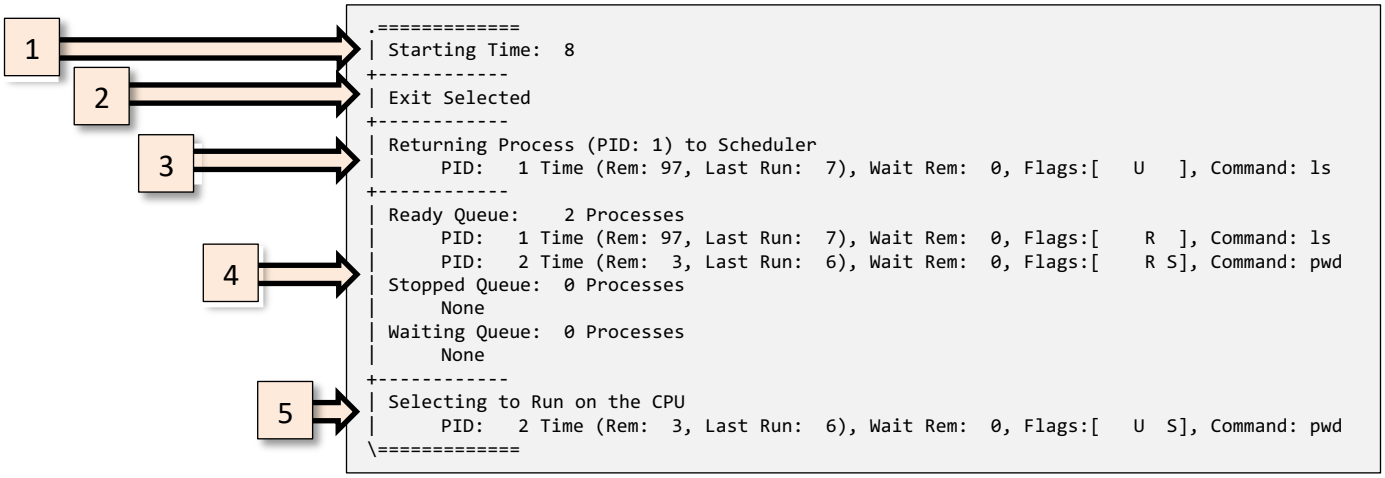
CS 367 Project #1 - Spring 2021:
CPU Process Scheduler
Sample Runs of Provided Traces

1 Reading the Output

Output from the scheduler shows the state of the OS simulator during each one of the loops.



Each block shows five pieces of information as indicated above:

1. The starting Clock time for that cycle.
2. Which action was processed for that Clock time.
3. Which, if any, process is taken off of the CPU and returned back to the scheduler (using your `scheduler_add` function) or finished (using your `scheduler_finish` function).
4. The state of all three lists as they are after the action finishes, but before a process is selected.
5. Finally, it shows which process, if any, were selected to run on the CPU.

The process information within any of these informational areas shows you the PID for that process, how much time is remaining, the time it was last run, and the name of the command itself.

The flags are also listed for each process in the list, using the following codes:

Code	Flag Name
C	CREATED
U	RUNNING
W	WAITING

Code	Flag Name
R	READY
X	TERMINATED
S	SUDO

Description of the Simulator Steps, Showing the Order the Output Prints in.

1. Starting Time is Printed
2. Action is read from the Trace File and Printed.
3. Action is parsed and executed by calling one of your functions. (nothing printed)
4. Process is removed from the CPU and Printed.
5. Process removed from the CPU is added using your **`scheduler_add`**, or **`scheduler_finish`** function, if the process has no run time left (nothing printed).
6. Current contents of each queue is Printed.
7. **`scheduler_select`** is called to remove a selected process from ready queue. (nothing printed)
8. **`scheduler_io_run`** is called to update the first waiting process and may move to ready queue.
9. Selected process is then dispatched to the CPU for execution and Printed.

