, `sched.scheduler_select()` in StrawHat's code will actually directly call `egul_select()` in your code.

2 Notes on Execution

When a process is scheduled to run, it will be executed on the CPU for 250ms. Some processes, however, take longer than this before they start printing out their first output! So, you may have to wait for it to be scheduled a few times before it prints out results. You will still see whenever a new process is selected to run in the output, even if the program itself doesn't print anything out.

```
...
[PID: 410368] slow_countup counting up: 1 ...
[PID: 410368] slow_countup counting up: 2 ...
stop
  [Status] Stopping the CS System
(PID: 410356) [StrawHat]$  slow_door
  [Status] Process slow_door created with PID 410440
(PID: 410356) [StrawHat]$  start
  [Status] Starting CS System: 250000 usec Run, 1000000 usec Between
(PID: 410356) [StrawHat]$  [PID: 410368] slow_countup counting up: 3 ...
  [Status] Switching to run PID: 410440 (slow_door)
  [Status] Switching to run PID: 410368 (slow_countup 100)
[PID: 410368] slow_countup counting up: 4 ...
  [Status] Switching to run PID: 410440 (slow_door)
[PID: 410440]                     . .:.:.:.:. .:\     /:. .:.:.:.:. ,
  [Status] Switching to run PID: 410368 (slow_countup 100)
[PID: 410368] slow_countup counting up: 5 ...
...
```

This output sample has slow_countup running as the only process. I type in the built-in command "stop" to stop the CS engine. Then I entered the command "slow_door" to create that program. Finally, I type the built-in command "start" to start the CS engine. At this point, slow_countup is at the front of the Ready Queue and slow_door is behind it. Both have the same priority and none are starving, so slow_door is the first one selected. It runs and prints out the count of 3.

At this point the CS engine hits its 250ms timeout and switches to the next process. Your **`egul_select`** function now returned slow_door, so StrawHat prints the message that slow_door is now running.

```
[Status] Switching to run PID: 410440 (slow_door)
```

This actually runs slow_door for 250ms, but the problem is that it takes so long to start running a new process, that in these 250ms nothing gets output! So, it times out and gets switched. Now your **`egul_select`** function returns slow_countup again to continue running.

```
  [Status] Switching to run PID: 410368 (slow_countup 100)
[PID: 410368] slow_countup counting up: 4 ...
```

After 250ms more, then StrawHat returns this process to the Ready Queue using **`egul_insert`** and then calls **`egul_select`** again. Now your function returns slow_door once again and StrawHat prints out the message saying it is switching to that. At this point, slow_door continues running for another 250ms from where it left off and this time it's ready to print out its first line of output.

```
  [Status] Switching to run PID: 410440 (slow_door)
[PID: 410440]                    . .:.:.:.:. .:\     /:. .:.:.:.:. ,
```

So, it is actually normal if you don't see output on the first time any process is selected and run!

If you want to make sure the scheduling is happening properly, you can also use the **debug** option:

```
...
  [DEBUG ] Context Switch: Iteration 1
  [DEBUG ] Schedule Select Returned PID 411332
  [Status] Switching to run PID: 411332 (slow_countup 100)
  [DEBUG ] PID: 411332 has been CONTINUED.
  [DEBUG ] PID: 411332 has been STOPPED.
  [DEBUG ] Context Switch: Iteration 2
  [DEBUG ] Schedule Select Returned PID 411332
  [DEBUG ] PID: 411332 has been CONTINUED.
[PID: 411332] slow_countup counting up: 0 ...
  [DEBUG ] PID: 411332 has been STOPPED.
  [DEBUG ] Context Switch: Iteration 3
...
```

You can see that when we turn the debug mode on with the **debug** built-in command, it activates a very large amount of extra debug output from StrawHat.

In this screenshot here, we started the engine and **slow_countup** was selected (returned from `egul_select`) to run. We see this happening in Iteration 1. When debug says it was CONTINUED, it means the process is being run on the CPU. When debug says it is STOPPED, it means it is being taken off the CPU. The 250ms run didn't produce any output, but debug mode let us see that it was actually running properly. Now in the second Iteration of the CS engine, we see `egul_select` again picked the same process and runs it. Now it's in its second 250ms run and we do finally see output.

