Topics: CPU Scheduling Examples and Simulator
(SGG[9ed] 6.7 and web notes)

CS 3733 Operating Systems

Instructor: Dr. Turgay Korkmaz

Department Computer Science

The University of Texas at San Antonio

Office: NPB 3.330
Phone: (210) 458-7346
Fax: (210) 458-4437
e-mail: korkmaz@cs.utsa.edu
web: www.cs.utsa.edu/~korkmaz

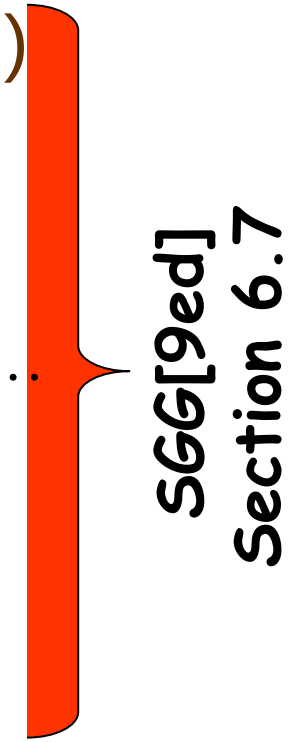
These slides are prepared based on the materials provided by Drs. Robbins, Zhu, and Liu. Thanks to all.

Outline

- Windows Scheduling (VAX VMS Scheduling)
- UNIX Scheduling Algorithm
- Linux Scheduling Algorithm

  Multi-level feedback queues, FCFS, RR, SJF, PR …
  Interactive vs. Background processes,
  Preemptive vs. Non-Preemptive
  Quantum (dynamically adjusted)
  Priorities (dynamically adjusted)

- CPU Scheduling simulator (Java)



Windows NT Scheduling (W2000, and XP)

- Based on VAX VMS Scheduling (next slide)
  - multi-level feedback queues
- Interactive processes: wait for keyboard or mouse
  - Get largest priority increment
- Foreground process in the current window get a larger quantum
- Preemptive even on the dynamic queues

VAX VMS Scheduling

- Use multi-level feedback queue: 32 priority levels

- 16-31 → static real-time queues
  - A process gets its priority when it starts
  - Preemptive according to priorities

- 0-15 → dynamic queues
  - Processes in higher number queue are executed first
  - Non-preemptive with quantum

VAX VMS Scheduling (cont.)

- Each system ***call*** has a priority increment
  - When an event occurs → a process become executable: its priority = base + increment

- When a process is scheduled to run → its priority decreases
  - No lower than the base

- Processes **age**
  - If spend long time in a queue → increase priority

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Windows XP Priorities

UNIX Scheduling Algorithm

- Also use multi-level feedback queues
- A runnable process get a number → which queue
- Lower numbers → higher priority
  - Negative numbers: system processes cannot be killed by signals
- First process in the lowest nonempty queue → run
  - nice to reduce priority
- Time quantum of 0.1 second (100 milliseconds)
- Priorities are re-calculated once a second
- Non-preemptive except for quantum expiration

UNIX Scheduling Algorithm (cont.)

- Some flavors of Unix→ interactive process with window focus gets highest priority
- A process' user-mode priority
  - p_usrpri = P_USER + .25 * p_cpu + 2 * p_nice
  - p_cpu : increased each time the system clock ticks and the process is running
  - p_cpu is adjusted once per second for ready processes using a digital decay filter

$$P_{cpu} = \frac{2\,\text{load}}{2\,\text{load} + 1} P_{cpu} + P_{nice}$$

  - Load → sampled average length of run queue for previous 1 minute interval of system operations

UNIX Scheduling Algorithm (cont.)

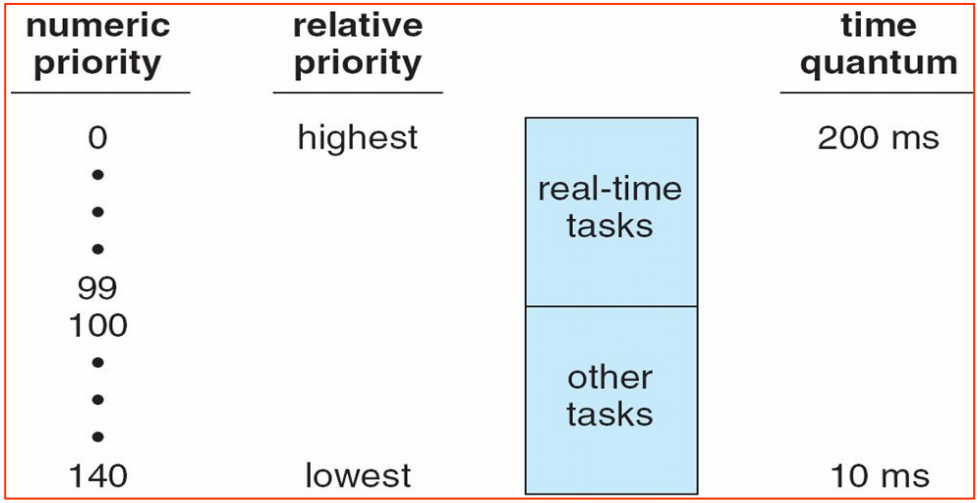
- When a process is blocked for an event, it cannot accumulate CPU time
- When a process sleeps for more than 1 second