Example Run (Below is a description of what this shows)

```
.=============
| Starting Time:  13
+------------
| Process Starting
|     PID:   5 Time (Rem:  4, Last Run: 13), Wait Rem:  0, Flags:[     C ], Command: cal
+------------
| Returning Process (PID: 1) to Scheduler
|     PID:   1 Time (Rem: 11, Last Run: 12), Wait Rem:  0, Flags:[   U   ], Command: ls
+------------
| Ready Queue:    2 Processes
|     PID:   1 Time (Rem: 11, Last Run: 12), Wait Rem:  0, Flags:[    R  ], Command: ls
|     PID:   5 Time (Rem:  4, Last Run: 13), Wait Rem:  0, Flags:[    R  ], Command: cal
| Stopped Queue:  0 Processes
|     None
| Waiting Queue:  0 Processes
|     None
+------------
| Selecting to Run on the CPU
|     PID:   5 Time (Rem:  4, Last Run: 13), Wait Rem:  0, Flags:[   U   ], Command: cal
\=============
```

Reading this sample output above using those same steps.

1. This sample output above shows all of the events that happen for Time 13
2. The next section says, ‘Process Starting’, meaning the next action on the trace file was to create a new process. The simulator will prepare all of the information from the tracefile to use.
3. The simulator calls **`scheduler_generate`** to create the process struct, followed by **`scheduler_add`**.
    a. It is created with state **`CREATE`** and will go into your Ready Queue as a **`READY`** state process.
    b. This will be inserted in ascending PID order.
4. The next section says, ‘Returning Process’. The process with PID 1 ran last cycle, so it is removed from the CPU and printed to the screen.
    a. It has a Time Remaining of 11, so it needs to run on the CPU 11 more times to finish.
5. Since it has a Time Remaining greater than 0, the simulator calls your **`scheduler_add`** on it (PID 1).
    a. This adds it back to the ready queue, where it will go before PID 5.
6. The simulator then prints the status of all queues.
    a. You can see PID 1 as the first process, followed by PID 5 in the Ready Queue.
7. The simulator calls your **`scheduler_select`** function to get the next process to run.
    a. Your function will remove the process PID 5 because it’s **`time_remaining`** is lowest.
8. The simulator will then call **`scheduler_io_run`**, but with an empty waiting queue, nothing happens.
9. Finally, the simulator prints the information on PID 5, which you just returned from **`scheduler_select`**.

2 traces/trace1.dat

Notes: This just exists on the first loop, at time 1. No processes are created, so all of the Queues should be empty.

2.1 Tracefile

```
exit
```

2.2 Sample Output from Scheduler

```
kandrea@zeus-1:handout$ ./scheduler traces/trace1.dat
.=============
| Starting Time:  1
+------------
| Exit Selected
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    0 Processes
|      None
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      None
\=============

We're done!
```

3 traces/trace2.dat

Notes: This starts a new process at time 1 (ls -al) and starts it with PID = 1, `time_remaining` = 1, and `exit_code` = 42. The exit code will be automatically applied to your process's flags field when the process runs its last time on the CPU. So, you will only need to extract it.

At Time 1, there's nothing currently running to be returned. For the action, it shows...

- A new process with PID 1 is starting up (initialized to `CREATED`).
- For the linked lists, you can see that this new process was moved into the Ready Queue and you changed its status to `READY`).
- You can finally see that your `scheduler_select` function properly chose the Process with PID 1 to run next because it had the lowest `time_remaining`

This shows that there is 1 time remaining at the end of the scheduling phase.

At Time 2, you can see...

- The action for time 2 is exit, so that is helpfully printed in the action area.
- The process you previously selected with PID 1 has run for 1 time unit (it has 0 time remaining) and is unloaded and returned to you through your `scheduler_finish`.
    - This is printed here as Finishing Process first before your function is called.
    - Your function extracts the exit_code, which is 42, and returns it to the simulator.
    - The simulator then prints out the value you returned.
- At this point, the scheduler ends.

< Tracefile and Expected Output on the Next Page >

3.1 Tracefile

```
ls -al [1,1,42]
exit
```

3.2 Sample Output from Scheduler

```
kandrea@zeus-1:handout$ ./scheduler traces/trace2.dat
.=============
| Starting Time:  1
+------------
| Process Starting
|      PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[     C ], Command: ls -al
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    1 Processes
|      PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls -al
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls -al
\=============

.=============
| Starting Time:  2
+------------
| Exit Selected
+------------
| Finishing Process (PID: 1)
|      PID:   1 Time (Rem:  0, Last Run:  1), Wait Rem:  0, Flags:[ X     ], Command: ls -al
|      Process Exited with Exit Code 42
+------------
| Ready Queue:    0 Processes
|      None
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      None
\=============

We're done!
```

4 traces/trace3.dat

Notes: This starts a new process at time 1 (ls) and starts it with PID = 1, `time_remaining` = 2, and `exit_code` = 42. At time 2, this will then go to the I/O device, where it will wait for 1 time unit for that device to finish. After that, it'll go back to the CPU to finish.