2.1 Notes on Missing Output or Extra Output on the same

One more note is that there are multiple processes running at the same time and they're all using the same screen, so sometimes text appears to be cut-off, or on multiple lines, or without a prompt. This is all normal! It's just because many programs are sharing the same screen.

```
(PID: 411330) [StrawHat]$    [DEBUG ] Context Switch: Iteration 1
sto[PID: 411332] slow_countup counting up: 0 ...
p
  [Status] Stopping the CS System
(PID: 411330) [StrawHat]$
```

This is expected and random. You can also see here that I was typing in the built-in command "stop" while slow_countup was printing, so my command looks like it was chopped up on the screen. This is

also fine! It just looks like this because you're typing between outputs. When I pressed enter, the "stop" built-in runs and you can see it stopped the CS system properly.

Sometimes you may have a missing prompt as well. You can just press enter at any time to get that back. The reason the prompt is missing is that it probably printed earlier and was followed by a lot of program output messages. You can press enter at any time to get a new prompt.

Press ENTER at any time to get a clean prompt.

3 Building and Running StrawHat (./strawhat)

You will receive the file **project1_handout_v1_0.tar**, which will create a handout folder on Zeus.

```
kandrea@zeus-1:handout$ tar -xvf project1_handout_v1_0.tar
```

In the handout folder, you will have three directories (**src, inc,** and **obj**). The source files are all in **src/**. The one you're working with is **egul_sched.c**, which is the only file you will be modifying and submitting. You will also have very useful header files (**egul_sched.h**) along with other files for the simulator itself in the **inc/** directory.

3.1 Building StrawHat

To build StrawHat, run the **make** command:

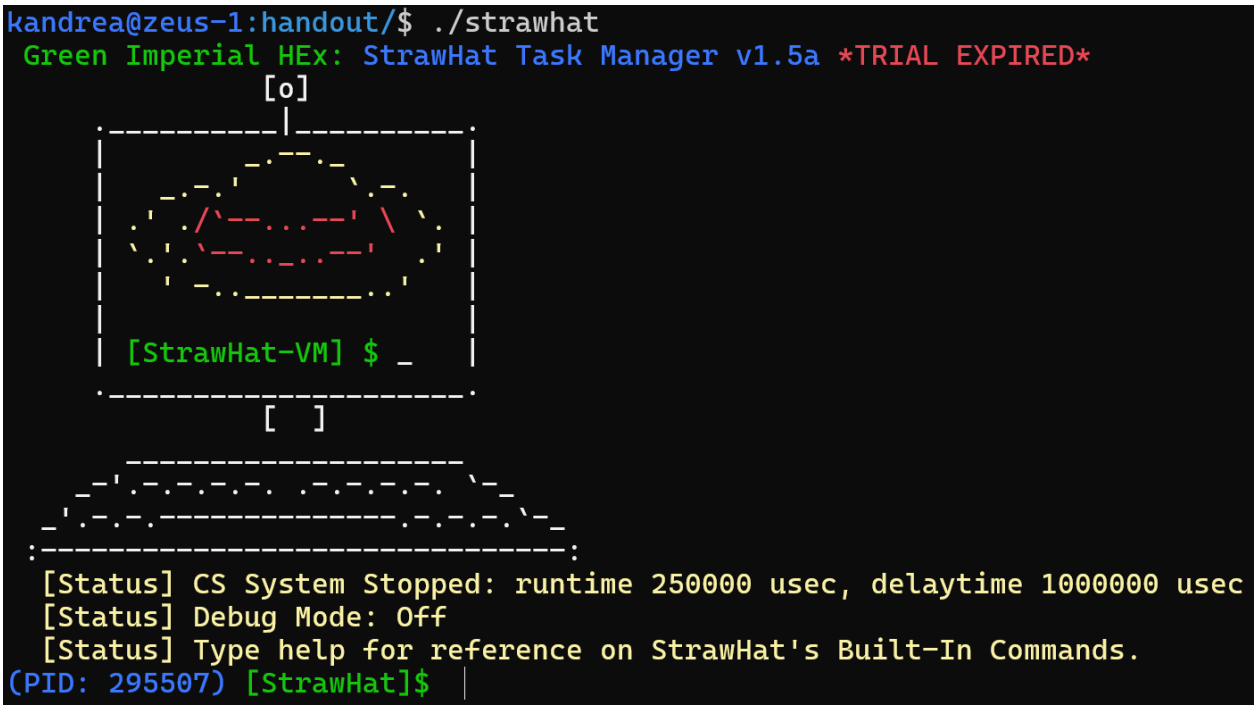
```
kandrea@zeus-1:handout$ make
gcc -std=gnu99  -Og -Wall -Werror -Wno-error=unused-variable -Wno-error=unused-function -
pthread -I./inc -L./obj -g  -c -o obj/strawhat.o src/strawhat.c
gcc -std=gnu99  -Og -Wall -Werror -Wno-error=unused-variable -Wno-error=unused-function -
...
kandrea@zeus-1:handout$
```

