CS 367 Project 1 - Fall 2022: Trilby User Manual

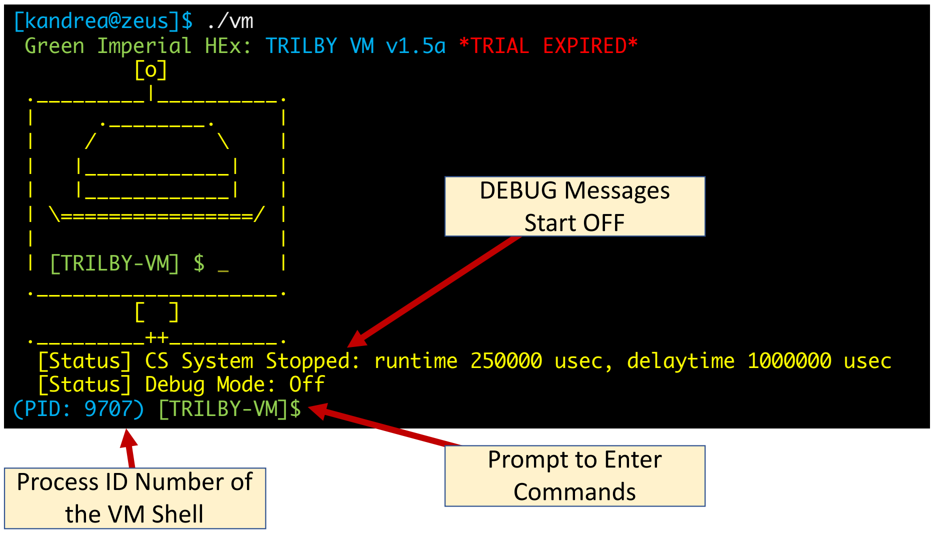
And Sample Executions

1 Introduction

Your code in **avan_sched.c** forms one library in the Trilby Virtual Machine. You can test your code entirely separately from Trilby (see P1_Self_Testing.pdf), but if you want to test the code running in full with the main VM, this manual will help you understand the various commands to use when running the Trilby VM (the vm executable that is generated when you run make).

1.1 TRILBY-VM Details

TRILBY-VM is not a simulator, it provides a shell that you can use to run normal non-interactive Linux commands like **ls**, **cal**, and **clear**. It can deal 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 **vim**, for example.



```
[kandrea@zeus]$ ./vm
 Green Imperial HEx: TRILBY VM v1.5a *TRIAL EXPIRED*
          [o]
 .__________|__________.
 |      .________.     |
 |     /          \    |
 |    |_____________|  |
 |    |_____________|  |
 | \================/  |
 |                     |
 | [TRILBY-VM] $ _     |
 .______________________.
          [  ]
 .__________++__________.
  [Status] CS System Stopped: runtime 250000 usec, delaytime 1000000 usec
  [Status] Debug Mode: Off
(PID: 9707) [TRILBY-VM]$
```

When TRILBY-VM starts, you will get a prompt just like it was a Shell in Linux. You can enter two types of commands here: **Built-In TRILBY Commands** or **Programs**. Built-In commands are used to control TRILBY-VM (such as 'quit'). Programs are normal Linux programs that you can run from either the current directory or from /usr/bin (like 'cal' or 'ls')

When you start TRILBY-VM, 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**.

When the CS system is running, any processes you have entered to run will be in the Ready Queue and will start being executed. When you finish writing your Avan Scheduler code, then this CS system calls your functions to do the following tasks related to running these Processes:

1. Create a new Process Struct and Initialize its Values
2. Chooses the Best Process to Run Next, Removes it from the Ready Queue and Returns It.
3. Insert a Process Struct into the Ready Queue
4. Insert a Process Struct into the Terminated Queue
5. Move a Process Struct from the Ready Queue to the Suspended Queue
6. Move a Process Struct from the Suspended Queue to the Ready Queue
7. Deallocate all Allocated Memory.