At Time 1, there's nothing currently running to be returned. For the action, it shows...

- A new process with PID 1 is starting up (initialized to `CREATED`).
- For the linked lists, you can see that this new process was moved into the Ready Queue and you changed its status to `READY`).
- You can finally see that your `scheduler_select` function properly chose the Process with PID 1 to run next because it had the lowest `time_remaining`.

This shows that there is 1 time remaining at the end of the scheduling phase.

At Time 2, you can see...

- The action for time 2 is wait on PID 1, so this will cause the process to be unloaded from the CPU and passed into your `scheduler_wait` function with `io_time` of 1.
  - This will cause the process to be added to your waiting queue.
- At this point, the process that was on the CPU has already moved to the waiting queue, so there's nothing to return to the Scheduler.
- Now, with nothing in the Ready Queue, nothing is returned for `scheduler_select`, so the CPU will not run anything this cycle.
- This is when the output prints all of the queues.
- For your `scheduler_io_run`, however, we see the wait_remaining on the first process in the waiting queue is a 1, so we decrement it to 0.
  - Now that it's a 0, we actually move it back to the Ready Queue! It won't run this cycle, since we've already selected, but it'll be ready to run next cycle.

At Time 3, the action is pass, which means no new actions will be done this cycle. We can also see that the process with PID 1 is back in the ready queue and will be selected for its last time.

At Time 4, exit was chosen, so after the process finishes, the simulator will exit.

4.1 Tracefile

```
ls [1,2,42]
wait 1
pass
exit
```

< Expected Output on the Next Page >

4.2 Sample Output from Scheduler

```
kandrea@zeus-1:handout$ ./scheduler traces/trace3.dat
.=============
| Starting Time:  1
+------------
| Process Starting
| PID:   1 Time (Rem:  2, Last Run:  1), Wait Rem:  0, Flags:[     C ], Command: ls
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    1 Processes
| PID:   1 Time (Rem:  2, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
| Stopped Queue:  0 Processes
| None
| Waiting Queue:  0 Processes
| None
+------------
| Selecting to Run on the CPU
| PID:   1 Time (Rem:  2, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls
\=============

.=============
| Starting Time:  2
+------------
| Waiting Process (PID:  1) for 1 time units.
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    0 Processes
| None
| Stopped Queue:  0 Processes
| None
| Waiting Queue:  1 Processes
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  1, Flags:[W      ], Command: ls
+------------
| Selecting to Run on the CPU
| None
\=============

.=============
| Starting Time:  3
+------------
| Passing (No Action)
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    1 Processes
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
| Stopped Queue:  0 Processes
| None
| Waiting Queue:  0 Processes
| None
+------------
| Selecting to Run on the CPU
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls
\=============
```

```
.=============
| Starting Time:  4
+------------
| Exit Selected
+------------
| Finishing Process (PID: 1)
|PID:   1 Time (Rem:  0, Last Run:  3), Wait Rem:  0, Flags:[ X     ], Command: ls
|       Process Exited with Exit Code 42
+------------
| Ready Queue:    0 Processes
|None
| Stopped Queue:  0 Processes
|None
| Waiting Queue:  0 Processes
|None
+------------
| Selecting to Run on the CPU
|None
\=============

We're done!
```

5 traces/trace4.dat

Notes: This trace demonstrates the starvation protection using aging we have in scheduling. The scheduling algorithm you implemented chooses the process with the lowest `time_remaining` from the ready queue (with the process with the lowest PID to break any ties). While we're walking the list to find the lowest `time_remaining` process, we're also looking for any starving process. This is when `clock_get_time()` – `time_last_run` >= `TIME_STARVATION`. If we find a starving process, we immediately select that one, regardless of `time_remaining`.

For this build, TIME_STARVATION is set to 6.

So, for this trace, two processes are created. PID 1 (ls) will run for 99 time units, and PID 2 (sudo pwd) will run for 8 time units. Since PID 1 is the only process created in Time 1, it will be selected to run.

After that, since PID 2 has the smallest `time_remaining`, it will be selected to run during Times 2, 3, 4, 5, and 6.

