1	CS 460 Scheduling Lab
2	
3	
4	Shutdown VB and change the System to use only 1 CPU. Use only
5	1 CPU in VB for this entire exercise. Boot into any working
6	kernel.
7	
8	
9	Open a terminal
10	-
11	sudo pacman -Sy terminator
12	
13	Open terminator and run top (V)
14	
15	Open terminator and do the following:
16	
17	wget http://zeus.cs.pacificu.edu/chadd/cs460s16/SchedLab.tar.gz
18	
19	tar xzf SchedLab.tar.gz
20	
21	cd CS460 SchedulingLab
22	
23	make
24	
25	This produces a number of executables. We will only use some
26	of these executables today.
27	-
28	PRIORITIES
29	
30	
31	Try out a few of the executables. Note how much work each executable
32	reports it has done.
33	-
34	time ./sleeper 20
35	
36	Work:
37	
38	time ./CPU 20
39	
40	Work:
41	
42	time ./IO 20
43	
44	Work:
45	
46	Each of the previous executables takes a command line argument that
47	is the runtime in seconds for the process. In the above examples,
48	each process should run for very close to 20 seconds.
49	
50	Each executable reports the amount of work done and the number
51	of voluntary and involuntary context switches done by that process.
52	
	steeper just continually calls steep(1) until the runtime is expired.
54   55	(DII mund a for loop and dood some calculations with the mustice
55	ero runs a for roop and does some carculations until the runtime
20	IS EXPILED.
57	TO rung a for loop and prints data to atdorp until the mustime is
50	TO TUNS a TOT TOOP and PITHES GALA TO SEGRET UNITE THE FUNCTIME IS
59	evhtten.
61	
62	Onen a terminator window Stretch to fill the screen width
UZ	open a commutator window. Derecen to itit the Sereen width.
1	

5/5/2016 9:41:21 PM

C:\Users\chadd\box\Box Sync\CS460\SchedulingLab.txt Page 1 of 5

C:\Use Page 2	ers\chadd\box\Box Sync\CS460\SchedulingLab.txt 2 of 5 5/5/2016 9:41:21 Pt
<b>_</b>	
63	Right Click, Split vertical
64	On the right half, Right Click, Split vertical
65	On the left half, Right Click, Split vertical
66	
67	Now you have 4 terminals in one window.
68	
69	In each of the four terminals, type one of the following commands
/0	but don't press enter.
/1	
72	time ./sleeper 20
73	
74	time /CDU 20
75	CINE ./CPU 20
70	
70	time /CPU 20
70	CIME ./CFO 20
80	
81	time /TO 20
82	
83	
84	
85	Click the little tiny red/blue/white squares in the top left of any
86	of the 4 terminal windows.
87	
88	Select Broadcast all.
89	
90	Press Enter. This will send the enter command to each terminal
91	and run each command at nearly the same time.
92	
93	Note how much work each executable has done. How does this compare
94	to the original work completed when only one executable is running?
95	Why?
96	
97	
98	
99	Without typing Enter, in any one of the terminals, type
100	time ./CPU 30
101	This should appear in all four terminals
102	This should appear in all four cerminals.
104	Open another terminator window (you should have two single windows and
105	ope four way window open)
106	one roar way window open,.
107	Press enter in the 4 way window.
108	
109	In the window running top, look at how much (%) of the CPU each
110	CPU process is getting.
111	
112	
113	Nice is a command to raise or lower the priority of a process. Only
114	root can raise the priority. You can always lower your priority.
115	
116	Let's rerun the CPU process in the 4 way window and nice down a process
117	to give that process a lower priority.
110	
120	Start time ./CPU 30 in all four terminals.
101	Find the DID of any CDU process in ten
121 122	Find the FiD of any CFD process in top.
122	In another terminator window.
123	sudo renice -n 10 -n PID
1	- 2 -

C:\Us Page	ers\chadd\bo 3 of 5	x\Boz	<pre>sync</pre>	cs46	50\Sche	duling	JLat	o.txt	2					5	5/5,
125															
126	Watch	top.	Does	that	PID's	share	of	the	CPU	drop?	Do	you	see	the	PR
127	or NI	chan	ge in	top?											
128															

129 Drat! 130 131 We have to turn off autogroup for nice to work properly. 132 133 cat /proc/sys/kernel/sched autogroup enabled 134 su # become root 135 echo 0 > /proc/sys/kernel/sched autogroup enabled # does not stick past reboot! 136 exit # leave root 137 138 Now, restart all 4 CPU processes, find one in top and 139 sudo renice -n 10 -p PID 140 141 Watch top. Does that PID's share of the CPU drop? Do you see the PR 142 or NI change in top? 143 144 Description of autogroups. 145 https://oakbytes.wordpress.com/2012/06/06/linux-scheduler-cfs-and-nice/ 146 http://serverfault.com/questions/405092/nice-level-not-working-on-linux 147 https://en.wikipedia.org/wiki/Completely Fair Scheduler#Fairer algorithms 148 http://forum.osdev.org/viewtopic.php?f=15&t=25612 149 150 You can also nice your own process at launch. 151 152 Click the little tiny red/blue/white squares in the top left of any 153 of the 4 terminal windows. 154 155 Select Broadcast off. 156 157 Type, but don't hit enter, these commands into 158 159 time sudo nice -n 0 ./CPU 40 160 time sudo nice -n -20 ./CPU 40 161 time sudo nice -n 10 ./CPU 40 162 time sudo nice -n 19 ./CPU 40 163 164 Click the little tiny red/blue/white squares in the top left of any 165 of the 4 terminal windows. 166 Select Broadcast on. 167 168 169 In the window running top: 170  $\cap$ 171 COMMAND=CPU 172 173 How much work did each process get done? 174 175 Arrow up in the 4 way window to run these again. 176 177 178 179 SCHEDULING 180 181 Now, let's play with scheduling algorithms. 182 183 Linux has a number of scheduling algorithms available: 184 185 Real time processes: 186 SCHED FIFO

