


```

76 /* Pintos main program. */
77 int
78 main (void)
79 {
80     char **argv;
81
82     /* Clear BSS. */
83     bss_init ();
84
85     /* Break command line into arguments and parse options. */
86     argv = read_command_line ();
87     argv = parse_options (argv);
88
89     /* Initialize ourselves as a thread so we can use locks,
90      | then enable console locking. */
91     thread_init ();
92     console_init ();
93
94     /* Greet user. */
95     printf ("Pintos booting with %"PRIu32" kB RAM...\n",
96            init_ram_pages * PGSIZE / 1024);
97
98     /* Initialize memory system. */
99     paloc_init (user_page_limit);
100    malloc_init ();
101    paging_init ();
102
103    /* Segmentation. */
104 #ifdef USERPROG
105    tss_init ();
106    gdt_init ();
107#endif
108
109    /* Initialize interrupt handlers. */
110    timer_init ();
111    kbd_init ();
112    input_init ();
113 #ifdef USERPROG
114    exception_init ();
115    syscall_init ();
116#endif
117
118    /* Start thread scheduler and enable interrupts. */
119    thread_start ();
120    serial_init_queue ();
121    timer_calibrate ();
122
123 #ifdef FILESYS
124    initialize_file_system ();
125    ide_init ();
126    locate_block_devices ();
127    fileys_init (format_filesys);
128#endif
129
130    printf ("Boot complete.\n");
131
132    /* Run actions specified on kernel command line. */
133    run_actions (argv);
134
135    /* Finish up. */
136    shutdown ();
137    thread_exit ();
138 }

```

```

118 /* Process all of the actions indicated on @argp. */
119 void
120 run_actions (char **argp)
121 {
122     /* Create a stack for each action. */
123     struct action {
124         /* Action name. */
125         const char *name;
126         /* Function to invoke when action name is used. */
127         void (*function) (char **argp);
128     } actions[] = {
129         { "halt", halt },
130         { "dump", dump },
131         { "exit", exit },
132         { "idle", idle },
133         { "mem", mem },
134         { "memdump", memdump },
135         { "meminfo", meminfo },
136         { "memset", memset },
137         { "memzero", memzero },
138         { "memcopy", memcopy },
139         { "memmove", memmove },
140         { "memfill", memfill },
141         { "memrand", memrand },
142         { "memrandfill", memrandfill },
143         { "memrandzero", memrandzero },
144         { "memrandcopy", memrandcopy },
145         { "memrandmove", memrandmove },
146         { "memrandfillfill", memrandfillfill },
147         { "memrandfillzero", memrandfillzero },
148         { "memrandcopycopy", memrandcopycopy },
149         { "memrandmovemove", memrandmovemove },
150         { "memrandfillcopy", memrandfillcopy },
151         { "memrandcopyfill", memrandcopyfill },
152         { "memrandmovefill", memrandmovefill },
153         { "memrandfillmove", memrandfillmove },
154         { "memrandcopymove", memrandcopymove },
155         { "memrandmovecopy", memrandmovecopy },
156         { "memrandmovefillfill", memrandmovefillfill },
157         { "memrandfillmovefill", memrandfillmovefill },
158     };
159
160     /* Check for required arguments. */
161     if (argp[0] == NULL || argp[1] == NULL)
162         PAME ("Action %s requires %d argument(s), %s or -s %s\n",
163               argp[0], 2, argp[0], argp[1]);
164
165     /* Execute action. */
166     for (i = 0; i < sizeof (actions) / sizeof (actions[0]); i++)
167         if (strcmp (actions[i].name, argp[0]) == 0)
168             actions[i].function (argp + 1);
169
170     /* If no action was found, print usage information. */
171     if (argp[0] != NULL)
172         usage (argp[0]);
173
174     /* Free memory allocated for actions. */