This may be the first time you are building a large project. Your code is just one small piece.

3.2 Running StrawHat

To start StrawHat, run **./strawhat**

This will give you the shell interface for the TM, which you can use to enter your commands into.



At the prompt, you can either type in the name of a program you want to run, or you can type in one of the StrawHat commands to control the TM.

All of the StrawHat Commands are detailed after the function details of what you will be writing.

You can also type in **help** at any time for a list of all StrawHat Commands.

4 StrawHat Manual

StrawHat is a full-featured Task Managing Shell that lets you run normal Linux commands and programs, but with a custom scheduler.

The basic idea is that when the Context Switch (CS) engine is running, it will call `egul_select` to get a process to run from your code, then it will run that process for a **runtime** number of microseconds (usec). The CS engine will then stop it from running on the CPU and return it (`egul_insert` or `egul_exited`). Finally, it waits for **delaytime** microseconds before getting the next process.

In a real live environment, **runtime** would be about 3000usec (3ms) and **delaytime** would be 0usec. But we want to be able to follow the action and make it easier to debug, so the default values for this system in our class is **runtime** is set to 250000usec (0.25 sec) and **delaytime** is 1000000usec (1 sec).

StrawHat starts up with all of its internal debug messages turned off and with the CS engine off. Because of this, when you start it up, you can type in commands to run and nothing will happen! To make them actually run, you need to start the CS engine.

Built-In Commands to Control the CS Engine

- **help** Prints out this reference.
- **start** This will start the CS Engine running.
  - Pressing ctrl-C will toggle the CS Engine as well.
- **stop** This will stop the CS Engine running. (You can start and stop it as much as you want!)
  - Pressing ctrl-C will toggle the CS Engine as well.
- **status** This will print out a status message about the CS Engine settings and if it's running.
- **schedule** This will print out the status of all three Queues.
- **kill X** This will terminate a process from your Queues with PID X.
- **suspend X** Suspends a process with PID X
  - You can type **suspend** without a PID to suspend the first process in the Ready Queue
- **resume X** **Resumes** a process with PID X
  - You can type **suspend** without a PID to resume the first process in the Suspended Queue
- **debug** Toggles extra debug messages.
- **runtime X** This will change the **runtime** to a new usec value. (Default 250000 usec)
  - If you type **runtime** without a new value, you'll get a warning, but can see the current value.
- **delaytime X** This will change the **delaytime** to a new usec value. (Default 1000000 usec)
  - If you type **delaytime** without a new value, you'll get a warning, but can see the current value.

You can also press Enter at any time to see the prompt if it is overwritten by program output.

- This is a very interesting concept! We have multiple programs running and writing to the screen at the same time, so you may see output from programs interrupting the command you're typing.
  - This doesn't affect your command at all, you can keep typing and hit enter.

```
  [Status] CS System Stopped: runtime 250000 usec, delaytime 1000000 usec
  [Status] Debug Mode: Off
(PID: 178230) [StrawHat]$  runtime 500000
  [Status] Setting CS System: runtime 500000 usec, delaytime 1000000 usec
(PID: 178230) [StrawHat]$  delaytime 2000000
  [Status] Setting CS System: runtime 500000 usec, delaytime 2000000 usec
(PID: 178230) [StrawHat]$  status
  [Status] CS System Stopped: runtime 500000 usec, delaytime 2000000 usec
(PID: 178230) [StrawHat]$  start
  [Status] Starting the CS Execution
(PID: 178230) [StrawHat]$
```

Before you have written anything for your 11 **egul_sched.c** functions, it will just exit. For the CS Engine to do anything, it needs to get Processes from your code. You will need to implement **egul_initialize**, **egul_create**, **egul_insert**, **egul_select**, and **egul_get_count**, at a minimum, for the system to begin processing.

