

LAB #2: MODIFYING XV6 SCHEDULER

Group Members:

- Yash Aggarwal (yagga004)
- Nityash Gautam (ngaut006)
- Parth Bhatt (pbhat029)

Demonstration Link:

[Click Here](#)

PART 1: SYSTEM CALLS

Similar to the previous Lab Assignment, System Calls have been added.

1. 2 new syscall numbers have been defined at **kernel/syscall.h**
2. The syscall table at **kernel/syscall.c** has been updated with respect to the new syscalls added.
3. A syscall function “**sys_sched_statistics**”, has been defined at **kernel/sysproc.c**.
 - a. **sched_statistics** iterates all the processes in the process table, and prints the results as per the UNUSED processes.
4. “**sys_sched_statistics**” is the return to the kernel function “**sched_statistics()**”.
5. The argument passed into the “**sys_sched_tickets**” is the ticket assigned to the process, and is adjusted to the range (0, MAX_NUM_TICKET_PER_PROCESS).
6. After this, the kernel function “**sched_tickets**” is called.
7. “**sched_tickets(tickets)**” sets the caller’s process tickets to the desired value.
8. Further, inside the **scheduler()**, if the process is chosen, the number of ticks is incremented.
9. The function is defined at **kernel/defs.h**
10. The user space is updated at **user/usys.pl** and **usys/user.h**
11. The “**\$U_lab2_l**” is added to the **Makefile** and the CPU is changed to 1.

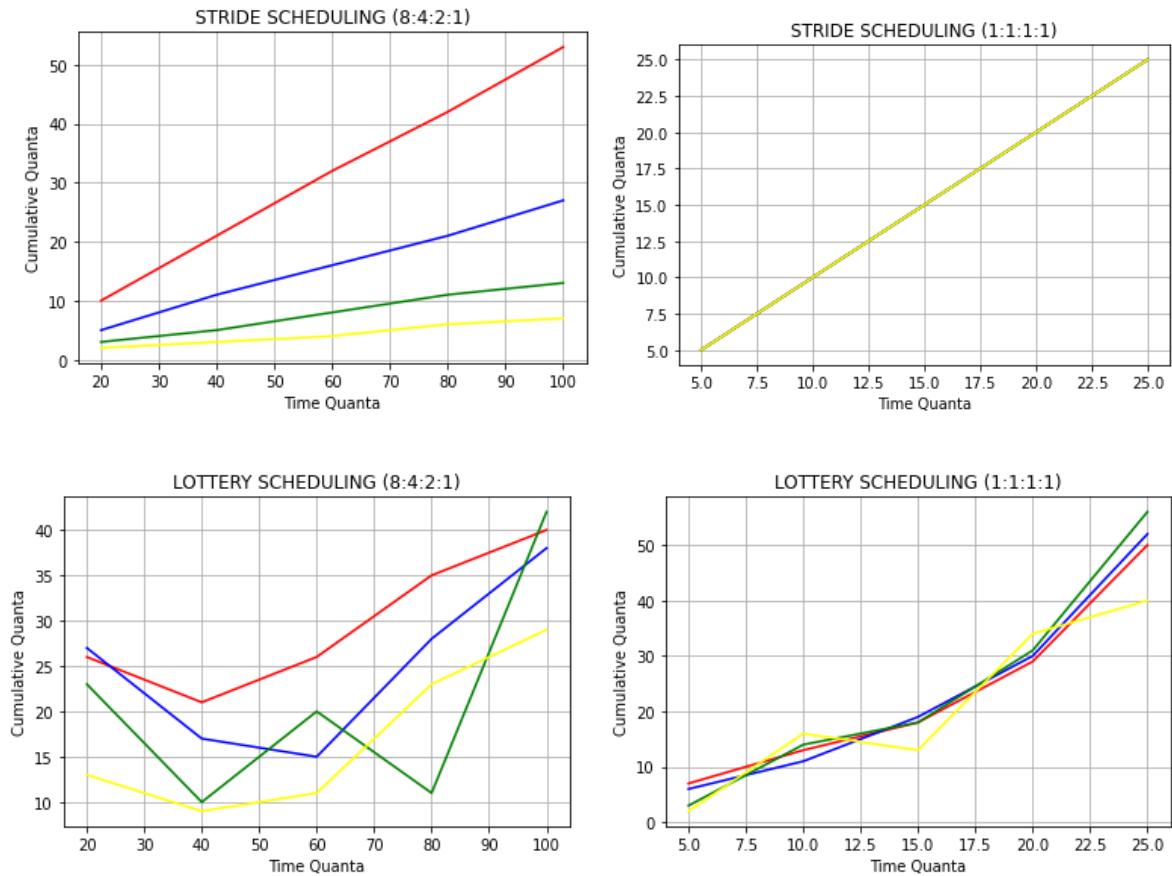
PART 2: SCHEDULER IMPLEMENTATION

The Scheduler Implementation has been done following the below-mentioned steps:

1. 2 Variables (strides & pass) are added in the **struct proc** at the **kernel/proc.h** file.
2. At the **kernel/proc.c**, the “**rand() function**” is added as per the Lab2 Instructions provided.
3. Inside the “**allocproc()**” function in the **kernel/proc.c** file, a new variable of struct proc is initialized before the process is returned.
4. In the **kernel/proc.c** file, inside the scheduler function, we use **#if define...#end** per instruction
 - a. Inside the infinite Loop, the original second loop is moved to the **#else** branch.
 - b. Within the “**#if defined(LOTTERY)**” branch:
 - i. The process table is iterated to find the runnable processes.

- ii. Total tickets of the Runnable processes are calculated.
- iii. Finally, a random number is generated within the range.
- iv. Also, the total number of tickets till this point “*tickets_sum*” is calculated.
- v. If “*tickets_sum*” is less than the LOTTERY, the next runnable process is executed. Also, the number of ticks is increased, as and when the runnable process is selected.
- c. Within the “#if defined(STRIKE)” branch:
 - i. The process table is iterated to find the process with the minimum pass.
 - ii. Once found, the process’s pass value is incremented by strides (K/#ticket, at p->strides)
 - iii. Once done, we switch and increase ticks as normal.

PART 3: EXPERIMENTS



From the results obtained above, it can easily be observed that the LOTTERY SCHEDULING algorithm shows significant variability on the given time scales.

This is the result of the fact that Lottery Scheduling uses a Probabilistic technique for ticket scheduling.

Observing the deterministic nature of the stride scheduling algorithm, we observe the results produced by it are periodic in nature.

CONCLUSION: The stride Scheduling Algorithm is more suitable than Lottery Scheduling, because of its deterministic nature of scheduling strides.

CONTRIBUTIONS

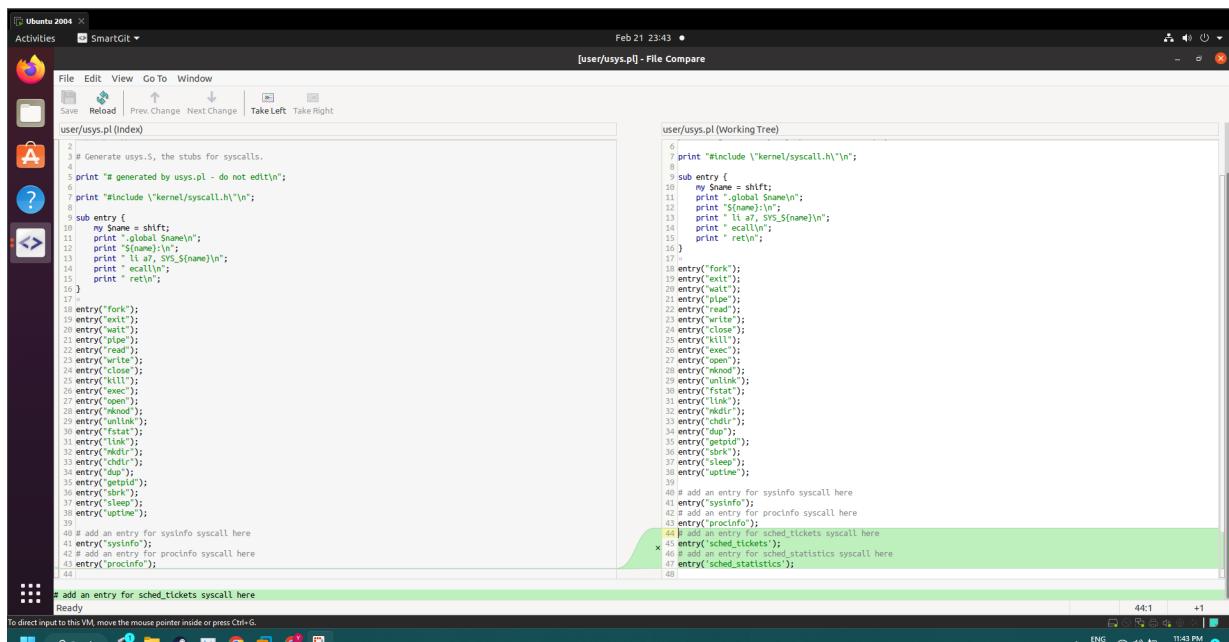
Yash Aggarwal: Design and implementation of Lottery Scheduling.