175     free (actions);
176 }

```

```

118 /* Initializes the interrupt system. */
119 void
120 init_irq (void)
121 {
122     uint64_t idtr_operand;
123     int i;
124
125     /* Initialize interrupt controller. */
126     idtr_operand = 0;
127
128     /* Set IDTR register. */
129     See (IA32-TR-2, section 16.1, page 102-103) "Interrupt
130     Table IDTR Register"
131     idtr_operand |= Table_IDTR_and_operand (sizeof (idtr) - 3, idtr);
132     idtr_operand |= Table_IDTR_and_operand (sizeof (idtr) - 1, idtr);
133
134     /* Initialize interrupt names. */
135     for (i = 0; i < sizeof (intr_names) / sizeof (intr_names[0]); i++)
136         intr_names[i] = "Unknown";
137
138     /* Initialize interrupt exceptions. */
139     for (i = 0; i < sizeof (intr_name2) / sizeof (intr_name2[0]); i++)
140         intr_name2[i] = "Unknown";
141
142     /* Initialize interrupt codes. */
143     for (i = 0; i < sizeof (intr_code) / sizeof (intr_code[0]); i++)
144         intr_code[i] = 0;
145
146     /* Initialize interrupt names. */
147     for (i = 0; i < sizeof (intr_names2) / sizeof (intr_names2[0]); i++)
148         intr_names2[i] = "Unknown";
149
150     /* Initialize interrupt codes. */
151     for (i = 0; i < sizeof (intr_code2) / sizeof (intr_code2[0]); i++)
152         intr_code2[i] = 0;
153
154     /* Initialize interrupt names. */
155     for (i = 0; i < sizeof (intr_names3) / sizeof (intr_names3[0]); i++)
156         intr_names3[i] = "Unknown";
157
158     /* Initialize interrupt codes. */
159     for (i = 0; i < sizeof (intr_code3) / sizeof (intr_code3[0]); i++)
160         intr_code3[i] = 0;
161
162     /* Initialize interrupt names. */
163     for (i = 0; i < sizeof (intr_names4) / sizeof (intr_names4[0]); i++)
164         intr_names4[i] = "Unknown";
165
166     /* Initialize interrupt codes. */
167     for (i = 0; i < sizeof (intr_code4) / sizeof (intr_code4[0]); i++)
168         intr_code4[i] = 0;
169
170     /* Initialize interrupt names. */
171     for (i = 0; i < sizeof (intr_names5) / sizeof (intr_names5[0]); i++)
172         intr_names5[i] = "Unknown";
173
174     /* Initialize interrupt codes. */
175     for (i = 0; i < sizeof (intr_code5) / sizeof (intr_code5[0]); i++)
176         intr_code5[i] = 0;
177
178     /* Initialize interrupt names. */
179     for (i = 0; i < sizeof (intr_names6) / sizeof (intr_names6[0]); i++)
180         intr_names6[i] = "Unknown";
181
182     /* Initialize interrupt codes. */
183     for (i = 0; i < sizeof (intr_code6) / sizeof (intr_code6[0]); i++)
184         intr_code6[i] = 0;
185
186     /* Initialize interrupt names. */
187     for (i = 0; i < sizeof (intr_names7) / sizeof (intr_names7[0]); i++)
188         intr_names7[i] = "Unknown";
189
190     /* Initialize interrupt codes. */
191     for (i = 0; i < sizeof (intr_code7) / sizeof (intr_code7[0]); i++)
192         intr_code7[i] = 0;
193
194     /* Initialize interrupt names. */
195     for (i = 0; i < sizeof (intr_names8) / sizeof (intr_names8[0]); i++)
196         intr_names8[i] = "Unknown";
197
198     /* Initialize interrupt codes. */
199     for (i = 0; i < sizeof (intr_code8) / sizeof (intr_code8[0]); i++)
200         intr_code8[i] = 0;
201
202     /* Initialize interrupt names. */
203     for (i = 0; i < sizeof (intr_names9) / sizeof (intr_names9[0]); i++)
204         intr_names9[i] = "Unknown";
205
206     /* Initialize interrupt codes. */
207     for (i = 0; i < sizeof (intr_code9) / sizeof (intr_code9[0]); i++)
208         intr_code9[i] = 0;
209
210     /* Initialize interrupt names. */
211     for (i = 0; i < sizeof (intr_names10) / sizeof (intr_names10[0]); i++)
212         intr_names10[i] = "Unknown";
213
214     /* Initialize interrupt codes. */