Trilby 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 steps until all processes have finished. The selection algorithm is detailed in the main Project Documentation, but is summarized as follows:

1. If there is a Process in the Ready Queue with the Critical Flag, it is the Best.
   a. On a Tie, the Best is the one with the lowest PID.
2. Otherwise, if a Process is Starving (skips count > MAX_SKIPS), choose it as Best.
   a. If more than one is Starving, pick the one with the lowest PID.
3. Otherwise, the Process with the highest Priority (lowest priority value!) is Best.
   a. On a Tie, the Best is the one with the lowest PID.

Once the Best is found, its skips are set to 0, it is removed from the Ready Queue, and a pointer to that Process Node is returned, so Trilby can run that process next. All other processes that were not picked should have their skips count incremented as well.

This means that when we run TRILBY with several programs, we'll see one of them running first for a while and then it will be interrupted to let starving processes run a little, before it resumes. So, the highest priority process at any given moment will be running most often, with all of the other processes getting little bits of run time throughout.

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 and it'll keep doing that until it's finished.

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

1.2 TRILBY-VM Sample Run

Here is an example run. We added line numbers on the left to help with the explanation, which is after the output sample. Here's a quick description of some of the programs being run:

- slow_cooker: This program counts from X down to 0, printing once every second.
- slow_printer: This program counts up from 0 to X, printing once every half-second.
- slow_bug: This program prints out an ASCII Art image, one line every half-second.

```
1   (PID: 11816) [TRILBY-VM]$  slow_cooker
2   (PID: 11816) [TRILBY-VM]$  slow_bug
3   (PID: 11816) [TRILBY-VM]$  cal
4   (PID: 11816) [TRILBY-VM]$  slow_printer
5   (PID: 11816) [TRILBY-VM]$  start
6     [Status] Starting the CS Execution
7   (PID: 11816) [TRILBY-VM]$  [PID: 11818] slow_cooker count down: 10 ...
8   [PID: 11818] slow_cooker count down: 9 ...
9   [PID: 11818] slow_cooker count down: 8 ...
10  [PID: 11818] slow_cooker count down: 7 ...
11  [PID: 11818] slow_cooker count down: 6 ...
12  [PID: 11818] slow_cooker count down: 5 ...
13  [PID: 11818] slow_cooker count down: 4 ...
14  [PID: 11818] slow_cooker count down: 3 ...
15  [PID: 11818] slow_cooker count down: 2 ...
16  [PID: 11819] Fight Bugs                      |     |
17       August 2022
18  Su Mo Tu We Th Fr Sa
19      1  2  3  4  5  6
20   7  8  9 10 11 12 13
21  14 15 16 17 18 19 20
22  21 22 23 24 25 26 27
23  28 29 30 31
24
25  [PID: 11821] slow_printer 0...
26  [PID: 11818] slow_cooker count down: 1 ...
27  [PID: 11818] slow_cooker count down: 0 ...
28  [PID: 11819]                              \\_V_//
29  [PID: 11819]                              \/=|=\/
30  [PID: 11819]                               [=v=]
31  [PID: 11821] slow_printer 1...
32  [PID: 11819]                             __\___/_____
33  [PID: 11819]                            /..[  _____ ]
34  [PID: 11819]                           /_  [ [  M /] ]
35  [PID: 11819]                          /../.[ [ M /@] ]
36  [PID: 11819]                         <-->[_[ [M /@/] ]
37  [PID: 11819]                        /../ [.[ [ /@/ ] ]
38  [PID: 11821] slow_printer 2...
39  [PID: 11819]     ________________]\ /__/  [_[ [/@/ C] ]
40  [PID: 11819]    <________________>>0---]  [=\ \@/ C / /
41  [PID: 11819]       ___     ___   ]/000o   /__\ \ C / /
42  [PID: 11819]          \    /            /....\ \_/ /
43  [PID: 11819]       ....\||/....         [___/=\___/
44  [PID: 11819]      .    .  .   .         [...] [...]
45  [PID: 11821] slow_printer 3...
46  [PID: 11819]     .      ..     .        [___/ \___]
```