Parth Bhatt: Design and implementation of Stride Scheduling.

Nityash Gautam: Implementing the System Calls and Experimentation.

Note*: The above-mentioned Contributions are not strict. Every member contributed to the working implementation of the mentioned sections of the Lab Assignments.

CODE CHANGES



```
File Edit View Go To Window
Save Reload Prev Change Next Change Take Left Take Right
user/usys.pl (index)
2
3 # Generate usys.S, the stubs for syscalls.
4
5 print "# generated by usys.pl - do not edit\n";
6
7 print "#include <kernel/syscall.h>\n";
8
9 sub entry {
10     my $name = shift;
11     print "#global $name\n";
12     print "#include <asm-$name.h>\n";
13     print "#define $name Sys,$(name)\n";
14     print "#    .ecall\n";
15     print "#    .ret\n";
16 }
17
18 entry("fork");
19 entry("exit");
20 entry("wait");
21 entry("pipe");
22 entry("read");
23 entry("write");
24 entry("close");
25 entry("kill");
26 entry("exec");
27 entry("mmap");
28 entry("mmap2");
29 entry("unlink");
30 entry("dup");
31 entry("link");
32 entry("rddir");
33 entry("chdir");
34 entry("setSIG");
35 entry("getpid");
36 entry("shbk");
37 entry("sleep");
38 entry("setSIG");
39
40 # add an entry for sysinfo syscall here
41 entry("sysinfo");
42 # add an entry for procinfo syscall here
43 entry("procinfo");
44
# add an entry for sched_ticks syscall here
Ready
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
Feb 21 23:43 • [user/usys.pl] - File Compare
use/usys.pl (Working Tree)
6
7 print "#include <kernel/syscall.h>\n";
8
9 sub entry {
10     my $name = shift;
11     print "#global $name\n";
12     print "#include <asm-$name.h>\n";
13     print "#define $name Sys,$(name)\n";
14     print "#    .ecall\n";
15     print "#    .ret\n";
16 }
17
18 entry("fork");
19 entry("exit");
20 entry("wait");
21 entry("close");
22 entry("read");
23 entry("write");
24 entry("mmap");
25 entry("mmap2");
26 entry("unlink");
27 entry("dup");
28 entry("link");
29 entry("chdir");
30 entry("setSIG");
31 entry("getpid");
32 entry("shbk");
33 entry("sleep");
34 entry("setSIG");
35
36 # add an entry for sysinfo syscall here
37 entry("sysinfo");
38 # add an entry for procinfo syscall here
39 entry("procinfo");
40 # add an entry for sched_ticks syscall here
41 entry("sched_ticks");
41 # add an entry for sched_statistics syscall here
42 entry("sched_statistics");
43
44 entry("sched_statistics");
45
```

Ubuntu 20.04 • Feb 21 23:44 • [user/user.h] - File Compare

```

File Edit View Go To Window
Save Reload Prev.Change Next.Change Take Left Take Right
user/user.h (Index)
7 int exit(int) __attribute__((noreturn));
8 int wait(int);
9 int pipe(int);
10 int read(int, void*, int);
11 int write(int, const void*, int);
12 int close(int);
13 int kill(int);
14 int execv(const char*, char**);
15 int open(const char*, int);
16 int mknod(const char*, short, short);
17 int link(const char*, const char*);
18 int unlink(const char*);
19 int fstat(int fd, struct stat*);
20 int lseek(int, off_t, int);
21 int dup(int);
22 int dup2(int);
23 int getopt(void);
24 char* strerror(int);
25 int sleep(int);
26 int uptime(void);
27
28 // define prototype of sysinfo syscall
29 #include <sys/types.h>
30 // define prototype of procinfo syscall
31 #include <sys/conf.h>
32
33 // define prototype of sched_statistics syscall
34 #include <sys/conf.h>
35
36 // define prototype of sched_tickets syscall
37
38 // libc
39 int stat(const char*, struct stat*);
40 char* strcpy(char*, const char*);
41 void *memchr(const void*, int, size_t);
42 char* strchr(const char*, char c);
43 int strcmp(const char*, const char*);
44 void sprintf(int, const char*, ...);
45 void vsprintf(int, const char*, va_list);
46 char* gets(char*, int max);
47 uint strlen(const char*);
48 void *memset(void*, int, uint);
49 void *valloc(uint);
50 void *free(void*);
51 int atoi(const char*);
52 int memcmp(const void*, const void*, uint);
53 void *memcpy(void*, const void*, uint);
54
55
56 // define prototype of sched_statistics syscall
57
Ready
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

32:1 +2 ENG 11:44 PM

Ubuntu 20.04 • Feb 21 23:44 • [kernel/sysproc.c] - File Compare

```

File Edit View Go To Window
Save Reload Prev.Change Next.Change Take Left Take Right
kernel/sysproc.c (Index)
77 {
78     int pid;
79
80     argint(0, &pid);
81     return kill(pid);
82 }
83
84 // return how many clock tick interrupts have occurred
85 // since start.
86 uint64
87 sys_uptime(void)
88 {
89     uint64 ticks;
90
91     acquire(&tickslock);
92     ticks = ticks;
93     release(&tickslock);
94     return ticks;
95 }
96
97 // body of defined syscall function.
98 // call a function in process to execute.
99 // get first argument passed to the system call
100 // pass the argument to process level function
101 uint64
102 sys_sysinfo(void)
103 {
104     int n;
105     argint(0, &n);
106     return get_sys_sysinfo(n, global_sys_calls_counter);
107 }
108
109 // body of defined syscall function.
110 // call a function in process to execute.
111 // get first argument passed to the system call
112 // pass the argument to process level function
113 uint64
114 sys_procinfo(void)
115 {
116     uint64 pinfo_pointer; // user pointer to struct pinfo
117     argaddr(0, &pinfo_pointer);
118     return get_sys_procinfo(pinfo_pointer);
119 }

Ready
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

119:1 +1 ENG 11:44 PM

Ubuntu 2004 Feb 21 23:44

Activities SmartGit File Edit View Go To Window

[kernel/syscall.h] - File Compare

File | Save | Reload | Prev Change | Next Change | Take Left | Take Right

kernel/syscall.h (index)

```

1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_reboot 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 // define the syscall here, assign it a number and define the corresponding function in syscall.c
24 #define SYS_sysinfo 22
25 // define the syscall here, assign it a number and define the corresponding function in syscall.c
26 #define SYS_procinfo 23

```

Ready

To direct input to this VM, move the mouse pointer inside or press Ctrl-G.

kernel/syscall.h (Working Tree)

```

1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_reboot 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 // define the syscall here, assign it a number and define the corresponding function in syscall.c
24 #define SYS_sysinfo 22
25 // define the syscall here, assign it a number and define the corresponding function in syscall.c
26 #define SYS_procinfo 23
27 // define the syscall here, assign it a number and define the corresponding function in syscall.c
28 #define SYS_sched_statistics 24
29 // define the syscall here, assign it a number and define the corresponding function in syscall.c
30 #define SYS_sched_tickets 25