215     for (i = 0; i < sizeof (intr_code10) / sizeof (intr_code10[0]); i++)
216         intr_code10[i] = 0;
217
218     /* Initialize interrupt names. */
219     for (i = 0; i < sizeof (intr_names11) / sizeof (intr_names11[0]); i++)
220         intr_names11[i] = "Unknown";
221
222     /* Initialize interrupt codes. */
223     for (i = 0; i < sizeof (intr_code11) / sizeof (intr_code11[0]); i++)
224         intr_code11[i] = 0;
225
226     /* Initialize interrupt names. */
227     for (i = 0; i < sizeof (intr_names12) / sizeof (intr_names12[0]); i++)
228         intr_names12[i] = "Unknown";
229
230     /* Initialize interrupt codes. */
231     for (i = 0; i < sizeof (intr_code12) / sizeof (intr_code12[0]); i++)
232         intr_code12[i] = 0;
233
234     /* Initialize interrupt names. */
235     for (i = 0; i < sizeof (intr_names13) / sizeof (intr_names13[0]); i++)
236         intr_names13[i] = "Unknown";
237
238     /* Initialize interrupt codes. */
239     for (i = 0; i < sizeof (intr_code13) / sizeof (intr_code13[0]); i++)
240         intr_code13[i] = 0;
241
242     /* Initialize interrupt names. */
243     for (i = 0; i < sizeof (intr_names14) / sizeof (intr_names14[0]); i++)
244         intr_names14[i] = "Unknown";
245
246     /* Initialize interrupt codes. */
247     for (i = 0; i < sizeof (intr_code14) / sizeof (intr_code14[0]); i++)
248         intr_code14[i] = 0;
249
250     /* Initialize interrupt names. */
251     for (i = 0; i < sizeof (intr_names15) / sizeof (intr_names15[0]); i++)
252         intr_names15[i] = "Unknown";
253
254     /* Initialize interrupt codes. */
255     for (i = 0; i < sizeof (intr_code15) / sizeof (intr_code15[0]); i++)
256         intr_code15[i] = 0;
257
258     /* Initialize interrupt names. */
259     for (i = 0; i < sizeof (intr_names16) / sizeof (intr_names16[0]); i++)
260         intr_names16[i] = "Unknown";
261
262     /* Initialize interrupt codes. */
263     for (i = 0; i < sizeof (intr_code16) / sizeof (intr_code16[0]); i++)
264         intr_code16[i] = 0;
265
266     /* Initialize interrupt names. */
267     for (i = 0; i < sizeof (intr_names17) / sizeof (intr_names17[0]); i++)
268         intr_names17[i] = "Unknown";
269
270     /* Initialize interrupt codes. */
271     for (i = 0; i < sizeof (intr_code17) / sizeof (intr_code17[0]); i++)
272         intr_code17[i] = 0;
273
274     /* Initialize interrupt names. */
275     for (i = 0; i < sizeof (intr_names18) / sizeof (intr_names18[0]); i++)
276         intr_names18[i] = "Unknown";
277
278     /* Initialize interrupt codes. */
279     for (i = 0; i < sizeof (intr_code18) / sizeof (intr_code18[0]); i++)
280         intr_code18[i] = 0;
281
282     /* Initialize interrupt names. */
283     for (i = 0; i < sizeof (intr_names19) / sizeof (intr_names19[0]); i++)
284         intr_names19[i] = "Unknown";
285
286     /* Initialize interrupt codes. */
287     for (i = 0; i < sizeof (intr_code19) / sizeof (intr_code19[0]); i++)
288         intr_code19[i] = 0;
289
290     /* Initialize interrupt names. */
291     for (i = 0; i < sizeof (intr_names20) / sizeof (intr_names20[0]); i++)
292         intr_names20[i] = "Unknown";
293
294     /* Initialize interrupt codes. */
295     for (i = 0; i < sizeof (intr_code20) / sizeof (intr_code20[0]); i++)
296         intr_code20[i] = 0;
297
298     /* Initialize interrupt names. */
299     for (i = 0; i < sizeof (intr_names21) / sizeof (intr_names21[0]); i++)
300         intr_names21[i] = "Unknown";
301
302     /* Initialize interrupt codes. */
303     for (i = 0; i < sizeof (intr_code21) / sizeof (intr_code21[0]); i++)
304         intr_code21[i] = 0;
305
306     /* Initialize interrupt names. */
307     for (i = 0; i < sizeof (intr_names22) / sizeof (intr_names22[0]); i++)