There are a few important things to note when looking at this 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!
2. This is a **concurrent** environment.
    a. The prompt on TRILBY is running on a different processor than the programs.
    b. You may see a missing prompt from time to time.
        i. This happens because it may have printed earlier!
        ii. Simply press the Enter key to get another prompt. It's ok.
    c. You may see an extra prompt from time to time.
        i. I/O on a real system is tricky in a concurrent environment. It's OK!
    d. You may see program output at the end of other lines.

Let's look at this output line by line:

Lines 1-4 show us entering programs to run in that order.
Line 5 shows us entering a built-in command 'start' to start the engine to run the programs.
Line 6 is a status output to let us know we're now running processes.

At this point, TRILBY will call your code to pick the first process. Avan told it to run slow_cooker.

Line 7 shows the prompt after out start command AND the output of the first process!
This is showing that the TRILBY-VM and the process we chose to run are running concurrently.
Lines 8-11 show that this process keeps getting selected to run while the others are waiting.

At this point, the other processes have been skipped 5 times, triggering Starvation protections. When running this, there is a noticeable pause at this point while those other processes are selected to run for 1 execution each. Unfortunately, as this is a real system, some processes may take a little time to run before they start printing out output, so there is no output.
(You can turn the debug mode on to show that the switches are actually happening).

Lines 12-15 show slow_cooker continues to be selected, skipping the others another 5 times.

Now, Avan sees all other processes are starving again, so it returns those to run once each.
Lines 16-25 show the starving processes getting a tiny bit of CPU time.
Lines 26-27 run slow_cooker until it's finished. Now, the highest priority process is slow_bug.

The rest of the output shows slow_bug running 5 times in a row, then the remaining process (slow_printer) gets to run once because it was starving. This repeats until all processes are finished.

2 Building and Running TRILBY-VM (./vm)

You will receive the file **project1_handout.tar**, which will create a handout folder on Zeus.

```
kandrea@zeus-1:handout$ tar -xvf project1_handout.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 **avan_sched.c**, which is the only file you will be modifying and submitting. You will also have very useful header files (**avan_sched.h**) along with other files for the simulator itself in the **inc/** directory.

2.1 Building TRILBY-VM

To build TRILBY-VM, run the make command:

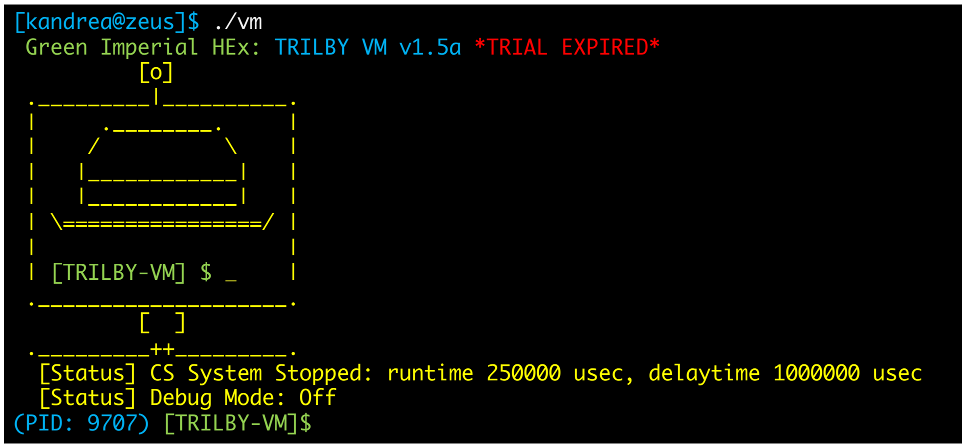
```
kandrea@zeus-1:handout$ make
gcc -std=gnu99  -Og -Wall -Werror -Wno-error=unused-variable -Wno-error=unused-function -
pthread -I./inc -L./obj -g -c -o obj/vm.o src/vm.c
gcc -std=gnu99  -Og -Wall -Werror -Wno-error=unused-variable -Wno-error=unused-function -
pthread -I./inc -L./obj -g -c -o obj/vm_cs.o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