At time 7, we see that PID 2 is still the lowest `time_remaining` by far, however, we also see that the current time is 7, and PID 1 last ran at 1, which means it's now starving. (7 – 1 >= 6). Since it's starving, that means it will automatically be selected and get to run, even though it has a higher `time_remaining`.

At time 8, we see that PID 1 has a new `time_last_run` of 7, so it's no longer starving, which means PID 2's lower `time_remaining` will win and it will be selected again.

This trace demonstrates that even when you have a process with a much better priority, the other processes can still make sure they get some occasional access to the CPU too!

5.1 Tracefile

```
ls [1,99,1]
sudo pwd [2,8,2]
pass
pass
pass
pass
pass
exit
```

5.2 Sample Output from Scheduler

```
kandrea@zeus-1:handout$ ./scheduler traces/trace4.dat
.=============
| Starting Time:  1
+------------
| Process Starting
|      PID:   1 Time (Rem: 99, Last Run:  1), Wait Rem:  0, Flags:[     C ], Command: ls
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    1 Processes
|      PID:   1 Time (Rem: 99, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   1 Time (Rem: 99, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls
\=============

.=============
| Starting Time:  2
+------------
| Process Starting
|      PID:   2 Time (Rem:  8, Last Run:  2), Wait Rem:  0, Flags:[     CS], Command: pwd
+------------
| Returning Process (PID: 1) to Scheduler
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls
+------------
| Ready Queue:    2 Processes
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
|      PID:   2 Time (Rem:  8, Last Run:  2), Wait Rem:  0, Flags:[    R S], Command: pwd
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   2 Time (Rem:  8, Last Run:  2), Wait Rem:  0, Flags:[   U  S], Command: pwd
\=============

.=============
| Starting Time:  3
+------------
| Passing (No Action)
+------------
| Returning Process (PID: 2) to Scheduler
|      PID:   2 Time (Rem:  7, Last Run:  2), Wait Rem:  0, Flags:[   U  S], Command: pwd
+------------
| Ready Queue:    2 Processes
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
|      PID:   2 Time (Rem:  7, Last Run:  2), Wait Rem:  0, Flags:[    R S], Command: pwd
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
```

```
|      PID:   2 Time (Rem:  7, Last Run:  2), Wait Rem:  0, Flags:[   U  S], Command: pwd
\=============

.=============
| Starting Time:  4
+------------
| Passing (No Action)
+------------
| Returning Process (PID: 2) to Scheduler
|      PID:   2 Time (Rem:  6, Last Run:  3), Wait Rem:  0, Flags:[   U  S], Command: pwd
+------------
| Ready Queue:    2 Processes
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
|      PID:   2 Time (Rem:  6, Last Run:  3), Wait Rem:  0, Flags:[    R S], Command: pwd
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   2 Time (Rem:  6, Last Run:  3), Wait Rem:  0, Flags:[   U  S], Command: pwd
\=============

.=============
| Starting Time:  5
+------------
| Passing (No Action)
+------------
| Returning Process (PID: 2) to Scheduler
|      PID:   2 Time (Rem:  5, Last Run:  4), Wait Rem:  0, Flags:[   U  S], Command: pwd
+------------
| Ready Queue:    2 Processes
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
|      PID:   2 Time (Rem:  5, Last Run:  4), Wait Rem:  0, Flags:[    R S], Command: pwd
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   2 Time (Rem:  5, Last Run:  4), Wait Rem:  0, Flags:[   U  S], Command: pwd
\=============

.=============
| Starting Time:  6
+------------
| Passing (No Action)
+------------
| Returning Process (PID: 2) to Scheduler
|      PID:   2 Time (Rem:  4, Last Run:  5), Wait Rem:  0, Flags:[   U  S], Command: pwd
+------------
| Ready Queue:    2 Processes
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
|      PID:   2 Time (Rem:  4, Last Run:  5), Wait Rem:  0, Flags:[    R S], Command: pwd
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   2 Time (Rem:  4, Last Run:  5), Wait Rem:  0, Flags:[   U  S], Command: pwd
```

```
\=============

.=============
| Starting Time:  7
+------------
| Passing (No Action)
+------------
| Returning Process (PID: 2) to Scheduler
|      PID:   2 Time (Rem:  3, Last Run:  6), Wait Rem:  0, Flags:[   U  S], Command: pwd
+------------
| Ready Queue:    2 Processes
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
|      PID:   2 Time (Rem:  3, Last Run:  6), Wait Rem:  0, Flags:[    R S], Command: pwd
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   1 Time (Rem: 98, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls
\=============

.=============
| Starting Time:  8
+------------
| Exit Selected
+------------
| Returning Process (PID: 1) to Scheduler
|      PID:   1 Time (Rem: 97, Last Run:  7), Wait Rem:  0, Flags:[   U   ], Command: ls
+------------
| Ready Queue:    2 Processes
|      PID:   1 Time (Rem: 97, Last Run:  7), Wait Rem:  0, Flags:[    R  ], Command: ls
|      PID:   2 Time (Rem:  3, Last Run:  6), Wait Rem:  0, Flags:[    R S], Command: pwd
| Stopped Queue:  0 Processes
|      None
| Waiting Queue:  0 Processes
|      None
+------------
| Selecting to Run on the CPU
|      PID:   2 Time (Rem:  3, Last Run:  6), Wait Rem:  0, Flags:[   U  S], Command: pwd
\=============


We're done!
```