The **Shell** component of StrawHat lets you run programs just like Linux. When you enter the program names and arguments, the shell will call your functions to create and add those programs to your Linked Lists. When the CS Engine is running, you'll see the output of those programs, and when the CS Engine is stopped, you'll see nothing because nothing is running.

Built-In Commands for the StrawHat Shell

- **schedule** This will print out all of the processes in all three of your Linked Lists!
- [Linux Command] You can enter any common Linux Command with arguments to run.
  - Example: **ls -al**
- [Local Command] Note, there are four special programs you can run that have very long outputs.
  - These are a lot easier to debug because you can see them being run over a long time.
  - **slow_countup [X]** Prints out one message per iteration for 10 (or X) iterations.
    - **Exits with exit code X**
  - **slow_countdown [X]** Prints out one message per iteration for 10 (or X) iterations.
    - **Exits with exit code 0**
  - **slow_bug** Prints out an ASCII art of a Bug Hunting Knight, one line per iteration.
    - **Exits with exit code 0**
  - **slow_door** Prints out an ASCII art of the Shire, one line per iteration.
    - **Exits with exit code 0**
- **debug** Toggles the StrawHat Debug Messages On/Off (can be spammy)
- **quit** Shuts EVERYTHING down responsibly and quits. (calls your egul_deallocate)

Special Options for Running Processes in the StrawHat Shell (Use these AFTER the Name and Arguments)

- **-p X** Run the process at Priority Level X. (Without this option, processes default to 128)
- **-c** Run the process with Critical Priority. (Always runs first to completion)

Here is an example of the Special Options to run slow_cooker at Priority 100, with command line argument 5, to run slow_bug as a Critical process, and finally to run slow_door as a default (Priority 128) level process.

```
(PID: 412756) [StrawHat]$  slow_cooker 5 -p 100
  [Status] Process slow_cooker 5 -p 100 created with PID 412758
(PID: 412756) [StrawHat]$  slow_bug -c
  [Status] Process slow_bug -c created with PID 412772
(PID: 412756) [StrawHat]$  slow_door
  [Status] Process slow_door created with PID 412773
```

You may also edit the top portion of `inc/strawhat_settings.h` header. This has all of the initial default settings for running StrawHat in it. Most of this is changeable with the above commands, however, this lets you disable the colors (change to **#define USE_COLORS 0**) if you wish. Do not modify any code below the line that says do not modify anything below this line. Once changed simply run '**make**' again as normal.

4.1 Notes on Concurrency and Visual Interruptions

You can type in new commands to run while the CS Engine is running, but you may notice that your typing gets interrupted anytime a process outputs its text to the screen! This is because this is a multi-tasking Task Manager. Don't worry, the text you were typing is there so you can keep typing and hit enter and it'll still work.

If you ever want to see the prompt again, you can always just hit Enter without typing anything.

If you are having a lot of trouble with the CS Engine interrupting you, you can use Control-C to toggle the CS Engine on and off as well. (This is more of a fall-back option and doesn't play well with GDB).