308         intr_names22[i] = "Unknown";
309
310     /* Initialize interrupt codes. */
311     for (i = 0; i < sizeof (intr_code22) / sizeof (intr_code22[0]); i++)
312         intr_code22[i] = 0;
313
314     /* Initialize interrupt names. */
315     for (i = 0; i < sizeof (intr_names23) / sizeof (intr_names23[0]); i++)
316         intr_names23[i] = "Unknown";
317
318     /* Initialize interrupt codes. */
319     for (i = 0; i < sizeof (intr_code23) / sizeof (intr_code23[0]); i++)
320         intr_code23[i] = 0;
321
322     /* Initialize interrupt names. */
323     for (i = 0; i < sizeof (intr_names24) / sizeof (intr_names24[0]); i++)
324         intr_names24[i] = "Unknown";
325
326     /* Initialize interrupt codes. */
327     for (i = 0; i < sizeof (intr_code24) / sizeof (intr_code24[0]); i++)
328         intr_code24[i] = 0;
329
330     /* Initialize interrupt names. */
331     for (i = 0; i < sizeof (intr_names25) / sizeof (intr_names25[0]); i++)
332         intr_names25[i] = "Unknown";
333
334     /* Initialize interrupt codes. */
335     for (i = 0; i < sizeof (intr_code25) / sizeof (intr_code25[0]); i++)
336         intr_code25[i] = 0;
337
338     /* Initialize interrupt names. */
339     for (i = 0; i < sizeof (intr_names26) / sizeof (intr_names26[0]); i++)
340         intr_names26[i] = "Unknown";
341
342     /* Initialize interrupt codes. */
343     for (i = 0; i < sizeof (intr_code26) / sizeof (intr_code26[0]); i++)
344         intr_code26[i] = 0;
345
346     /* Initialize interrupt names. */
347     for (i = 0; i < sizeof (intr_names27) / sizeof (intr_names27[0]); i++)
348         intr_names27[i] = "Unknown";
349
350     /* Initialize interrupt codes. */
351     for (i = 0; i < sizeof (intr_code27) / sizeof (intr_code27[0]); i++)
352         intr_code27[i] = 0;
353
354     /* Initialize interrupt names. */
355     for (i = 0; i < sizeof (intr_names28) / sizeof (intr_names28[0]); i++)
356         intr_names28[i] = "Unknown";
357
358     /* Initialize interrupt codes. */
359     for (i = 0; i < sizeof (intr_code28) / sizeof (intr_code28[0]); i++)
360         intr_code28[i] = 0;
361
362     /* Initialize interrupt names. */
363     for (i = 0; i < sizeof (intr_names29) / sizeof (intr_names29[0]); i++)
364         intr_names29[i] = "Unknown";
365
366     /* Initialize interrupt codes. */
367     for (i = 0; i < sizeof (intr_code29) / sizeof (intr_code29[0]); i++)
368         intr_code29[i] = 0;
369
370     /* Initialize interrupt names. */
371     for (i = 0; i < sizeof (intr_names30) / sizeof (intr_names30[0]); i++)
372         intr_names30[i] = "Unknown";
373
374     /* Initialize interrupt codes. */
375     for (i = 0; i < sizeof (intr_code30) / sizeof (intr_code30[0]); i++)
376         intr_code30[i] = 0;
377
378     /* Initialize interrupt names. */
379     for (i = 0; i < sizeof (intr_names31) / sizeof (intr_names31[0]); i++)
380         intr_names31[i] = "Unknown";
381
382     /* Initialize interrupt codes. */
383     for (i = 0; i < sizeof (intr_code31) / sizeof (intr_code31[0]); i++)
384         intr_code31[i] = 0;
385
386     /* Initialize interrupt names. */
387     for (i = 0; i < sizeof (intr_names32) / sizeof (intr_names32[0]); i++)
388         intr_names32[i] = "Unknown";
389
390     /* Initialize interrupt codes. */
391     for (i = 0; i < sizeof (intr_code32) / sizeof (intr_code32[0]); i++)
392         intr_code32[i] = 0;
393
394     /* Initialize interrupt names. */
395     for (i = 0; i < sizeof (intr_names33) / sizeof (intr_names33[0]); i++)
396         intr_names33[i] = "Unknown";
397
398     /* Initialize interrupt codes. */
399     for (i = 0; i < sizeof (intr_code33) / sizeof (intr_code33[0]); i++)
400         intr_code33[i] = 0;