```

26:27 -1 ENG 11:44 PM

Ubuntu 2004 Feb 21 23:45

Activities SmartGit File Edit View Go To Window

[kernel/syscall.c] - File Compare

File | Save | Reload | Prev Change | Next Change | Take Left | Take Right

kernel/syscall.c (index)

```

102 extern uint64 sys_mkdir(void);
103 extern uint64 sys_close(void);
104 // add mapped syscall function here.
105 extern uint64 sys_sysinfo(void);
106 // map the defined syscall to a user function.
107 extern uint64 sys_procinfo(void);
108
109 // ---- system level variables ----
110 // store the total system calls made by the system
111 extern uint64 global_sys_calls_counter;
112
113 // An array mapping syscall numbers from syscall.h
114 // to the function that handles the system call.
115 static uint64 (*syscalls[])(void) = {
116     [SYS_fork] sys_fork,
117     [SYS_exit] sys_exit,
118     [SYS_wait] sys_wait,
119     [SYS_pipe] sys_pipe,
120     [SYS_read] sys_read,
121     [SYS_kill] sys_kill,
122     [SYS_exec] sys_exec,
123     [SYS_fstat] sys_fstat,
124     [SYS_chdir] sys_chdir,
125     [SYS_dup] sys_dup,
126     [SYS_getpid] sys_getpid,
127     [SYS_sbrk] sys_sbrk,
128     [SYS_sleep] sys_sleep,
129     [SYS_uptime] sys_uptime,
130     [SYS_open] sys_open,
131     [SYS_write] sys_write,
132     [SYS_mknod] sys_mknod,
133     [SYS_unlink] sys_unlink,
134     [SYS_link] sys_link,
135     [SYS_mkdir] sys_mkdir,
136     [SYS_close] sys_close,
137     [SYS_sysinfo] sys_sysinfo, // map the defined syscall to a user function.
138     [SYS_procinfo] sys_procinfo, // map the defined syscall to a user function.
139 };
140
141 void
142 syscall(void)
143 {
144     ...
145     [SYS_sched_statistics] sys_sched_statistics, // map the defined syscall to a user function.
146     [SYS_sched_tickets] sys_sched_tickets, // map the defined syscall to a user function.
147 }
148 void

```

kernel/syscall.c (Working Tree)

```

105 // Add mapped syscall function here.
106 extern uint64 sys_procinfo(void);
107 // Add mapped syscall function here.
108 extern uint64 sys_sched_statistics(void);
109 // An array mapping syscall numbers from syscall.h
110 // to the function that handles the system call.
111 extern uint64 sys_sched_tickets(void);
112
113 // ---- system level variables ----
114
115 // store the total system calls made by the system
116 extern uint64 global_sys_calls_counter;
117
118 // An array mapping syscall numbers from syscall.h
119 // to the function that handles the system call.
120 static uint64 (*syscalls[])(void) = {
121     [SYS_fork] sys_fork,
122     [SYS_exit] sys_exit,
123     [SYS_wait] sys_wait,
124     [SYS_pipe] sys_pipe,
125     [SYS_read] sys_read,
126     [SYS_kill] sys_kill,
127     [SYS_exec] sys_exec,
128     [SYS_fstat] sys_fstat,
129     [SYS_chdir] sys_chdir,
130     [SYS_dup] sys_dup,
131     [SYS_getpid] sys_getpid,
132     [SYS_sbrk] sys_sbrk,
133     [SYS_sleep] sys_sleep,
134     [SYS_uptime] sys_uptime,
135     [SYS_open] sys_open,
136     [SYS_write] sys_write,
137     [SYS_mknod] sys_mknod,
138     [SYS_unlink] sys_unlink,
139     [SYS_link] sys_link,
140     [SYS_mkdir] sys_mkdir,
141     [SYS_close] sys_close,
142     [SYS_sysinfo] sys_sysinfo, // map the defined syscall to a user function.
143     [SYS_procinfo] sys_procinfo, // map the defined syscall to a user function.
144     [SYS_sched_statistics] sys_sched_statistics, // map the defined syscall to a user function.
145     [SYS_sched_tickets] sys_sched_tickets, // map the defined syscall to a user function.
146 };
147
148 void

```

140:1 +2 ENG 11:45 PM

Ubuntu 2004 • Feb 21 23:45

Activities SmartGit ▾

[kernel/proc.h] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

kernel/proc.h (Index)

```

67 /* 184 */ utrte64 s3;
68 /* 192 */ utrte64 s4;
69 /* 196 */ utrte64 s5;
70 /* 208 */ utrte64 s6;
71 /* 216 */ utrte64 s7;
72 /* 224 */ utrte64 s8;
73 /* 232 */ utrte64 s9;
74 /* 240 */ utrte64 s10;
75 /* 248 */ utrte64 s11;
76 /* 256 */ utrte64 s12;
77 /* 264 */ utrte64 s13;
78 /* 272 */ utrte64 s15;
79 /* 280 */ utrte64 s16;
80 };
81
82 enum procsstate { UNUSED, USED, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
83
84 // Per-process state
85 struct proc {
86     struct spinlock lock;
87
88     // p-lock must be held when using these:
89     enum procsstate state; // Process state
90     void *chan; // If non-zero, sleeping on chan
91     int killed; // If non-zero, have been killed
92     int exitstatus; // Exit status to be returned to parent's wait
93     int pid; // Process ID
94
95     // wait_lock must be held when using this:
96     struct proc *parent; // Parent process
97
98     // these are private to the process, so p>lock need not be held.
99     uint64 stack; // Virtual address of kernel stack
100    uint64 sz; // Size of process memory (bytes)
101    pagetable_t pagetable; // User page table
102    struct trapframe *trapframe; // Data page for trampoline.S
103    int lfsr; // Pseudo random number generator
104    struct file *ofile[NFILE]; // Open files
105    struct inode *cd; // Current directory
106    char name[16]; // Process name (debugging)
107    int sys_call_count; // Total system calls made by the process
108 };
109

```

kernel/proc.h (Working Tree)

```

80 };
81
82 // Constants to be defined as per lab
83 // Large constant value
84 #define LARGE_CONSTANT_K 10000
85
86 // max tickets allowed
87 #define MAX_TICKETS 10000
88
89 enum procsstate { UNUSED, USED, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
90
91 // Per-process state
92 struct proc {
93     struct spinlock lock;
94
95     // p-lock must be held when using these:
96     enum procsstate state; // Process state
97     void *chan; // If non-zero, sleeping on chan
98     int killed; // If non-zero, have been killed
99     int exitstatus; // Exit status to be returned to parent's wait
100    int pid; // Process ID
101
102    // wait_lock must be held when using this:
103    struct proc *parent; // Parent process
104
105    // these are private to the process, so p-lock need not be held.
106    uint64 kstack; // Virtual address of kernel stack
107    uint64 sz; // Size of process memory (bytes)
108    pagetable_t pagetable; // User page table
109    struct trapframe *trapframe; // Data page for trampoline.S
110    struct context context; // switch() here to run process
111    struct file *ofile[NFILE]; // Open files
112    struct inode *cd; // Current directory
113    char name[16]; // Process name (debugging)
114    int sys_call_count; // Total System calls made by the process
115
116    int tickets; // Count of tickets of the process
117
118    int tick; // Current tick value
119    int strides; // Current stride value
120    int pass; // Current pass value
121 };
122