5 StrawHat Sample Runs

Here is an example run. We added line numbers on the left to help with the explanation below.

```
1  (PID: 542270) [StrawHat]$  slow_countdown
2    [Status] Process slow_countdown  created with PID 542331
3  (PID: 542270) [StrawHat]$  ls
4    [Status] Process ls created with PID 542372
5  (PID: 542270) [StrawHat]$  slow_countup 5 -p 200
6    [Status] Process slow_countup 5 -p 200 created with PID 542471
7  (PID: 542270) [StrawHat]$  start
8    [Status] Starting CS System: 250000 usec Run, 1000000 usec Between
9  (PID: 542270) [StrawHat]$    [Status] Switching to run PID: 542331 (slow_countdown )
10   [Status] Switching to run PID: 542372 (ls)
11   [Status] Switching to run PID: 542331 (slow_countdown )
12 [PID: 542331] slow_countdown count down: 10 ...
13   [Status] Switching to run PID: 542372 (ls)
14 inc  lib  Makefile  obj  slow_bug  slow_countdown  slow_countup  slow_door  src  strawhat
15   [Status] Switching to run PID: 542331 (slow_countdown )
16 [PID: 542331] slow_countdown count down: 9 ...
17   [Status] Switching to run PID: 542471 (slow_countup 5 -p 200)
18   [Status] Switching to run PID: 542331 (slow_countdown )
19 [PID: 542331] slow_countdown count down: 8 ...
20 [PID: 542331] slow_countdown count down: 7 ...
21 [PID: 542331] slow_countdown count down: 6 ...
22 [PID: 542331] slow_countdown count down: 5 ...
23 [PID: 542331] slow_countdown count down: 4 ...
24   [Status] Switching to run PID: 542471 (slow_countup 5 -p 200)
25 [PID: 542471] slow_countup counting up: 0 ...
26   [Status] Switching to run PID: 542331 (slow_countdown )
27 [PID: 542331] slow_countdown count down: 3 ...
28 [PID: 542331] slow_countdown count down: 2 ...
29 [PID: 542331] slow_countdown count down: 1 ...
30 [PID: 542331] slow_countdown count down: 0 ...
31   [Status] Switching to run PID: 542471 (slow_countup 5 -p 200)
32 [PID: 542471] slow_countup counting up: 1 ...
33 [PID: 542471] slow_countup counting up: 2 ...
34 [PID: 542471] slow_countup counting up: 3 ...
35 [PID: 542471] slow_countup counting up: 4 ...
36 [PID: 542471] slow_countup counting up: 5 ...
37   [Status] No Processes Ready to Run
```

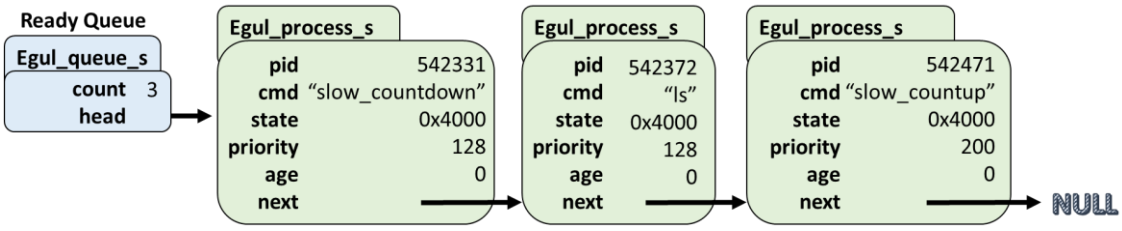
There are a few important things to note when looking at this portion of the sample output:

1. These are real processes running on a real computer.
    a. They don't all start and run at the same speed.
    b. Each run may be slightly different due to these factors.
    c. It may take being selected twice before any program produces output!
        i. Sometimes the first run is all setup before it gets to the first printf!
2. This is a **concurrent** environment.
    a. The prompt on StrawHat is running on a different processor than the programs.
        i. You may see a missing prompt because it may have printed earlier!
        ii. Simply press the Enter key to get another prompt. It's ok.
        iii. I/O on a real system is tricky in a concurrent environment. It's OK!

Let's look at this output line by line:

Lines 1-6 show us entering programs to run in that order.

At this point, StrawHat will call your code to pick the first process.
The **Ready Queue** at this point should look like this:



When we were entering commands on lines 1 and 3, we just entered the name of the command without any other options. So, these are both started with the default priority level of 128. On line 6, we added "-p 200" to the line, which set the priority level for slow_countup to 200.

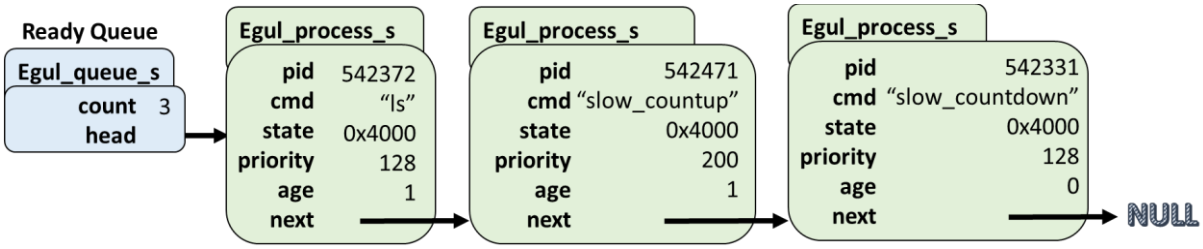
Line 7 shows us entering a built-in command **'start'** to start the engine to run the programs.
Line 8 is a status output to let us know we're now running processes.