$$p_{cpu} = \left[\frac{2\,\mathrm{load}}{2\,\mathrm{load}+1}\right]^{p_{\mathrm{slptime}}} p_{cpu}$$

  - p_slptime is an estimate of how long it is blocked
- **Non-preemptive** process running in **kernel** mode
  - **Not suitable for real-time systems**

Linux Scheduling Algorithm

- Use two separate scheduling algorithms
  - One for time-sharing with focus on fairness
  - One for real-time tasks with absolute priorities



Priorities and Time-slice length

- Time sharing processes: based on a credit system
  - Process has a fixed priority and variable number of credits
  - To choose a process → the one with most credits to run
  - Running process loses one credit per timer interrupt, which is removed from CPU when its credits run out
  - If no process has any credit → credits = credits/2 + priority

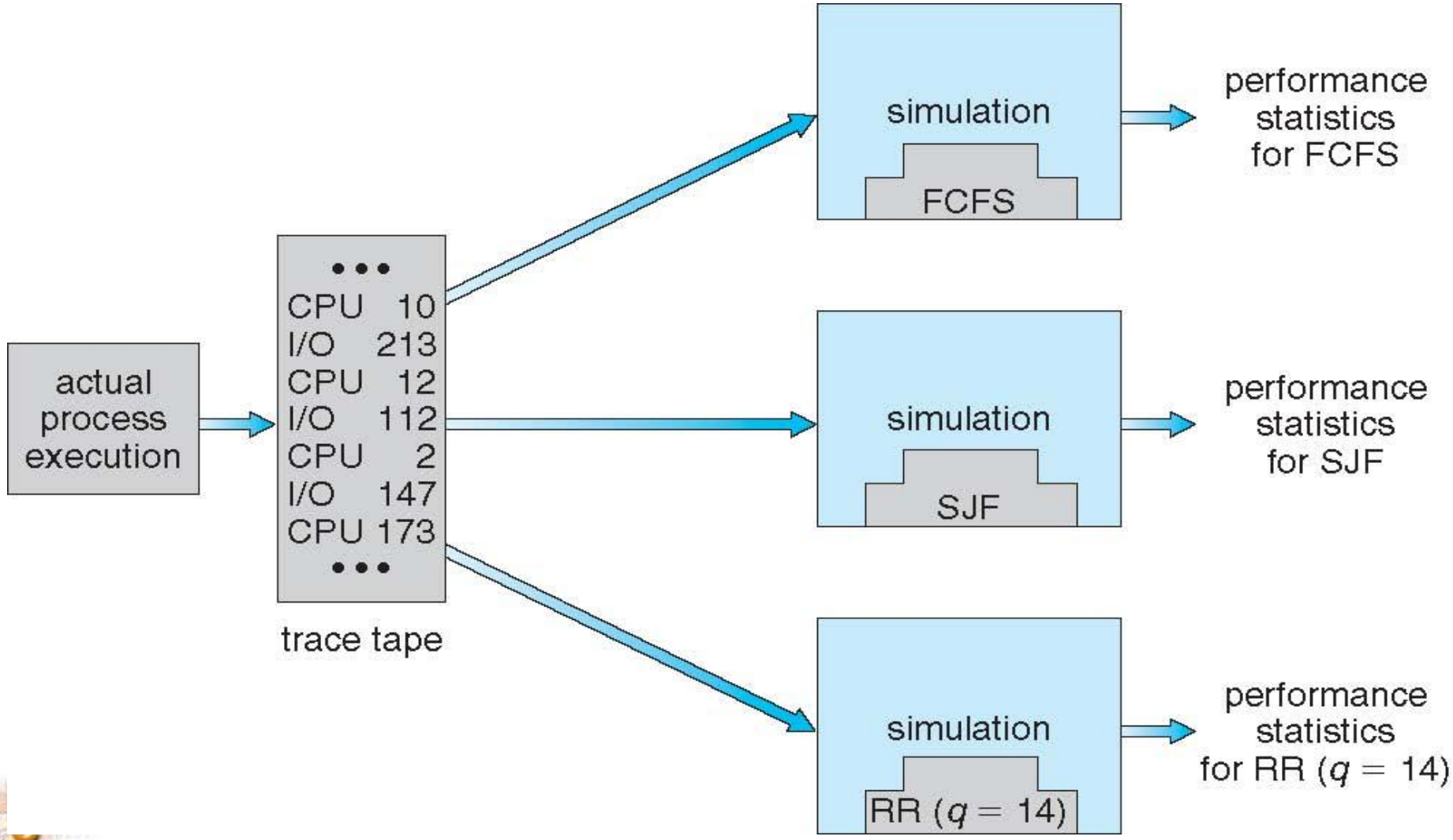
Linux Scheduling Algorithm (cont.)