```

Ready

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

108:1 +2

ENG 11:03 PM

Ubuntu 2004 • Feb 21 23:46

Activities SmartGit ▾

[kernel/proc.c] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

kernel/proc.c (Index)

```

8
9 struct cpus[NCPU];
10
11 struct proc proc[NPROC];
12
13 struct proc *initproc;
14
15 int nextpid = 1;
16 struct spinlock pid_lock;
17
18 extern void forkret(void);
19 static void freeproc(struct proc *p);
20
21 extern char trampoline[]; // trampoline.S
22
23 // Helps ensure that wakeups of wait()ing
24 // parents are not lost, helps obey the
25 // memory model when using p->parent.
26 // must be acquired before any p->lock.
27 struct spinlock wait_lock;
28
29 // Allocate a page for each process's kernel stack.
30 // Map it high in memory, followed by an invalid
31 // page.
32 void
33 proc_mapstacks(pagetable_t kpgtbl)
34 {
35     struct proc *p;
36
37     for(p = proc; p < &proc[NPROC]; p++) {
38         char *pa = kalloc();
39         if(pa == 0) {
40             panic("kalloc");
41             uint64 va = KSTACK((int)(p - proc));
42             kvmmap(kpgtbl, va, (uint64)pa, PGSIZE, PTE_R | PTE_W);
43         }
44     }
45
46     // Initialize the proc table.
47 void
48 procinit(void)
49 {
50     // ...
51
52     // psuedo random number generator
53
54     Ready
55
56
57
58
59
59 // pseudo random number generator
60
61
62
63
64
65
66
67
68
69
69 // psuedo random number generator
70
71
72
73
74
75
76
77
78
79
79 // psuedo random number generator
80
81
82
83
84
85
86
87
88
89
89 // psuedo random number generator
90
91
92
93
94
95
96
97
98
99
99 // psuedo random number generator
100
101
102
103
104
105
106
107
108
109
109 // psuedo random number generator
110
111
112
113
114
115
116
117
118
119
119 // psuedo random number generator
120
121
122
123
124
125
126
127
128
129
129 // psuedo random number generator
130
131
132
133
134
135
136
137
138
138 // psuedo random number generator
139
140
141
142
143
144
145
146
147
147 // psuedo random number generator
148
149
150
151
152
153
154
155
156
156 // psuedo random number generator
157
158
159
160
161
162
163
164
165
165 // psuedo random number generator
166
167
168
169
170
171
172
173
174
174 // psuedo random number generator
175
176
177
178
179
179 // psuedo random number generator
180
181
182
183
184
185
186
187
187 // psuedo random number generator
188
189
190
191
192
193
194
195
195 // psuedo random number generator
196
197
198
199
200
201
202
203
203 // psuedo random number generator
204
205
206
207
208
209
210
211
211 // psuedo random number generator
212
213
214
215
216
217
218
219
219 // psuedo random number generator
220
221
222
223
224
225
226
227
227 // psuedo random number generator
228
229
230
231
232
233
234
235
235 // psuedo random number generator
236
237
238
239
240
241
242
243
243 // psuedo random number generator
244
245
246
247
248
249
250
251
251 // psuedo random number generator
252
253
254
255
256
257
258
259
259 // psuedo random number generator
260
261
262
263
264
265
266
267
267 // psuedo random number generator
268
269
270
271
272
273
274
275
275 // psuedo random number generator
276
277
278
279
280
281
282
283
283 // psuedo random number generator
284
285
286
287
288
289
290
291
291 // psuedo random number generator
292
293
294
295
296
297
298
299
299 // psuedo random number generator
300
301
302
303
304
305
306
307
307 // psuedo random number generator
308
309
310
311
312
313
314
315
315 // psuedo random number generator
316
317
318
319
320
321
322
323
323 // psuedo random number generator
324
325
326
327
328
329
330
331
331 // psuedo random number generator
332
333
334
335
336
337
338
339
339 // psuedo random number generator
340
341
342
343
344
345
346
347
347 // psuedo random number generator
348
349
350
351
352
353
354
355
355 // psuedo random number generator
356
357
358
359
360
361
362
363
363 // psuedo random number generator
364
365
366
367
368
369
370
371
371 // psuedo random number generator
372
373
374
375
376
377
378
379
379 // psuedo random number generator
380
381
382
383
384
385
386
387
387 // psuedo random number generator
388
389
390
391
392
393
394
395
395 // psuedo random number generator
396
397
398
399
400
401
402
403
403 // psuedo random number generator
404
405
406
407
408
409
410
411
411 // psuedo random number generator
412
413
414
415
416
417
418
419
419 // psuedo random number generator
420
421
422
423
424
425
426
427
427 // psuedo random number generator
428
429
430
431
432
433
434
435
435 // psuedo random number generator
436
437
438
439
440
441
442
443
443 // psuedo random number generator
444
445
446
447
448
449
450
451
451 // psuedo random number generator
452
453
454
455
456
457
458
459
459 // psuedo random number generator
460
461
462
463
464
465
466
467
467 // psuedo random number generator
468
469
470
471
472
473
474
475
475 // psuedo random number generator
476
477
478
479
480
481
482
483
483 // psuedo random number generator
484
485
486
487
488
489
490
491
491 // psuedo random number generator
492
493
494
495
496
497
498
499
499 // psuedo random number generator
500
501
502
503
504
505
506
507
507 // psuedo random number generator
508
509
510
511
512
513
514
515
515 // psuedo random number generator
516
517
518
519
520
521
522
523
523 // psuedo random number generator
524
525
526
527
528
529
530
531
531 // psuedo random number generator
532
533
534
535
536
537
538
539
539 // psuedo random number generator
540
541
542
543
544
545
546
547
547 // psuedo random number generator
548
549
550
551
552
553
554
555
555 // psuedo random number generator
556
557
558
559
560
561
562
563
563 // psuedo random number generator
564
565
566
567
568
569
570
571
571 // psuedo random number generator
572
573
574
575
576
577
578
579
579 // psuedo random number generator
580
581
582
583
584
585
586
587
587 // psuedo random number generator
588
589
590
591
592
593
594
595
595 // psuedo random number generator
596
597
598
599
600
601
602
603
603 // psuedo random number generator
604
605
606
607
608
609
610
611
611 // psuedo random number generator
612
613
614
615
616
617
618
619
619 // psuedo random number generator
620
621
622
623
624
625
626
627
627 // psuedo random number generator
628
629
630
631
632
633
634
635
635 // psuedo random number generator
636
637
638
639
640
641
642
643
643 // psuedo random number generator
644
645
646
647
648
649
650
651
651 // psuedo random number generator
652
653
654
655
656
657
658
659
659 // psuedo random number generator
660
661
662
663
664
665
666
667
667 // psuedo random number generator
668
669
670
671
672
673
674
675
675 // psuedo random number generator
676
677
678
679
680
681
682
683
683 // psuedo random number generator
684
685
686
687
688
689
690
691
691 // psuedo random number generator
692
693
694
695
696
697
698
699
699 // psuedo random number generator
700
701
702
703
704
705
706
707
707 // psuedo random number generator
708
709
710
711
712
713
714
715
715 // psuedo random number generator
716
717
718
719
720
721
722
723
723 // psuedo random number generator
724
725
726
727
728
729
730
731
731 // psuedo random number generator
732
733
734
735
736
737
738
739
739 // psuedo random number generator
740
741
742
743
744
745
746
747
747 // psuedo random number generator
748
749
750
751
752
753
754
755
755 // psuedo random number generator
756
757
758
759
760
761
762
763
763 // psuedo random number generator
764
765
766
767
768
769
770
771
771 // psuedo random number generator
772
773
774
775
776
777
778
779
779 // psuedo random number generator
780
781
782
783
784
785
786
787
787 // psuedo random number generator
788
789
790
791
792
793
794
795
795 // psuedo random number generator
796
797
798
799
800
801
802
803
803 // psuedo random number generator
804
805
806
807
808
809
810
811
811 // psuedo random number generator
812
813
814
815
816
817
818
819
819 // psuedo random number generator
820
821
822
823
824
825
826
827
827 // psuedo random number generator
828
829
830
831
832
833
834
835
835 // psuedo random number generator
836
837
838
839
840
841
842
843
843 // psuedo random number generator
844
845
846
847
848
849
850
851
851 // psuedo random number generator
852
853
854
855
856
857
858
859
859 // psuedo random number generator
860
861
862
863
864
865
866
867