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

2.2 Running TRILBY-VM

To start TRILBY-VM, run **./vm**

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



```
[kandrea@zeus]$ ./vm
 Green Imperial HEx: TRILBY VM v1.5a *TRIAL EXPIRED*
          [o]
 ._________|__________.
 |     .________.     |
 |    /          \    |
 |   |____________|   |
 |   |____________|   |
 | \================/ |
 |                    |
 | [TRILBY-VM] $ _    |
 .____________________.
          [  ]
 ._________++_________.
  [Status] CS System Stopped: runtime 250000 usec, delaytime 1000000 usec
  [Status] Debug Mode: Off
(PID: 9707) [TRILBY-VM]$
```

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 TRILBY-VM commands to control the VM. All of the TRILBY-VM Commands are detailed after the function details of what you will be writing.

2.3 Compiling Options

There is one very important note about our compiling options that may differ from what you used in CS262/CS222 (or other C classes). We're using **-Wall -Werror**. This means that it'll warn you about a lot of bad practices and makes every warning into an error.

To compile your program, you will need to address all warnings!

3 TRILBY-VM Manual

TRILBY-VM is a full-featured Virtual Environment 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 avan_select to get a process to run from your code, then it will run that process for a **runtime** number of microseconds (usec). After that time, the CS engine will then stop it from running on the CPU and return it (either avan_insert or avan_quit). 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).

TRILBY-VM 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

- **start** This will start the CS Engine running.
- **stop** This will stop the CS Engine running. (You can start and stop it as much as you want!)
- **status** This will print out a status message about the CS Engine settings and if it's running.
- **runtime X** This will change the **runtime** to a new usec value. (Default 250000 usec)
- **delaytime X** This will change the **delaytime** to a new usec value. (Default 1000000 usec)

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) [TRILBY-VM]$  runtime 500000
  [Status] Setting CS System: runtime 500000 usec, delaytime 1000000 usec
(PID: 178230) [TRILBY-VM]$  delaytime 2000000
  [Status] Setting CS System: runtime 500000 usec, delaytime 2000000 usec
(PID: 178230) [TRILBY-VM]$  status
  [Status] CS System Stopped: runtime 500000 usec, delaytime 2000000 usec
(PID: 178230) [TRILBY-VM]$  start
  [Status] Starting the CS Execution
(PID: 178230) [TRILBY-VM]$
```

Before you have written anything for your avan_sched.c functions, this is all it will do. For the CS Engine to do anything, it needs to get Processes from your code.

You will need to implement avan_insert, avan_new_process, avan_select, and avan_get_size, at a minimum, for the system to begin processing.

The **Shell** component of TRILBY-VM 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 TRILBY-VM Shell

- **schedule** This will print out all of the processes in all three of your Linked Lists!
- **suspend X** This will suspend running process with PID X (and call your avan_suspend)
- **resume X** This will resume running process with PID X (and call your avan_resume)
- [Linux Command] You can enter any common Linux Command with arguments to run.
  - Note, there are three 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_cooker [X]** Prints out one message every second for 10 seconds (or X sec)
  - **slow_printer [X]** Prints out one message every half-second for 10 (or X) iterations
  - **slow_bug** Prints out an ASCII art of a Bug Hunting Knight, one line per 0.5 sec
    - **Courtesy of** https://www.asciiart.eu/computers/bug (Author Unknown)
- **debug** Toggles the TRILBY-VM Debug Messages On/Off (can be spammy)
- **quit** Shuts EVERYTHING down responsibly and quits. (calls your avan_cleanup)\

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

- **-p X** Run the process at priority level X. (Ranges from 1 - 255, the default level is 128)
- **-c** Run the process with Critical Priority. (Always runs first to completion)

Example of the Special Options to run slow_hat at Priority 200, with command line argument 10.