401
402     /* Initialize interrupt names. */
403     for (i = 0; i < sizeof (intr_names34) / sizeof (intr_names34[0]); i++)
404         intr_names34[i] = "Unknown";
405
406     /* Initialize interrupt codes. */
407     for (i = 0; i < sizeof (intr_code34) / sizeof (intr_code34[0]); i++)
408         intr_code34[i] = 0;
409
410     /* Initialize interrupt names. */
411     for (i = 0; i < sizeof (intr_names35) / sizeof (intr_names35[0]); i++)
412         intr_names35[i] = "Unknown";
413
414     /* Initialize interrupt codes. */
415     for (i = 0; i < sizeof (intr_code35) / sizeof (intr_code35[0]); i++)
416         intr_code35[i] = 0;
417
418     /* Initialize interrupt names. */
419     for (i = 0; i < sizeof (intr_names36) / sizeof (intr_names36[0]); i++)
420         intr_names36[i] = "Unknown";
421
422     /* Initialize interrupt codes. */
423     for (i = 0; i < sizeof (intr_code36) / sizeof (intr_code36[0]); i++)
424         intr_code36[i] = 0;
425
426     /* Initialize interrupt names. */
427     for (i = 0; i < sizeof (intr_names37) / sizeof (intr_names37[0]); i++)
428         intr_names37[i] = "Unknown";
429
430     /* Initialize interrupt codes. */
431     for (i = 0; i < sizeof (intr_code37) / sizeof (intr_code37[0]); i++)
432         intr_code37[i] = 0;
433
434     /* Initialize interrupt names. */
435     for (i = 0; i < sizeof (intr_names38) / sizeof (intr_names38[0]); i++)
436         intr_names38[i] = "Unknown";
437
438     /* Initialize interrupt codes. */
439     for (i = 0; i < sizeof (intr_code38) / sizeof (intr_code38[0]); i++)
440         intr_code38[i] = 0;
441
442     /* Initialize interrupt names. */
443     for (i = 0; i < sizeof (intr_names39) / sizeof (intr_names39[0]); i++)
444         intr_names39[i] = "Unknown";
445
446     /* Initialize interrupt codes. */
447     for (i = 0; i < sizeof (intr_code39) / sizeof (intr_code39[0]); i++)
448         intr_code39[i] = 0;
449
450     /* Initialize interrupt names. */
451     for (i = 0; i < sizeof (intr_names40) / sizeof (intr_names40[0]); i++)
452         intr_names40[i] = "Unknown";
453
454     /* Initialize interrupt codes. */
455     for (i = 0; i < sizeof (intr_code40) / sizeof (intr_code40[0]); i++)
456         intr_code40[i] = 0;
457
458     /* Initialize interrupt names. */
459     for (i = 0; i < sizeof (intr_names41) / sizeof (intr_names41[0]); i++)
460         intr_names41[i] = "Unknown";
461
462     /* Initialize interrupt codes. */
463     for (i = 0; i < sizeof (intr_code41) / sizeof (intr_code41[0]); i++)
464         intr_code41[i] = 0;
465
466     /* Initialize interrupt names. */
467     for (i = 0; i < sizeof (intr_names42) / sizeof (intr_names42[0]); i++)
468         intr_names42[i] = "Unknown";
469
470     /* Initialize interrupt codes. */
471     for (i = 0; i < sizeof (intr_code42) / sizeof (intr_code42[0]); i++)
472         intr_code42[i] = 0;
473
474     /* Initialize interrupt names. */
475     for (i = 0; i < sizeof (intr_names43) / sizeof (intr_names43[0]); i++)
476         intr_names43[i] = "Unknown";
477
478     /* Initialize interrupt codes. */
479     for (i = 0; i < sizeof (intr_code43) / sizeof (intr_code43[0]); i++)
480         intr_code43[i] = 0;
481
482     /* Initialize interrupt names. */
483     for (i = 0; i < sizeof (intr_names44) / sizeof (intr_names44[0]); i++)
484         intr_names44[i] = "Unknown";
485
486     /* Initialize interrupt codes. */
487     for (i = 0; i < sizeof (intr_code44) / sizeof (intr_code44[0]); i++)
488         intr_code44[i] = 0;
489
490     /* Initialize interrupt names. */
491     for (i = 0; i < sizeof (intr_names45) / sizeof (intr_names45[0]); i++)
492         intr_names45[i] = "Unknown";
493