Now started, StrawHat will call `egul_select`, which removes and returns slow_countdown.

Line 9 shows StrawHat switching to slow_countdown and starting it to run.

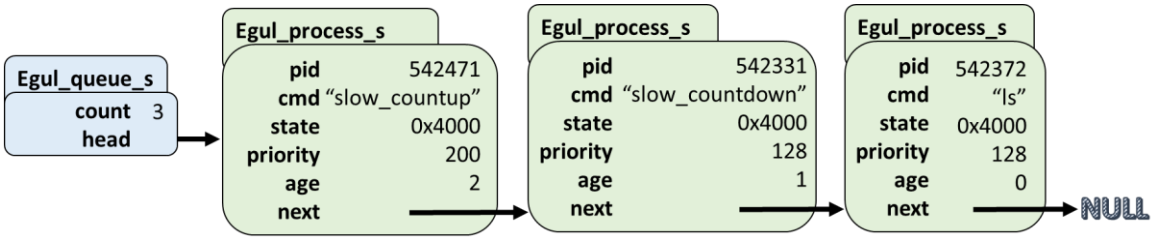
All processes take a little time to start up and begin to process, so it's common for the first 250ms of the execution to not print anything! This is what's happening here; the process starts and then ends without printing anything yet.

Line 10 shows StrawHat returning slow_countdown to the Ready Queue. At this point, here's what Ready Queue looks like:



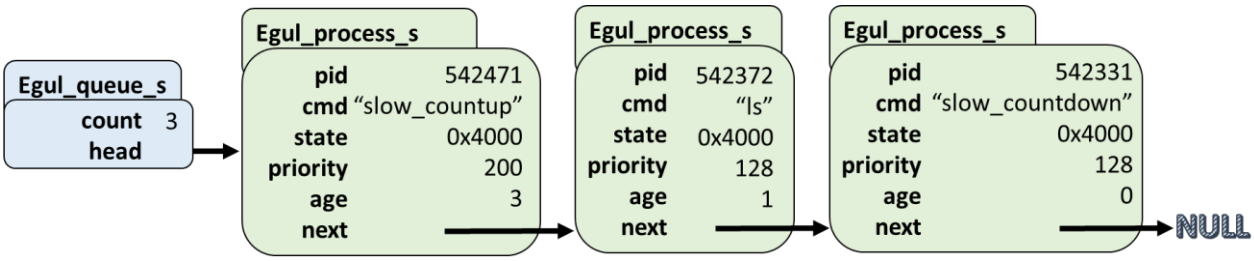
We see StrawHat called `egul_insert` to put slow_countdown back into the Ready Queue, where it was added to the end of the linked list. At this point, StrawHat calls `egul_select` and this time the ls process is returned.

Just like before, it took more than 250ms for ls to start printing, so it too is stopped and returned back to the Ready Queue, which now looks like this:

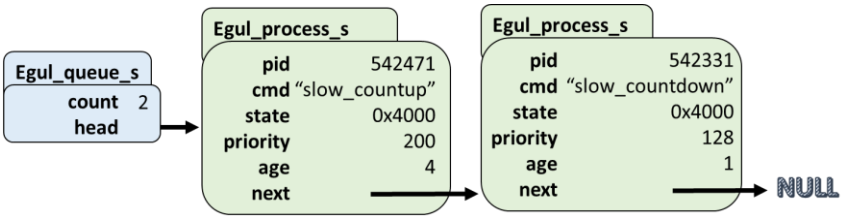


Line 11 Now the head pointer is pointing to slow_countup, however, we see the priority level of that process is 200. That is a much lower priority process than the other two, so when StrawHat calls `egul_select` next, we go with the first highest priority process we can find, which is slow_countdown again. This is shown being switched to. Now it continues running.