- **Real-time processes have higher priority than time-sharing processes**
  - Time-sharing tasks run only if no runnable real-time tasks
- Real-Time Scheduling
  - Each process has a priority and a scheduling class
  - Scheduling class: can be FIFO = FCFS or RR
- The highest priority real-time task runs first
- For tasks with the same priority: FCFS
- For FCFS, a process runs until its I/O operation
- RT processes do NOT preempt other processes

http://classque.cs.utsa.edu/classes/cs3733f2015/notes/ps/index.html

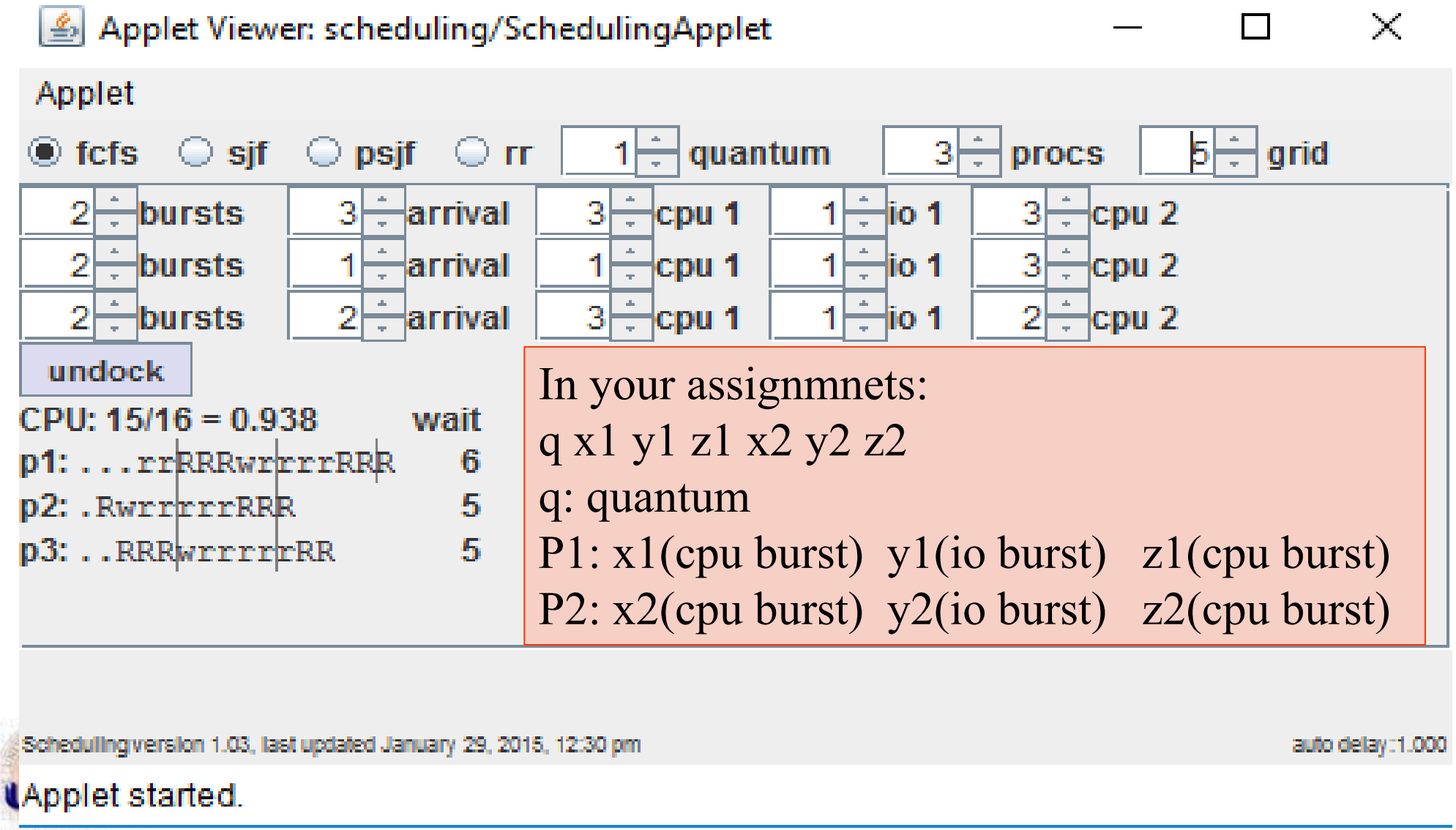
CPU SCHEDULING SIMULATOR (JAVA)