867 // psuedo random number generator
868
869
870
871
872
873
874
875
875 // psuedo random number generator
876
877
878
879
880
881
882
883
883 // psuedo random number generator
884
885
886
887
888
889
890
891
891 // psuedo random number generator
892
893
894
895
896
897
898
899
899 // psuedo random number generator
900
901
902
903
904
905
906
907
907 // psuedo random number generator
908
909
910
911
912
913
914
915
915 // psuedo random number generator
916
917
918
919
920
921
922
923
923 // psuedo random number generator
924
925
926
927
928
929
930
931
931 // psuedo random number generator
932
933
934
935
936
937
938
939
939 // psuedo random number generator
940
941
942
943
944
945
946
947
947 // psuedo random number generator
948
949
950
951
952
953
954
955
955 // psuedo random number generator
956
957
958
959
960
961
962
963
963 // psuedo random number generator
964
965
966
967
968
969
970
971
971 // psuedo random number generator
972
973
974
975
976
977
978
979
979 // psuedo random number generator
980
981
982
983
984
985
986
987
987 // psuedo random number generator
988
989
990
991
992
993
994
995
995 // psuedo random number generator
996
997
998
999
1000
1001
1002
1002 // psuedo random number generator
1003
1004
1005
1006
1007
1008
1009
1009 // psuedo random number generator
1010
1011
1012
1013
1014
1015
1016
1017
1017 // psuedo random number generator
1018
1019
1020
1021
1022
1023
1024
1025
1025 // psuedo random number generator
1026
1027
1028
1029
1030
1031
1032
1033
1033 // psuedo random number generator
1034
1035
1036
1037
1038
1039
1040
1041
1041 // psuedo random number generator
1042
1043
1044
1045
1046
1047
1048
1049
1049 // psuedo random number generator
1050
1051
1052
1053
1054
1055
1056
1057
1057 // psuedo random number generator
1058
1059
1060
1061
1062
1063
1064
1065
1065 // psuedo random number generator
1066
1067
1068
1069
1070
1071
1072
1073
1073 // psuedo random number generator
1074
1075
1076
1077
1078
1079
1080
1081
1081 // psuedo random number generator
1082
1083
1084
1085
1086
1087
1088
1089
1089 // psuedo random number generator
1090
1091
1092
1093
1094
1095
1096
1097
1097 // psuedo random number generator
1098
1099
1099 // psuedo random number generator
1100
1101
1102
1103
1104
1105
1106
1107
1107 // psuedo random number generator
1108
1109
1110
1111
1112
1113
1114
1115
1115 // psuedo random number generator
1116
1117
1118
1119
1120
1121
1122
1123
1123 // psuedo random number generator
1124
1125
1126
1127
1128
1129
1130
1131
1131 // psuedo random number generator
1132
1133
1134
1135
1136
1137
1138
1139
1139 // psuedo random number generator
1140
1141
1142
1143
1144
1145
1146
1147
1147 // psuedo random number generator
1148
1149
1150
1151
1152
1153
1154
1155
1155 // psuedo random number generator
1156
1157
1158
1159
1160
1161
1162
1163
1163 // psuedo random number generator
1164
1165
1166
1167
1168
1169
1170
1171
1171 // psuedo random number generator
1172
1173
1174
1175
1176
1177
1178
1179
1179 // psuedo random number generator
1180
1181
1182
1183
1184
1185
1186
1187
1187 // psuedo random number generator
1188
1189
1190
1191
1192
1193
1194
1195
1195 // psuedo random number generator
1196
1197
1198
1199
1200
1201
1202
1203
1203 // psuedo random number generator
1204
1205
1206
1207
1208
1209
1210
1211
1211 // psuedo random number generator
1212
1213
1214
1215
1216
1217
1218
1219
1219 // psuedo random number generator
1220
1221
1222
1223
1224
1225
1226
1227
1227 // psuedo random number generator
1228
1229
1230
1231
1232
1233
1234
1235
1235 // psuedo random number generator
1236
1237
1238
1239
1240
1241
1242
1243
1243 // psuedo random number generator
1244
1245
1246
1247
1248
1249
1250
1251
1251 // psuedo random number generator
1252
1253
1254
1255
1256
1257
1258
1259
1259 // psuedo random number generator
1260
1261
1262
1263
1264
1265
1266
1267
1267 // psuedo random number generator
1268
1269
1270
1271
1272
1273
1274
1275
1275 // psuedo random number generator
1276
1277
1278
1279
1280
1281
1282
1283
1283 // psuedo random number generator
1284
1285
1286
1287
1288
1289
1290
1291
1291 // psuedo random number generator
1292
1293
1294
1295
1296
1297
1298
1299
1299 // psuedo random number generator
1300
1301
1302
1303
1304
1305
1306
1307
1307 // psuedo random number generator
1308
1309
1310
1311
1312
1313
1314
1315
1315 // psuedo random number generator
1316
1317
1318
1319
1320
1321
1322
1323
1323 // psuedo random number generator
1324
1325
1326
1327
1328
1329
1330
1331
1331 // psuedo random number generator
1332
1333
1334
1335
1336
1337
1338
1339
1339 // psuedo random number generator
1340
1341
1342
1343
1344
1345
1346
1347
1347 // psuedo random number generator
1348
1349
1350
1351
1352
1353
1354
1355
1355 // psuedo random number generator
1356
1357
1358
1359
1360
1361
1362
1363
1363 // psuedo random number generator
1364
1365
1366
1367
1368
1369
1370
1371
1371 // psuedo random number generator
1372
1373
1374
1375
1376
1377
1378
1379
1379 // psuedo random number generator
1380
1381
1382
1383
1384
1385
1386
1387
1387 // psuedo random number generator
1388
1389
1390
1391
1392
1393
1394
1395
1395 // psuedo random number generator
1396
1397
1398
1399
1400
1401
1402
1403
1403 // psuedo random number generator
1404
1405
1406
1407
1408
1409
1410
1411
1411 // psuedo random number generator
1412
1413
1414
1415
1416
1417
1418
1419
1419 // psuedo random number generator
1420
1421
1422
1423
1424
1425
1426
1427
1427 // psuedo random number generator
1428
1429
1430
1431
1432
1433
1434
1435
1435 // psuedo random number generator
1436
1437
1438
1439
1440
1441
1442
1443
1443 // psuedo random number generator
1444
1445
1446
1447
1448
1449
1450
1451
1451 // psuedo random number generator
1452
1453
1454
1455
1456
1457
1458
1459
1459 // psuedo random number generator
1460
1461
1462
1463
1464
1465
1466
1467
1467 // psuedo random number generator
1468
1469
1470
1471
1472
1473
1474
1475
1475 // psuedo random number generator
1476
1477
1478
1479
1480
1481
1482
1483
1483 // psuedo random number generator
1484
1485
1486
1487
1488
1489
1490
1491
1491 // psuedo random number generator
1492
1493
1494
1495
1496
1497
1498
1499
1499 // psuedo random number generator
1500
1501
1502
1503
1504
1505
1506
1507
1507 // psuedo random number generator
1508
1509
1510
1511
1512
1513
1514
1515
1515 // psuedo random number generator
1516
1517
1518
1519
1520
1521
1522
1523
1523 // psuedo random number generator
1524
1525
1526
1527
1528
1529
1530
1531
1531 // psuedo random number generator
1532
1533
1534
1535
1536
1537
1538
1539
1539 // psuedo random number generator
1540
1541
1542
1543
1544
1545
1546
1547
1547 // psuedo random number generator
1548
1549
1550
1551
1552
1553
1554
1555
1555 // psuedo random number generator
1556
1557
1558
1559
1560
1561
1562
1563
1563 // psuedo random number generator
1564
1565
1566
1567
1568
1569
1570
1571
1571 // psuedo random number generator
1572
1573
1574
1575
1576
1577
1578
1579
1579 // psuedo random number generator
1580
1581
1582
1583
1584
1585
1586
1587
1587 // psuedo random number generator
1588
1589
1590
1591
1592
1593
1594
1595
1595 // psuedo random number generator
1596
1597
1598
1599
1600
1601
1602
1603
1603 // psuedo random number generator
1604
1605
1606
1607
1608
1609
1610
1611
1611 // psuedo random number generator
1612
1613
1614
1615
1616
1617
1618
1619
1619 // psuedo random number generator
1620
1621
1622
1623
1624
1625
1626
1627
1627 // psuedo random number generator
1628
1629
1630
1631
1632
1633
1634
1635
1635 // psuedo random number generator
1636
1637
1638
1639
1640
1641
1642
1643
1643 // psuedo random number generator
1644
1645
1646
1647
1648
1649
1650
1651
1651 // psuedo random number generator
1652
1653
1654
1655
1656
1657
1658
1659
1659 // psuedo random number generator
1660
1661
1662
1663
1664
1665
1666
1667
1667 // psuedo random number generator
1668
1669
1670
1671
1672
1673
1674
1675
1675 // psuedo random number generator
1676
1677
1678
1679
1680
1681
1682
1683
1683 // psuedo random number generator
1684
1685
1686
1687
1688
1689
1690
1691
1691 // psuedo random number generator
1692
1693
1694
1695
1696
1697
1698
1699
1699 // psuedo random number generator
1700
1701
1702
1703
1704
1705
1706
1707
1707 // psuedo random number generator
1708
1709
1710
1711
1712
1713
1714
1715
1715 // psuedo random number generator
1716
1717
1
```