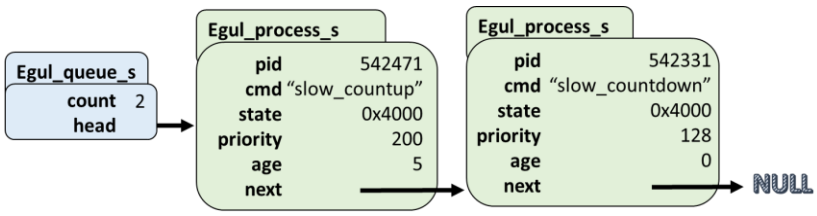
Line 12 finally shows output being printed. After the first switch, generally speaking all of the programs that start with **slow_** will print one line each time they are switched to. This shows it starting the countdown timer by printing out 10. It is then returned to the Ready Queue.



Lines 13 and **14** continue this by calling `egul_select` and getting the ls program. That program now outputs its first line of output, which shows the current files in the directory. At this point, the ls program is actually finished! Since it exited, StrawHat calls `egul_exited` and it will be put into the Terminated Queue. Your Ready Queue now looks like this:

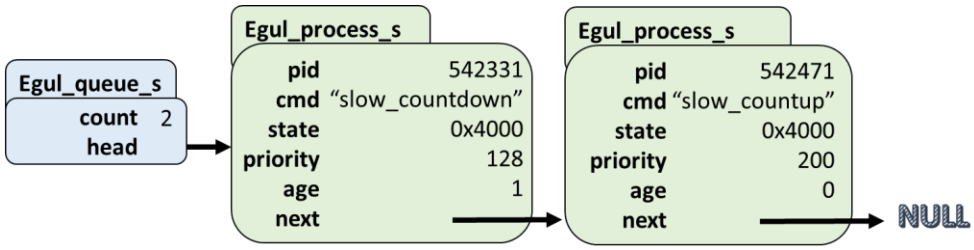


Lines 15 and **16** now show StrawHat calling `egul_select` and getting slow_countdown, again because it has the highest priority. slow_countdown now prints its second line with the number 9 and is returned when StrawHat calls `egul_insert`. The Ready Queue now looks like:



At this point, notice the age member on slow_countup has finally hit 5, which is the starving age. This means that regardless of its priority, as long as there are no critical processes, it will be picked.

Line 17 shows that when StrawHat called `egul_select`, it did get slow_countup, which runs for 250ms. Of course, this is the first time slow_countup is running, so it doesn't print anything and it gets put back into the Ready Queue.

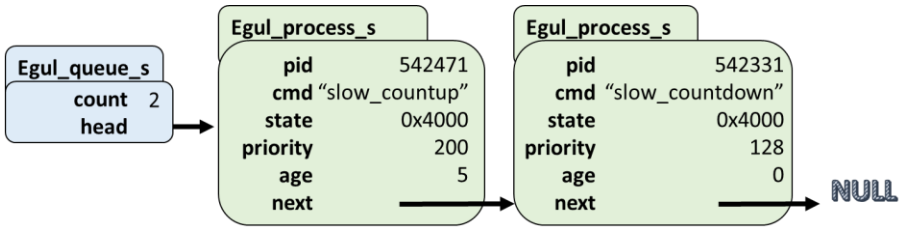


Now we see that since slow_countup just ran a little, its age is back to 0, meaning it will have to wait all over again until it becomes starving to run.

Line 18 shows when StrawHat calls `egul_select` to get the next process, slow_countdown is picked.

Lines 19-23 actually show the results of StrawHat calling `egul_select` five times in a row, which we can see with the five printed lines. (If you want to see the full output with all of the actions, you can turn debug mode on in StrawHat).

After Line 23, slow_countup is returned to the Ready Queue by `egul_insert`. Since it was selected five times in a row, the age on slow_countup went up to 5.



Lines 24 and **25** show that StrawHat called `egul_select` and got the starving process slow_countup.

At this point, you can follow the cycle of selects by tracking the age of each process to see the output follows the proper rules. After Line 30, slow_countdown exits and goes to the Terminated Queue, leaving slow_countup as the highest priority process left, which is why it now runs until it finishes.

6 Document Changelog

- v1.0 (build 4a99cc46)
  - Release Version
- v1.1 (build 4a99cc46)
  - Fixed a typo on Page 8. The command to terminate a process in StrawHat is **kill**. The typo said terminate, which is not a command on StrawHat.