Evaluation of CPU schedulers by Simulation



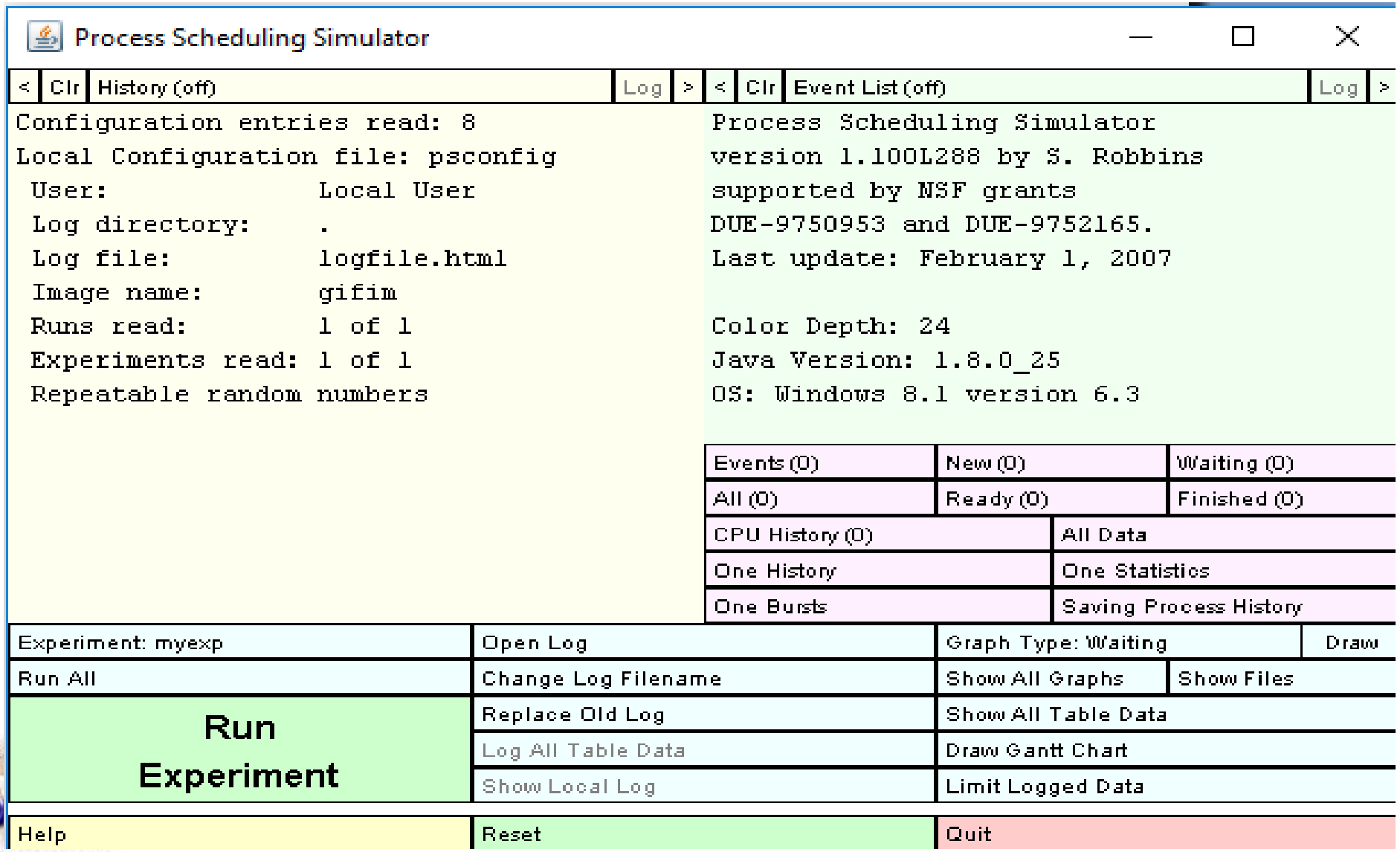
http://classque.cs.utsa.edu/classes/cs3733f2015/notes/ps/index.html