```
(PID: 178360) [TRILBY-VM]$  slow_hat 10 -p 200
```

You may also edit the top portion of **inc/vm_settings.h** header. This has all of the initial default settings for running TRILBY-VM 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.

3.1 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 virtual machine. Don't worry, the text you were typing is still there so you can keep typing and hit enter and it'll still work.

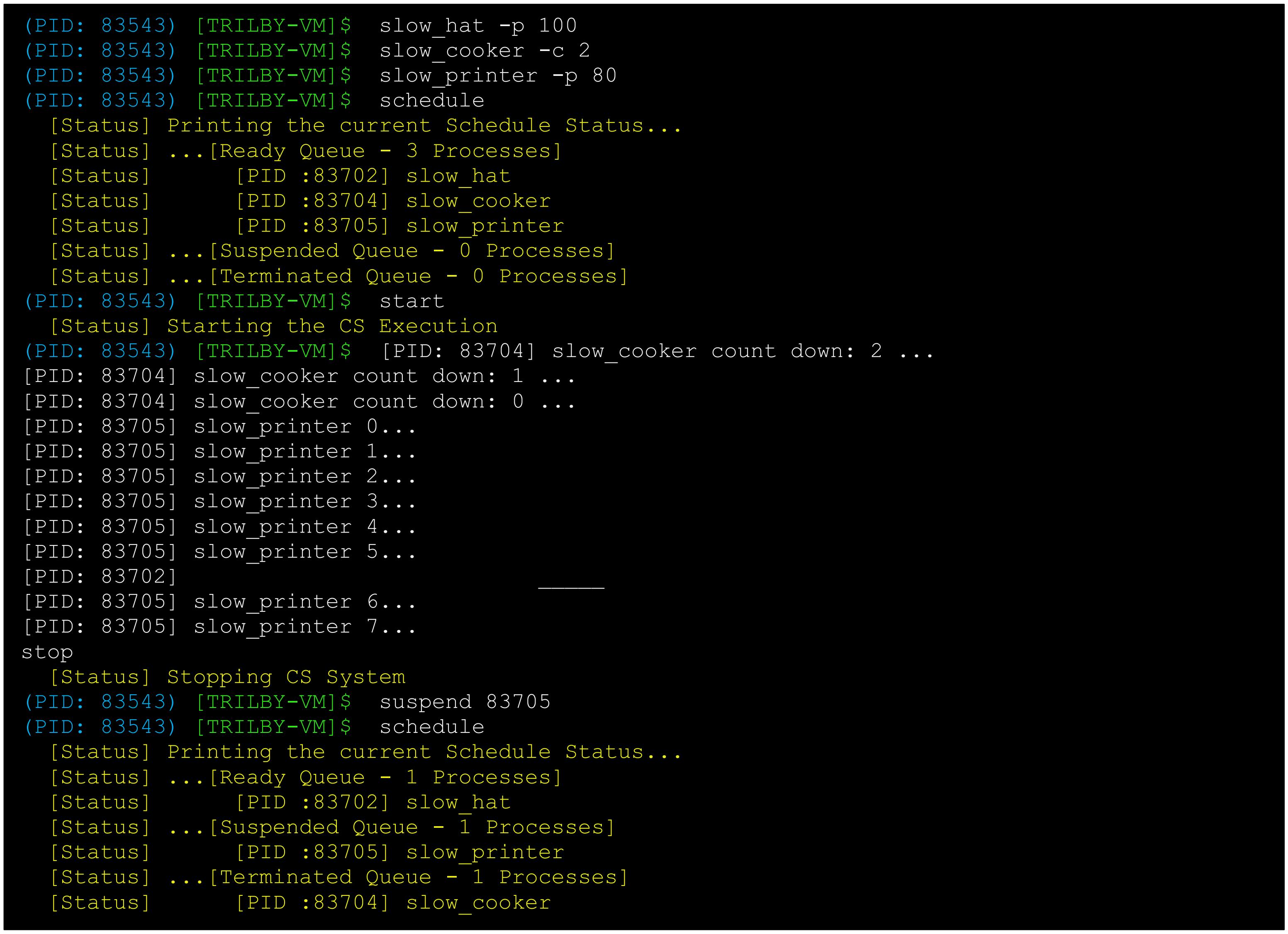
If you ever want to see the prompt again, you can always just hit Enter without typing anything.

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).

Here is a sample run of TRILBY-VM

In this sample,

1. I start three processes and then check the current schedule to show that they've all been added to the Ready Queue.
    a. slow_hat is started with the -p 100 option to set its Priority Level to 100.
    b. slow_cooker is started with the -c option to set its Critical Flag.
    c. slow_printer is started with the -p 80 option to set its Priority Level to 80 (higher than 100)
2. Then I use the **start** command to start up the CS Engine.
    a. You see the slow_cooker (Critical Process) run until it is finished.
    b. Then slow_printer begins executing.
    c. When skips on slow_hat reaches the MAX_SKIPS, it runs once, then we go back to slow_printer.
3. After a few seconds, I typed in the **stop** command to Stop the CS Engine.
4. I then use the **suspend 83705** command to suspend the process with that pid (slow_printer).
5. Then I use **schedule** again.
    a. You can see slow_cooker is in the Terminated Queue because it already finished.
    b. You can see slow_printer is in the Suspended Queue. It will not run again until resumed.
    c. Finally, slow_hat is the only process left in the Ready Queue, so it will run next.