494     /* Initialize interrupt codes. */
495     for (i = 0; i < sizeof (intr_code45) / sizeof (intr_code45[0]); i++)
496         intr_code45[i] = 0;
497
498     /* Initialize interrupt names. */
499     for (i = 0; i < sizeof (intr_names46) / sizeof (intr_names46[0]); i++)
500         intr_names46[i] = "Unknown";
501
502     /* Initialize interrupt codes. */
503     for (i = 0; i < sizeof (intr_code46) / sizeof (intr_code46[0]); i++)
504         intr_code46[i] = 0;
505
506     /* Initialize interrupt names. */
507     for (i = 0; i < sizeof (intr_names47) / sizeof (intr_names47[0]); i++)
508         intr_names47[i] = "Unknown";
509
510     /* Initialize interrupt codes. */
511     for (i = 0; i < sizeof (intr_code47) / sizeof (intr_code47[0]); i++)
512         intr_code47[i] = 0;
513
514     /* Initialize interrupt names. */
515     for (i = 0; i < sizeof (intr_names48) / sizeof (intr_names48[0]); i++)
516         intr_names48[i] = "Unknown";
517
518     /* Initialize interrupt codes. */
519     for (i = 0; i < sizeof (intr_code48) / sizeof (intr_code48[0]); i++)
520         intr_code48[i] = 0;
521
522     /* Initialize interrupt names. */
523     for (i = 0; i < sizeof (intr_names49) / sizeof (intr_names49[0]); i++)
524         intr_names49[i] = "Unknown";
525
526     /* Initialize interrupt codes. */
527     for (i = 0; i < sizeof (intr_code49) / sizeof (intr_code49[0]); i++)
528         intr_code49[i] = 0;
529
530     /* Initialize interrupt names. */
531     for (i = 0; i < sizeof (intr_names50) / sizeof (intr_names50[0]); i++)
532         intr_names50[i] = "Unknown";
533
534     /* Initialize interrupt codes. */
535     for (i = 0; i < sizeof (intr_code50) / sizeof (intr_code50[0]); i++)
536         intr_code50[i] = 0;
537
538     /* Initialize interrupt names. */
539     for (i = 0; i < sizeof (intr_names51) / sizeof (intr_names51[0]); i++)
540         intr_names51[i] = "Unknown";
541
542     /* Initialize interrupt codes. */
543     for (i = 0; i < sizeof (intr_code51) / sizeof (intr_code51[0]); i++)
544         intr_code51[i] = 0;
545
546     /* Initialize interrupt names. */
547     for (i = 0; i < sizeof (intr_names52) / sizeof (intr_names52[0]); i++)
548         intr_names52[i] = "Unknown";
549
550     /* Initialize interrupt codes. */
551     for (i = 0; i < sizeof (intr_code52) / sizeof (intr_code52[0]); i++)
552         intr_code52[i] = 0;
553
554     /* Initialize interrupt names. */
555     for (i = 0; i < sizeof (intr_names53) / sizeof (intr_names53[0]); i++)
556         intr_names53[i] = "Unknown";
557
558     /* Initialize interrupt codes. */
559     for (i = 0; i < sizeof (intr_code53) / sizeof (intr_code53[0]); i++)
560         intr_code53[i] = 0;
561
562     /* Initialize interrupt names. */
563     for (i = 0; i < sizeof (intr_names54) / sizeof (intr_names54[0]); i++)
564         intr_names54[i] = "Unknown";
565
566     /* Initialize interrupt codes. */
567     for (i = 0; i < sizeof (intr_code54) / sizeof (intr_code54[0]); i++)
568         intr_code54[i] = 0;
569
570     /* Initialize interrupt names. */
571     for (i = 0; i < sizeof (intr_names55) / sizeof (intr_names55[0]); i++)
572         intr_names55[i] = "Unknown";
573
574     /* Initialize interrupt codes. */
575     for (i = 0; i < sizeof (intr_code55) / sizeof (intr_code55[0]); i++)
576         intr_code55[i] = 0;
577
578     /* Initialize interrupt names. */
579     for (i = 0; i < sizeof (intr_names56) / sizeof (intr_names56[0]); i++)
580         intr_names56[i] = "Unknown";
581
582     /* Initialize interrupt codes. */
583     for (i = 0; i < sizeof (intr_code56) / sizeof (intr_code56[0]); i++)
584         intr_code56[i] = 0;
585
586     /* Initialize interrupt names. */