> appletviewer http://classque.cs.utsa.edu/classes/cs3733/scheduling2/index.html



Now we will look at a more general case.

Get http://vip.cs.utsa.edu/simulators/zipfiles/ps.zip



Input needed by a process scheduling simulator:

- Algorithm,
  - FCFS, SJF, SJFA 0.5, PSJF, PSJFA 0.5, RR 10
- Processes
  - arrival time of each process
  - all CPU bursts for each process
  - all I/O bursts for each process
- How would you describe the processes? A group of processes has similar characteristics, described by
  - first arrival time
  - interarrival time distribution, e.g. constant 3
  - cpu bursts distribution, e.g. uniform 10 20
  - I/O bursts distribution, e.g. exponential 15

A RUN

- Running the simulator with a given algorithm and a given collection of processes.
- This is described in a file with extension **.run**
- The file starts with four lines. Each line starts with a key word followed by a value

key word	**value**
name	a single word, the name of the file (without the .run)
comment	any characters, all on one line
algorithm	FCFS, SJF, RR 5, etc.
seed	an integer seed for the random number generator

A RUN (cont'd)

- The rest of the file describes groups of processes.
- A group is described with 7 lines in a similar format.
- There may be as many groups as you like.
- Blank lines are ignored.
- A group looks like this:

key word	value
numprocs	an integer representing the number of processes in this group
firstarrival	a number giving the arrival time of the first process in the group.
interarrival	a distribution giving interarrival times.
duration	a distribution giving total CPU times of a process.
cpuburst	a distribution giving CPU burst of a process.
ioburst	a distribution giving IO burst of a process.
basepriority	a number that is ignored by the simulator.

myrun.run

```
name myrun
comment This contains two types of processes
algorithm SJF
seed 5000

numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration uniform 10.0 15.0
cpuburst constant 10.0
ioburst uniform 10 20
basepriority 1.0

numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration constant 4.0
cpuburst constant 1.0
ioburst uniform 10.0 20.0
basepriority 1.0
```

A RUN (cont'd)

- A distribution is one of the following where x and y are numbers:
  - constant x: always has the value x
  - uniform x y: has a value between x and y (inclusive)
  - exponential x: has an exponential distribution with mean x
- When you perform a run, you get some numbers for CPU utilization, average waiting time, etc.
- These are not too useful unless you have something to compare them to.

Experiment

- An ***experiment*** is a collection of **runs** that have almost the same parameters.
- For example, you might keep everything the same and just change the algorithm FCFS → SJF.

- An experiment is specified by a file that ends in **.exp**.
- The first two lines are similar to that of a run file, giving the name and a comment.
- The additional lines each start with the word *run* and correspond to a given run.

Experiment (cont'd)

myexp.exp

```
name myexp
comment This experiment contains 2  runs
run myrun algorithm FCFS key "FCFS"
run myrun algorithm SJF key "SJF"
```

- Specify a run by naming the run file (without the .run).
- The same run file can be reused, by following it with options that override the contents of the run file.
- The format of the options are similar to the lines of the run file.
- Any number of options can modify a run file.

Configuration

psconfig
logdir Remote
logfn logfile.html
imagename gifim
user Remote Users
portable true
run myrun
exp myexp

- A **configuration** file is also necessary to tell the simulator which **.run** and **.exp** files to use as well as the names and location of the output files.