6 traces/trace5.dat

Notes: This trace demonstrates stopping and continuing a process.

For this trace, you have ls created with PID 1 and a `time_remaining` of 2. After Time 1, this process has been selected and run once. At Time 2, it has `time_remaining` of 1 and is moved back to the Ready Queue. The action now is to STOP PID 1, which causes the simulator to call your `scheduler_stop` function. Your function will go into the ready queue, remove PID 1, change its state to `STOPPED`, then move it to the stopped queue.

At the end of Time 2, there are no processes in the ready queue, so nothing is scheduled. Likewise with Time 3, which leaves the CPU idle.

At Time 4, CONT is sent, which calls your `scheduler_continue` function. PID 1 is removed from the stopped queue, has its state changed to `READY`, and moved into the ready queue. At this point, the scheduler will see it in the ready queue and schedule it to run.

At Time 5, it finishes running and moves to the defunct queue as normal.

6.1 Tracefile

```
ls [1,2,1]
kill -STOP 1
pass
kill -CONT 1
exit
```

6.2 Sample Output from Scheduler

```
kandrea@zeus-1:handout$ ./scheduler traces/trace5.dat
.=============
| Starting Time:  1
+------------
| Process Starting
| PID:   1 Time (Rem:  2, Last Run:  1), Wait Rem:  0, Flags:[     C ], Command: ls
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    1 Processes
| PID:   1 Time (Rem:  2, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
| Stopped Queue:  0 Processes
| None
| Waiting Queue:  0 Processes
| None
+------------
| Selecting to Run on the CPU
| PID:   1 Time (Rem:  2, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls
\=============

.=============
| Starting Time:  2
+------------
```

```
| Sending STOP to Process (PID:  1)
+------------
| Returning Process (PID: 1) to Scheduler
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[  T    ], Command: ls
+------------
| Ready Queue:    0 Processes
| None
| Stopped Queue:  1 Processes
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[  T    ], Command: ls
| Waiting Queue:  0 Processes
| None
+------------
| Selecting to Run on the CPU
| None
\=============

.=============
| Starting Time:  3
+------------
| Passing (No Action)
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    0 Processes
| None
| Stopped Queue:  1 Processes
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[  T    ], Command: ls
| Waiting Queue:  0 Processes
| None
+------------
| Selecting to Run on the CPU
| None
\=============

.=============
| Starting Time:  4
+------------
| Sending CONT to Process (PID:  1)
+------------
| Nothing to Return to Scheduler
+------------
| Ready Queue:    1 Processes
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[    R  ], Command: ls
| Stopped Queue:  0 Processes
| None
| Waiting Queue:  0 Processes
| None
+------------
| Selecting to Run on the CPU
| PID:   1 Time (Rem:  1, Last Run:  1), Wait Rem:  0, Flags:[   U   ], Command: ls
\=============

.=============
| Starting Time:  5
+------------
| Exit Selected
+------------
| Finishing Process (PID: 1)
| PID:   1 Time (Rem:  0, Last Run:  4), Wait Rem:  0, Flags:[ X     ], Command: ls
|        Process Exited with Exit Code 1
+------------
```

```
| Ready Queue:    0 Processes
| None
| Stopped Queue:  0 Processes
| None
| Waiting Queue:  0 Processes
| None
+------------
| Selecting to Run on the CPU
| None
\=============

We're done!
```