5/5/2016 9:41:21 PM

C:\Users\chadd\box\Box Sync\CS460\SchedulingLab.txt Page 4 of 5

187	SCHED RR	
188		
189	Everything else:	
190	SCHED_OTHER	
191	SCHED BATCH	
192	SCHED IDLE	
193		
194	Read the man page for sched to understand each algorithm.	
195		
196	Step 0:	
197		
198	schedTest launches 5 threads (via pthreads) and sets the scheduling	
199	policy based on the command line argument given. R is RR and F is FIFO.	
200	Each thread prints 10 messages containing the thread ID, a progress	
201	number, and a time stamp.	
202		
203	At the end, schedTest prints out the number of voluntary and involuntary	
204	context switches that occurred.	
205		
206	Note: all output happens just before a process terminates so as to not	
207	generate any extra context switches via printf.	
208		
209	time sudo ./schedTest R	
210		
211	time sudo ./schedTest F	
212		
213	Does it seem like the scheduler is working correctly? Justify your answer.	
214		
215	# Bonus: do the following if you have time after completing Step 1.	
216	# Restart with 2 CPUs and run the above tests again. Any difference?	
217	# Restart with 1 CPU and continue.	
210		
218		
218	Step 1:	
218 219 220	Step 1:	
218 219 220 221	Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority	
218 219 220 221 222	Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority	
218 219 220 221 222 223	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not</pre>	
218 219 220 221 222 223 224	Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf.	
218 219 220 221 222 223 224 225 226	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf.</pre>	
218 219 220 221 222 223 224 225 226 227	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argy[1]. Prior PD and Fright FLEO. Prior PARCY. Trigo TPUE, Origo policy based on argy[1]. Prior PD and Fright FLEO. Prior PARCY.</pre>	-
218 219 220 221 222 223 224 225 226 227	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER</pre>	₽
218 219 220 221 222 223 224 225 226 227	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER.</pre>	
218 219 220 221 222 223 224 225 226 227 228 229	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER.</pre>	₽
218 219 220 221 222 223 224 225 226 227 228 229 230	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp</pre>	₽
218 219 220 221 222 223 224 225 226 227 228 229 230 231	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp.</pre>	₽
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via:</pre>	₽
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[21) + 1 # threadid starts at 0 and increments by 1</pre>	<u> </u>
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1</pre>	P
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context</pre>	R
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed.</pre>	P
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed.</pre>	R
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually.</pre>	₽
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually.</pre>	<u>∼</u>
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must</pre>	
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges.</pre>	Ð
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges.</pre>	₽
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges. time sudo ./schedTestFork R 1 2</pre>	R
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges. time sudo ./schedTestFork R 1 2</pre>	P
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges. time sudo ./schedTestFork R 1 2 time sudo ./schedTestFork R 98 2</pre>	
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges. time sudo ./schedTestFork R 1 2 time sudo ./schedTestFork R 98 2</pre>	₽
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247	<pre>Step 1: Note: RT scheduling priorities run from 1-99. 99 is highest priority Note: all output happens just before a process terminates so as to not generate any extra context switches via printf. schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp. argv[2] sets the priority of each process via: (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1 As each process ends, the counts of voluntary and involuntary context switches are listed. Run each of the following command individually. Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges. time sudo ./schedTestFork R 98 2 time sudo ./schedTestFork F 1 2</pre>	

C:\Use Page !	ers\chadd\box\Box Sync\CS460\SchedulingLab.txt 5 of 5 5 5/5/2016 9:41:21 PM
248	
249	time sudo ./schedTestFork F 98 2
250	
251	
252	Does it seem like the scheduler is working correctly? Justify your answer.
253	
254	
255	What happens to other processes in VB while these run? Geany? Firefox, etc?
256	
257	Start in 4 way terminator window to see how processes with different
258	scheduling algorithms interact.
259	
260	(approx 5 min).
261	time sudo ./schedTestFork R 1 2
262	time sudo ./schedTestFork R 1 2
263	time sudo ./schedTestFork R 1 2
264	time sudo ./schedTestFork R 1 2
265	
266	time sudo ./schedTestFork R 1 2
267	time sudo ./schedTestFork R 98 2
268	time sudo ./schedTestFork R 1 2
269	time sudo ./schedTestFork 0 1 2
270	
271	time sudo ./schedTestFork R 1 2
272	time sudo ./schedTestFork R 98 2
273	time sudo ./schedTestFork R 1 2
274	time sudo ./schedTestFork 1 1 2
213	time and (achedmostEarly D 1 2
∠/७ 277	time sude (achedTestFork R 1 2
211 279	time sude /schedTestFork E 08 2
2/0 270	time sude /schedTestFork I 1 2
219	LINE SUGO ./SCHEQIESTFOIK I I Z
200 281	time sudo /schedTestFork P 1 2
201 282	time sudo /schedTestFork R 98 2
202	time sudo /schedTestFork F 98 2
284	time sudo /schedTestFork B 1 2
285	
286	Does the output match your expectation?
287	
288	
200	