Ubuntu 2004 Activities SmartGit

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

kernel/proc.c (Index)

```
1 // Allocate a trapframe page.
2 if(p->trapframe = (struct trapframe *)kalloc()) == 0{
3     freeproc(p);
4     release(p->lock);
5     return 8;
6 }
7
8 // An empty user page table.
9 p->pagetable = proc_pagedtable(p);
10 if(p->pagetable == 0){
11     freeproc(p);
12     release(p->lock);
13     return 8;
14 }
15
16 // Set up new context to start executing at forkret,
17 // which returns to user space.
18 if(proc_newcontext(p, 0, p->context) < 0)
19     p->context.ra = (uint32)forkret;
20 p->context.sp = p->kstack + PGSIZE;
21
22 return p;
23 }
24
25 // Free a proc structure and the data hanging from it,
26 // including user pages.
27 // p->lock must be held.
28 static void
29 freeproc(struct proc *p)
30 {
31     if(p->trapframe)
32         kfree((void*)p->trapframe);
33     p->trapframe = 0;
34     if(p->pagetable)
35         proc_Freepagetable(p->pagetable, p->sz);
36     p->pagetable = 0;
37     p->pid = 0;
38     p->parent = 0;
39     p->name[0] = 0;
40     p->killed = 0;
41 }
```

[kernel/proc.c] - File Compare

kernel/proc.c (Working Tree)

```
141     release(p->lock);
142     return 0;
143 }
144
145 // An empty user page table.
146 p->pagetable = proc_pagedtable(p);
147 if(p->pagetable == 0){
148     freeproc(p);
149     release(p->lock);
150     return 0;
151 }
152
153 // Set up new context to start executing at forkret,
154 // which returns to user space.
155 memset(p->context, 0, sizeof(p->context));
156 p->context.ra = (uint32)forkret;
157 p->context.sp = p->kstack + PGSIZE;
158
159 // Initialize the default value of ticks, tickets, strides and pass
160 p->ticks = 0;
161 p->tickets = 0;
162 p->strides = 1;
163 p->pass = p->strides;
164
165
166 return p;
167 }
168
169 // Free a proc structure and the data hanging from it,
170 // including user pages.
171 // p->lock must be held.
172 static void
173 freeproc(struct proc *p)
174 {
175     if(p->trapframe)
176         kfree((void*)p->trapframe);
177     p->trapframe = 0;
178     if(p->pagetable)
179         proc_Freepagetable(p->pagetable, p->sz);
180     p->pagetable = 0;
181     p->pid = 0;
182     p->parent = 0;
183     p->name[0] = 0;
184 }
```

Show Applications default value of ticks, tickets, strides and pass

Ready

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

149:1 43-1 ENIG 11:56 PM

The screenshot shows a file comparison between two versions of the Linux kernel source code: `kernel/proc.c (Index)` and `kernel/proc.c (Working Tree)`. The Working Tree version contains several additional features:

- Support for the Lottery Scheduler, indicated by the `#IF defined(LOTTERY)` macro.
- A new scheduler type, `SCHED_TYPE_STRIDE`, which is used in place of the primary address of the structure passed to system call `copyout`.
- Implementation of the Stride Scheduler, which is selected via the `SCHED_TYPE_STRIDE` macro.
- Implementation of the RR Scheduler, which is selected via the `SCHED_TYPE_RR` macro.
- Support for lottery scheduling in `copyout` operations, indicated by the `#IF defined(LOTTERY)` macro.

The code also includes comments explaining the changes made in the Working Tree version, such as "print stats as mentioned in lab document" and "Function to update the structure passed with received values".

File Edit View Go To Window

Save Reload | Prev.Change Next.Change | Take Left Take Right

kernel/proc.c (Index)

```
718     return proc_ctr;
719 }
720 else if (param == 1)
721 {
722     return sys_calls_count;
723 }
724 else if (param == 2)
725 {
726     int count = get_free_memory();
727     return count;
728 }
729 else
730 {
731     return -1;
732 }
733 }
```

// function to update the structure passed with required values
734 void putprocinfo(struct proc *parent, struct wait_lock *to_be_held
735 {
736 if (parent->sys_call_count <= 0) return;
737 if (parent->page_usage <= 0) return;
738 if (parent->page_size <= 0) return;
739 if (parent->processes_size <= 0) return;
740 if (parent->ticks <= 0) return;
741 if (parent->pid <= 0) return;
742 if (parent->name[0] == '\0') return;
743 int get_sys_procinfo(uint64 addrs)
744 {
745 struct proc *p = myproc();
746 struct pinfo pinfo;
747 acquire(&wait_lock);
748 pinfo.pid = p->parent->pid;
749 release(&wait_lock);
750
751 pinfo.syscall_count = p->sys_call_count;
752 pinfo.page_usage = ((p->sz) / 4096) + (((p->sz) % 4096) != 0);
753
754 // copy the data from temp struct to the memory address of struct passed to system call
755 if (copyout(p->pageable, addrs, (char *) &pinfo, sizeof(pinfo)) < 0)
756 return 1;
757 return 0;
758 }
759 }

#if defined(LOTTERY)

Ready

[kernel/proc.c] - File Compare

kernel/proc.c (Working Tree)

```
873
874 #if defined(LOTTERY)
875     printf("\n .. Lottery Scheduler .. \n\n");
876 #elif defined(STRIDE)
877     printf("\n .. Stride Scheduler .. \n\n");
878 #else
879     printf("\n .. RR Scheduler .. \n\n");
880 #endif
881
882 struct proc *p;
883
884 for (p = proc; p < &proc[NPROC]; p++)
885 {
886     if (p->state != UNUSED)
887     {
888         acquire(&p->lock);
889         #if defined(STRIDE)
890             printf("%d(%s): tickets: %d, ticks: %d\n", p->pid, p->name, p->tickets, 0, p->ticks);
891         #else
892             printf("%d(%s): tickets: %d, ticks: %d\n", p->pid, p->name, p->tickets, p->ticks);
893         #endif
894         seminit(&p->sem);
895         release(&p->lock);
896     }
897 }
898
899 // always return 0 as per lab
900 return 0;
901 }
902
903 // assign tickets to process
904 int sched_tickets(int tickets)
905 {
906     struct proc *p = myproc();
907     p->tickets = tickets;
908     p->ticks = 0;
909     p->strides = LARGE_CONSTANT_K / tickets;
910
911     // calculate the value of pass
912     p->pass = p->strides;
913
914     // always return 0 as per lab
915     return 0;
916 }
```