Get http://vip.cs.utsa.edu/simulators/zipfiles/ps.zip

myrun.run

```
name myrun
comment This contains two types of processes
algorithm SJF
seed 5000
numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration uniform 10.0 15.0
cpuburst constant 10.0
ioburst uniform 10 20
basepriority 1.0

numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration constant 4.0
cpuburst constant 1.0
ioburst uniform 10.0 20.0
basepriority 1.0
```

The example files

psconfig

```
logdir Remote
logfn logfile.html
imagename gifim
user Remote Users
portable true
run myrun
exp myexp
```

myexp.exp

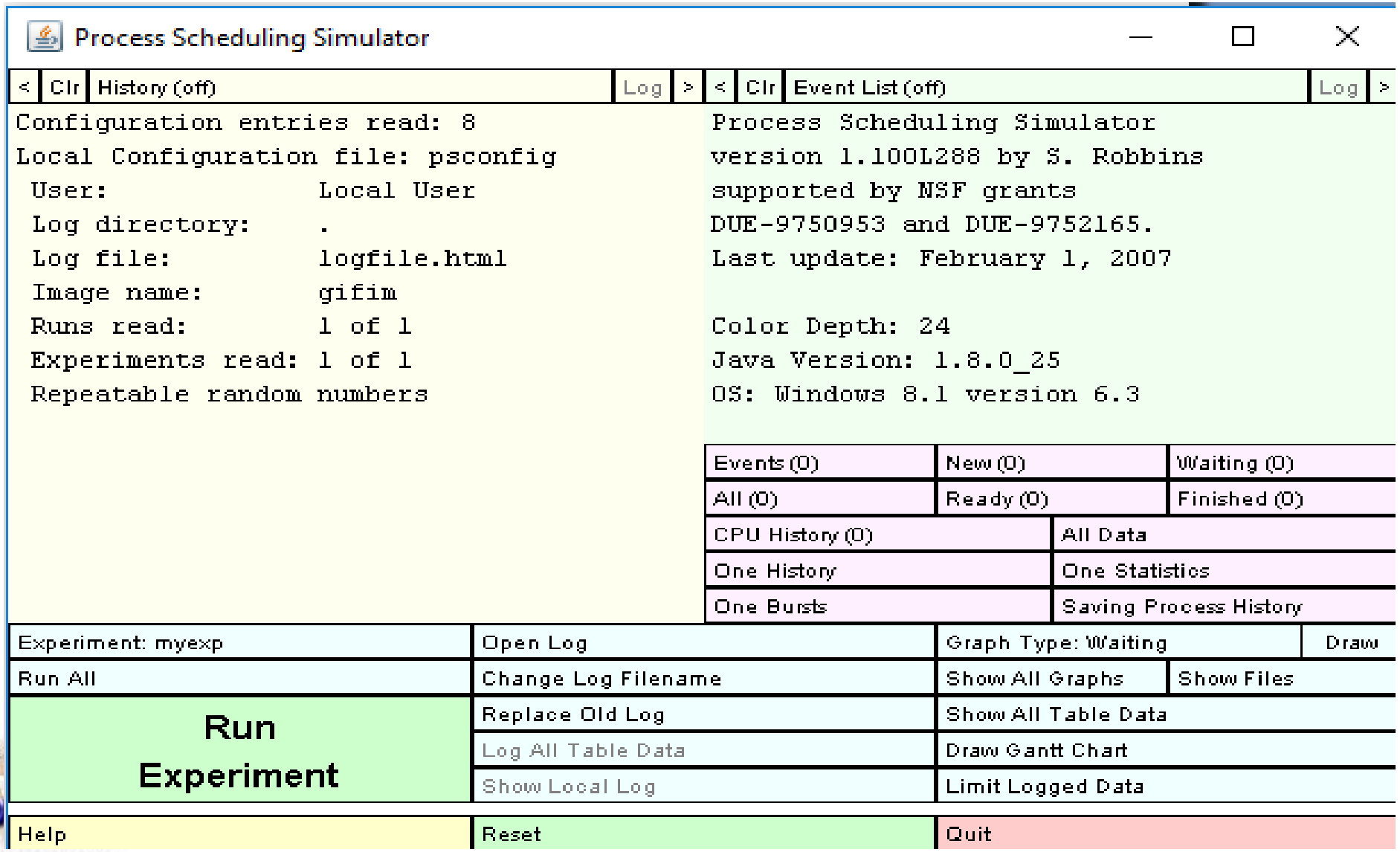
```
name myexp
comment This experiment contains 2  runs
run myrun algorithm FCFS key "FCFS"
run myrun algorithm SJF key "SJF"
```

see ps_doc.html

> runps

Now we will look at a more general case.

Get http://vip.cs.utsa.edu/simulators/zipfiles/ps.zip



Summary

- Windows Scheduling (VAX VMS Scheduling)
- UNIX Scheduling Algorithm
- Linux Scheduling Algorithm

- CPU Scheduling Simulator (Java)