CENG328
Operating Systems

Laboratory VII
Processes III & CPU Scheduling

1. Processes – Signals & Signal Handling

- **Signal**; code28.c
  - Signals are mechanisms for communicating with and manipulating processes.
  - A signal is a special message sent to a process. Signals are asynchronous; when a process receives a signal, it processes the signal immediately, without finishing the current function or even the current line of code.
  - Each signal type is specified by its signal number, but in programs, you usually refer to a signal by its name.
  - How to terminate the program? Break with **CTRL + Z**, you will get

    ```
    [1]+ Stopped code28
    ```

  - then kill the stopped process with **kill %1**.
  - To view a list of all available signals, use **man kill**.

1. Processes – Signals & Signal Handling

- **Signal Handling**; - code29.c
  - Even assigning a value to a global variable can be dangerous because the assignment may actually be carried out in two or more machine instructions, and a second signal may occur between them, leaving the variable in a corrupted state.
  - If you use a global variable to flag a signal from a signal-handler function, it should be of the special type **sig_atomic_t**.
  - Assignments to variables of this type are performed in a single instruction and therefore cannot be interrupted midway.
  - This program uses a signal-handler function to count the number of times that the program receives **SIGUSR1**, one of the signals reserved for application use.
- **Exercise**: Write a C program which controls the POSIX interrupt signal (**SIGINT**).
  - Your program should continuously increase an integer value.
  - With each interrupt request (**CTRL + C**), your program should;
    - Display the current value of counter.
    - Toggle whether the counter should continue increasing, or stop.

2. CPU Scheduling Simulation

- Download the simulator from link: ProcessCPUScheduling.tar.gz.

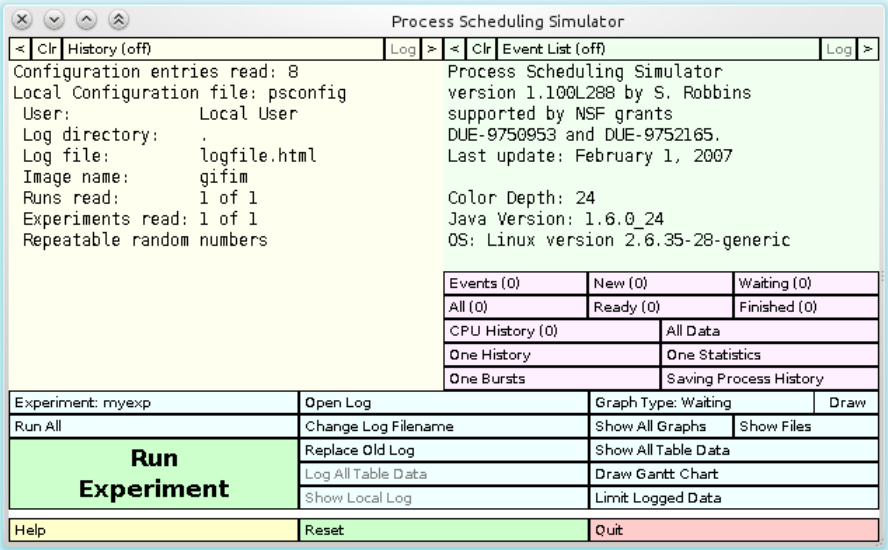
- Unpack the simulator with the following command:

  cd Downloads
  tar zxvf ProcessCPUScheduling.tar.gz

- Run the simulator with the following command:

  cd ProcessCPUScheduling
  chmod +x runps
  ./runps

- First of all, read the **psdoc.html** file carefully. This file contains detailed information about running and modifying the parameters of the simulation.



2. CPU Scheduling Simulation

- Each time you want to modify simulation parameters such as first arrival or CPU burst, you have to edit the **myrun.run** file and restart the simulator. The contents of myrun.run file look like this:

```
name         myrun
comment      This is a sample experimental run file
algorithm    SJF
numprocs     5
firstarrival 0.0
interarrival constant 0.0
duration     constant 25.0
cpuburst     constant 5.0
ioburst      constant 1.0
basepriority 1.0
```

- Each line in this file hold various parameters related to the simulation such as the algorithm used for the simulation, number of processes, CPU burst time, etc.
- After starting the simulator, click on the **Run Experiment** button. This will start the simulation.
- When the simulation is completed, click on the **Draw Gannt Chart** button. A new window will appear with a gannt chart representing how the processes were scheduled.

2. CPU Scheduling Simulation

- In this graph, each horizontal line represent processes. Green parts represent **READY** state (waiting for its turn), red parts represent **RUNNING** state.
- Change the simulation parameters many times and observe how each change affect the simulation itself.
- **Exercise**: Create a sample run with;
    - First-Come/First-Served algorithm,
    - 10 processes,
    - Interarrival varying between 5 and 10,
    - Duration varying between 5 and 60,
    - CPU burst as 5,
    - IO burst as 1.

  Then, change the algorithm to **Shortest Job First** and compare the results.