468:1 +3~1

```
File Edit View Go To Window
Save Reload Prev.Change Next.Change Take Left Take Right
kernel/proc.c (Index)
411
412 // Per-CPU process scheduler.
413 // When CPU calls scheduler() after setting itself up,
414 // Scheduler never returns. It loops, doing:
415 // - choose a process to run,
416 // - switch to start running that process,
417 // - switch back to the scheduler.
418 // via switch back to the scheduler.
419 void
420 scheduler(void)
421 {
422     struct proc *p;
423     struct cpu *c = mycpu();
424
425     c->proc = 0;
426     for(;;) {
427         // Avoid deadlock by ensuring that devices can interrupt.
428         spin_on();
429
430         for(p = proc; p < aproc[NPROC]; p++) {
431             if(p->lock) {
432                 // Switch to chosen process. It is the process's job
433                 // to release its lock and then reacquire it
434                 // before jumping back to us.
435                 p->state = RUNNING;
436                 c->proc = p;
437                 switch_to(&c->context, &p->context);
438
439                 // Process is done running for now,
440                 // It should have changed its p->state before coming back.
441                 c->proc = 0;
442             }
443             releases(&p->lock);
444         }
445     }
446 }
447 // Switch to scheduler. Must hold only p->lock
448 // and have changed proc->state. Saves and restores
449 // intena because intena is a property of this
450 // kernel thread, not this CPU. It should
451 // be proc->intena and proc->off... but that would
452
453 #if defined(LOTTERY)
454
455 kernel/proc.c (Working Tree)
456
457 ...
458 c->proc = 0;
459 for(;;)
460 {
461     // Avoid deadlock by ensuring that devices can interrupt.
462     //int_on();
463
464     #if defined(LOTTERY)
465     // printf("Lottery Scheduler working\n");
466     for (p = proc; p < aproc[NPROC]; p++)
467     {
468         acquire(&p->lock);
469         int tickets_sum = 0;
470         int total_tickets = 0;
471         if (p->state != RUNNABLE)
472         {
473             releases(&p->lock);
474             continue;
475         }
476         struct proc *pp;
477         for (pp = proc; pp < aproc[NPROC]; pp++)
478         {
479             if (pp->state == RUNNABLE)
480             {
481                 total_tickets += pp->tickets;
482             }
483         }
484
485         int win = rand() % (total_tickets + 1);
486
487         if ((tickets_sum + p->tickets) < win)
488         {
489             tickets_sum += p->tickets;
490             releases(&p->lock);
491             continue;
492         }
493
494         if (p->state == RUNNABLE)
495         {
496             // Switch to chosen process.. It is the process's job
497             // to release its lock and then reacquire it
498             // before jumping back to us.
499             p->state = RUNNING;
500             c->proc = p;
501         }
502     }
503
504     #endif
505
506     if (c->proc)
507     {
508         // Switch to chosen process.. It is the process's job
509         // to release its lock and then reacquire it
510         // before jumping back to us.
511         p->state = RUNNING;
512         c->proc = 0;
513     }
514 }
```

Ubuntu 2004 - VMware Workstation

File Edit View VM Tabs Help | ||| □ □ □ □ □ □ □ □

Ubuntu 2004 Activities SmartGit Feb 21 23:47 • [kernel/proc.c] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

kernel/proc.c (Index)

```

446 // - switch to start running that process.
447 // - eventually that process transfers control
448 // via switch back to the scheduler.
449 void
450 scheduler(void)
451 {
452     struct proc *p;
453     struct cpu *c = mycpu();
454     c->proc = 0;
455     for(;;)
456     {
457         // Avoid deadlock by ensuring that devices can interrupt.
458         int_on();
459
460         for(p = proc; p < Aproc[NPROC]; p++)
461         {
462             acquire(&p->lock);
463             if(p->state == RUNNABLE)
464             {
465                 // Switch to chosen process. It is the process's job
466                 // to release its lock and then reacquire it
467                 // before jumping back to us.
468                 p->state = RUNNING;
469                 c->proc = p;
470                 switch(&c->context, &p->context);
471
472                 // Process is done running for now.
473                 // It should have changed its p->state before coming back.
474                 c->proc = 0;
475             }
476             release(&p->lock);
477         }
478
479         // Switch to scheduler. Must hold only p->lock
480         // and have changed proc->state. Saves and restores
481         // Intena because Intena is a property of this
482         // kernel thread, not this CPU. It should
483         // be proc->intena and proc->off, but that would
484         // break in the few places where a lock is held but
485         // there's no process.
486     void
487     sched(void)
488     {
489         int_intena;
490         struct proc *p = myproc();
491
492         if(holding(&p->lock))
493             panic("sched: p->lock");
494
495 #if defined(LOTTERY)
496
497 Ready
498
499 To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

kernel/proc.c (Working Tree)

```

504
505         }
506
507         if (p->state == RUNNABLE)
508         {
509             // Switch to chosen process. It is the process's job
510             // to release its lock and then reacquire it
511             // before jumping back to us.
512             p->state = RUNNING;
513             c->proc = p;
514             switch(&c->context, &p->context);
515             p->ticks++; // Lab
516
517             // Process ls running for now.
518             // It should have changed its p->state before coming back.
519             c->proc = 0;
520         }
521         release(&p->lock);
522
523 #endif
524 // printf("Scheduler working\n");
525 int min_pass = LARGE_CONSTANT_K;
526 int max_pass = D / 2;
527 // find the min_p
528 for (p = proc; p < Aproc[NPROC]; p++)
529 {
530     acquire(&p->lock);
531     if (p->state == RUNNABLE)
532     {
533         if (p->pass < min_pass)
534         {
535             min_pass = p->pass;
536             min_p = p;
537         }
538     }
539     release(&p->lock);
540 }
541 if(min_p)
542 {
543     p = min_p;
544     acquire(&p->lock);
545     p->pass += p->ticks;
546     if (p->state == RUNNABLE)
547         // Switch to chosen process. It is the process's job
548         // to release its lock and then reacquire it
549         // before jumping back to us.
550         p->state = RUNNING;
551         c->proc = p;
552         switch(&c->context, &p->context);
553         p->ticks++; // Lab
554
555         // Process is done running for now.
556         // It should have changed its p->state before coming back.
557         c->proc = 0;
558     }
559     release(&p->lock);
560 }
561 #endif
562 // printf("Default RR Scheduler working\n");
563 // nothing changes here
564 for (p = proc; p < Aproc[NPROC]; p++)
565 {
566     acquire(&p->lock);
567     if (p->state == RUNNABLE)
568     {
569         // Switch to chosen process. It is the process's job
570         // to release its lock and then reacquire it
571         // before jumping back to us.
572         p->state = RUNNING;
573         c->proc = p;
574         switch(&c->context, &p->context);
575         p->ticks++; // Lab
576
577         // Process is done running for now.
578         // It should have changed its p->state before coming back.
579         c->proc = 0;
580     }
581 }

```

468:1 +3-1

Ubuntu 2004 - VMware Workstation

File Edit View VM Tabs Help | ||| □ □ □ □ □ □ □ □

Ubuntu 2004 Activities SmartGit Feb 21 23:47 • [kernel/proc.c] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

kernel/proc.c (Index)

```

451
452     struct proc *p;
453     struct cpu *c = mycpu();
454     ...
455     c->proc = 0;
456     for(;;)
457     {
458         // Avoid deadlock by ensuring that devices can interrupt.
459         int_on();
460
461         for(p = proc; p < Aproc[NPROC]; p++)
462         {
463             acquire(&p->lock);
464             if(p->state == RUNNABLE)
465             {
466                 // Switch to chosen process. It is the process's job
467                 // to release its lock and then reacquire it
468                 // before jumping back to us.
469                 p->state = RUNNING;
470                 c->proc = p;
471                 switch(&c->context, &p->context);
472
473                 // Process is done running for now.
474                 // It should have changed its p->state before coming back.
475                 c->proc = 0;
476             }
477             release(&p->lock);
478         }
479
480         // Switch to scheduler. Must hold only p->lock
481         // and have changed proc->state. Saves and restores
482         // Intena because Intena is a property of this
483         // kernel thread, not this CPU. It should
484         // be proc->intena and proc->off, but that would
485         // break in the few places where a lock is held but
486         // there's no process.
487     void
488     sched(void)
489     {
490         int_intena;
491         struct proc *p = myproc();
492
493         if(holding(&p->lock))
494             panic("sched: p->lock");
495
496 #if defined(LOTTERY)
497
498 Ready
499
500 To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

kernel/proc.c (Working Tree)

```

538         }
539         release(&p->lock);
540     }
541     if(min_p)
542     {
543         p = min_p;
544         acquire(&p->lock);
545         p->pass += p->ticks;
546         if (p->state == RUNNABLE)
547         {
548             // Switch to chosen process. It is the process's job
549             // to release its lock and then reacquire it
550             // before jumping back to us.
551             p->state = RUNNING;
552             c->proc = p;
553             switch(&c->context, &p->context);
554             p->ticks++; // Lab
555
556             // Process is done running for now.
557             // It should have changed its p->state before coming back.
558             c->proc = 0;
559         }
560         release(&p->lock);
561     }
562 #endif
563 // printf("Default RR Scheduler working\n");
564 // nothing changes here
565 for (p = proc; p < Aproc[NPROC]; p++)
566 {
567     acquire(&p->lock);
568     if (p->state == RUNNABLE)
569     {
570         // Switch to chosen process. It is the process's job
571         // to release its lock and then reacquire it
572         // before jumping back to us.
573         p->state = RUNNING;
574         c->proc = p;
575         switch(&c->context, &p->context);
576         p->ticks++; // Lab
577
578         // Process is done running for now.
579         // It should have changed its p->state before coming back.
580         c->proc = 0;
581     }
582 }