```
(PID: 83543) [TRILBY-VM]$  slow_hat -p 100
(PID: 83543) [TRILBY-VM]$  slow_cooker -c 2
(PID: 83543) [TRILBY-VM]$  slow_printer -p 80
(PID: 83543) [TRILBY-VM]$  schedule
  [Status] Printing the current Schedule Status...
  [Status] ...[Ready Queue - 3 Processes]
  [Status]      [PID :83702] slow_hat
  [Status]      [PID :83704] slow_cooker
  [Status]      [PID :83705] slow_printer
  [Status] ...[Suspended Queue - 0 Processes]
  [Status] ...[Terminated Queue - 0 Processes]
(PID: 83543) [TRILBY-VM]$  start
  [Status] Starting the CS Execution
(PID: 83543) [TRILBY-VM]$  [PID: 83704] slow_cooker count down: 2 ...
[PID: 83704] slow_cooker count down: 1 ...
[PID: 83704] slow_cooker count down: 0 ...
[PID: 83705] slow_printer 0...
[PID: 83705] slow_printer 1...
[PID: 83705] slow_printer 2...
[PID: 83705] slow_printer 3...
[PID: 83705] slow_printer 4...
[PID: 83705] slow_printer 5...
[PID: 83702]                           _____
[PID: 83705] slow_printer 6...
[PID: 83705] slow_printer 7...
stop
  [Status] Stopping CS System
(PID: 83543) [TRILBY-VM]$  suspend 83705
(PID: 83543) [TRILBY-VM]$  schedule
  [Status] Printing the current Schedule Status...
  [Status] ...[Ready Queue - 1 Processes]
  [Status]      [PID :83702] slow_hat
  [Status] ...[Suspended Queue - 1 Processes]
  [Status]      [PID :83705] slow_printer
  [Status] ...[Terminated Queue - 1 Processes]
  [Status]      [PID :83704] slow_cooker
```