```

468:1 +3-1

Ubuntu 2004

Activities SmartGit

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

Makefile (index)

```

116 # $Revision: 1.1522 $
117 # http://www.gnu.org/software/make/manual/html_node/chained-Rules.html
118 .PRECIOUS: %.o
119 %.o:
120   SU/_cat
121   SU/_echo
122   SU/_forktest
123   SU/_grep
124   SU/_int
125   SU/_kill
126   SU/_l11
127   SU/_ls
128   SU/_mkdir
129   SU/_rm
130   SU/_sh
131   SU/_stressfs
132   SU/_usertests
133   SU/_vcd
134   SU/_wc
135   SU/_zombie
136   SU/_lab1
137   SU/_lab2
138   SU/_lab3
139   SU/_lab4
140
141   mks/mkfs READEME $(UPROGS)
142   mks/mkfs fs.lng READEME $(UPROGS)
143
144   -include kernel/*.d user/*.d
145
146 clean:
147   rm -f *.tex *.dvi *.dvi *.aux *.log *.ind *.ilg \
148   */*.d */*.aux */*.syn
149   SU/unicode SU/unicode.out SK/kernel fs.lng \
150   mks/mkfs .gdbinit \
151   SU/sys.S \
152   $(UPROGS)
153
154 # try to generate a unique GDB port
155 GDBPORT = $(shell expr `id -u` % 5980 + 25980)
156 # QEMU's gdb stub command line changed in 0.11
157 QEMUQOB = $(shell if $(QEMU) .help | grep -q '^gdb'; \
158   then echo " -gdb tcp:$GDBPORT"; \
159   else echo " -s -p $(GDBPORT)"; fi)
160
161 ifndef CPU
162   define CPU
163     $(GDB) -x $(GDBPORT)
164   endef
165
166   SU/_lab2

```

Ready

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Feb 21 23:48 •

136:1 +2

ENG 11:48 PM

Ubuntu 2004

Activities SmartGit

File Edit View VM Tabs Help

Save Reload Prev Change Next Change Take Left Take Right

kernel/defs.h - File Compare

kernel/defs.h (index)

```

102 int      killed(struct proc* );
103 void    mycpu(void);
104 struct cpu* getmycpu(void);
105 struct proc* myproc();
106 void    scheduler(void) __attribute__((noreturn));
107 void    sched(void);
108 void    sleep(void*, struct spinlock* );
109 void    wake(void*, struct spinlock* );
110 void    wait(uint64);
111 void    wakeup(void* );
112 void    yield(void* );
113 void    precdump(void);
114 void    either_copyout(int user_dst, uint64 dst, void *src, uint64 len);
115 void    either_copyin(void *dst, int user_src, uint64 src, uint64 len);
116 void    get_sys_sysinfo(uint64, uint64);
117 int     get_sys_proctitle(uint64, uint64);
118 void    get_sys_procinfo(uint64 addr);
119
120 // switch.S
121 void    swtch(struct context*, struct context* );
122
123 // spinlock.c
124 void    acquire(struct spinlock* );
125 void    holding(struct spinlock* );
126 int     initlock(struct spinlock*, char* );
127 void    release(struct spinlock* );
128 void    push_off(void* );
129 void    pop_off(void* );
130
131 // sleeplock.c
132 void    acquire_sleeplock(struct sleeplock* );
133 void    release_sleeplock(struct sleeplock* );
134 int     holding_sleeplock(struct sleeplock* );
135 void    inittsleeplock(struct sleeplock*, char* );
136 void    release_tsleeplock(struct sleeplock*, char* );
137 void    push_tsleeplock(void* );
138 void    pop_tsleeplock(void* );
139
140 // string.c
141 void    nmemcmp(const void*, const void*, uint);
142 void    nremove(void*, const void*, uint);
143
144 // system call to print statistics of the process
145 void    sched_statistics(void);

```

Ready

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Feb 21 23:48 •

124:1 +1-1

ENG 11:48 PM

The screenshot shows a desktop environment with a terminal window and a code editor.

Terminal Window:

- Title: Ubuntu 2004
- Date/Time: Feb 21 23:49
- Path: ~./Downloads/ming_github/user
- Content:

```
lab2c
```

Code Editor:

- Title: Text Editor
- File: lab2c.c
- Content:

```
1 #include <kernel/types.h>
2 #include <kernel/synch.h>
3 #include <user/user.h>
4 #define MAX_PROC 10
5
6 int main(int argc, char *argv[])
7 {
8     int sleep_ticks, n_proc, ret, proc_pid[MAX_PROC];
9     if (argc < 4)
10    {
11        printf("Usage: %s [SLEEP] [N_PROC] [TICKET1] [TICKET2]...\n", argv[0]);
12        exit(-1);
13    }
14    sleep_ticks = atoi(argv[1]);
15    n_proc = atoi(argv[2]);
16    if (n_proc > MAX_PROC)
17    {
18        printf("Cannot test with more than %d processes\n", MAX_PROC);
19        exit(-1);
20    }
21    for (int i = 0; i < n_proc; i++)
22    {
23        int n_tickets = atoi(argv[3 + i]);
24
25        #if defined(STRIKE)
26        n_tickets *= 10;
27        #endif
28        n_tickets = n_tickets;
29        #endif
30
31        ret = fork();
32        if (ret == 0)
33        { // child process
34            sched_tickets(n_tickets);
35            while(1)
36            ;
37        }
38        else
39        { // parent
40            proc_pid[i] = ret;
41            continue;
42        }
43    }
44    sleep(sleep_ticks);
45    sched_statistics();
46 }
```
- Bottom status bar: Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

The screenshot shows a Linux desktop environment with a terminal window and a file browser window.

Terminal Window:

- Title: Ubuntu 2004
- Text Editor: labz.c
- Content:

```
4 #include <sys/types.h>
5 #include <sys/stat.h>
6 #include <sys/conf.h>
7 #include <sys/systm.h>
8 #include <sys/kernel.h>
9 #include <sys/malloc.h>
10 #include <sys/thread.h>
11 #include <sys/threadlist.h>
12 #include <sys/threadlist.h>
13 #include <sys/threadlist.h>
14 #include <sys/threadlist.h>
15 #include <sys/threadlist.h>
16 #include <sys/threadlist.h>
17 #include <sys/threadlist.h>
18 #include <sys/threadlist.h>
19 #include <sys/threadlist.h>
20 #include <sys/threadlist.h>
21 #include <sys/threadlist.h>
22 #include <sys/threadlist.h>
23 #include <sys/threadlist.h>
24 #include <sys/threadlist.h>
25 #include <sys/threadlist.h>
26 #include <sys/threadlist.h>
27 #include <sys/threadlist.h>
28 #include <sys/threadlist.h>
29 #include <sys/threadlist.h>
30 #include <sys/threadlist.h>
31 #include <sys/threadlist.h>
32 #include <sys/threadlist.h>
33 #include <sys/threadlist.h>
34 #include <sys/threadlist.h>
35 #include <sys/threadlist.h>
36 #include <sys/threadlist.h>
37 #include <sys/threadlist.h>
38 #include <sys/threadlist.h>
39 #include <sys/threadlist.h>
40 #include <sys/threadlist.h>
41 #include <sys/threadlist.h>
42 #include <sys/threadlist.h>
43 #include <sys/threadlist.h>
44 #include <sys/threadlist.h>
45 #include <sys/threadlist.h>
46 #include <sys/threadlist.h>
47 #include <sys/threadlist.h>
48 #include <sys/threadlist.h>
49 }
```
- Status bar: Feb 21 23:49 • /Downloads/my_github/user

File Browser Window:

- Title: Text Editor
- Icon: A yellow flame-like icon.
- Content:
 - Open
 - Save
 - Activities
 - File
 - Edit
 - View
 - Help

Bottom Bar:

- Ubuntu 20.04 logo
- Q Search
- Windows Start button
- Google Chrome icon
- File Explorer icon
- Terminal icon
- System tray icons: ENG IN, 11:48 PM, 2/21/2023