587     for (i = 0; i < sizeof (intr_names57) / sizeof (intr_names57[0]); i++)
588         intr_names57[i] = "Unknown";
589
590     /* Initialize interrupt codes. */
591     for (i = 0; i < sizeof (intr_code57) / sizeof (intr_code57[0]); i++)
592         intr_code57[i] = 0;
593
594     /* Initialize interrupt names. */
595     for (i = 0; i < sizeof (intr_names58) / sizeof (intr_names58[0]); i++)
596         intr_names58[i] = "Unknown";
597
598     /* Initialize interrupt codes. */
599     for (i = 0; i < sizeof (intr_code58) / sizeof (intr_code58[0]); i++)
600         intr_code58[i] = 0;
601
602     /* Initialize interrupt names. */
603     for (i = 0; i < sizeof (intr_names59) / sizeof (intr_names59[0]); i++)
604         intr_names59[i] = "Unknown";
605
606     /* Initialize interrupt codes. */
607     for (i = 0; i < sizeof (intr_code59) / sizeof (intr_code59[0]); i++)
608         intr_code59[i] = 0;
609
610     /* Initialize interrupt names. */
611     for (i = 0; i < sizeof (intr_names60) / sizeof (intr_names60[0]); i++)
612         intr_names60[i] = "Unknown";
613
614     /* Initialize interrupt codes. */
615     for (i = 0; i < sizeof (intr_code60) / sizeof (intr_code60[0]); i++)
616         intr_code60[i] = 0;
617
618     /* Initialize interrupt names. */
619     for (i = 0; i < sizeof (intr_names61) / sizeof (intr_names61[0]); i++)
620         intr_names61[i] = "Unknown";
621
622     /* Initialize interrupt codes. */
623     for (i = 0; i < sizeof (intr_code61) / sizeof (intr_code61[0]); i++)
624         intr_code61[i] = 0;
625
626     /* Initialize interrupt names. */
627     for (i = 0; i < sizeof (intr_names62) / sizeof (intr_names62[0]); i++)
628         intr_names62[i] = "Unknown";
629
630     /* Initialize interrupt codes. */
631     for (i = 0; i < sizeof (intr_code62) / sizeof (intr_code62[0]); i++)
632         intr_code62[i] = 0;
633
634     /* Initialize interrupt names. */
635     for (i = 0; i < sizeof (intr_names63) / sizeof (intr_names63[0]); i++)
636         intr_names63[i] = "Unknown";
637
638     /* Initialize interrupt codes. */
639     for (i = 0; i < sizeof (intr_code63) / sizeof (intr_code63[0]); i++)
640         intr_code63[i] = 0;
641
642     /* Initialize interrupt names. */
643     for (i = 0; i < sizeof (intr_names64) / sizeof (intr_names64[0]); i++)
644         intr_names64[i] = "Unknown";
645
646     /* Initialize interrupt codes. */
647     for (i = 0; i < sizeof (intr_code64) / sizeof (intr_code64[0]); i++)
648         intr_code64[i] = 0;
649
650     /* Initialize interrupt names. */
651     for (i = 0; i < sizeof (intr_names65) / sizeof (intr_names65[0]); i++)
652         intr_names65[i] = "Unknown";
653
654     /* Initialize interrupt codes. */
655     for (i = 0; i < sizeof (intr_code65) / sizeof (intr_code65[0]); i++)
656         intr_code65[i] = 0;
657
658     /* Initialize interrupt names. */
659     for (i = 0; i < sizeof (intr_names66) / sizeof (intr_names66[0]); i++)
660         intr_names66[i] = "Unknown";
661
662     /* Initialize interrupt codes. */
663     for (i = 0; i < sizeof (intr_code66) / sizeof (intr_code66[0]); i++)
664         intr_code66[i] = 0;
665
666     /* Initialize interrupt names. */
667     for (i = 0; i < sizeof (intr_names67) / sizeof (intr_names67[0]); i++)
668         intr_names67[i] = "Unknown";
669
670     /* Initialize interrupt codes. */
671     for (i = 0; i < sizeof (intr_code67) / sizeof (intr_code67[0]); i++)
672         intr_code67[i] = 0;
673
674     /* Initialize interrupt names. */
675     for (i = 0; i < sizeof (intr_names68) / sizeof (intr_names68[0]); i++)
676         intr_names68[i] = "Unknown";
677
678     /* Initialize interrupt codes. */
679     for (i = 0; i < sizeof (intr_code68) / sizeof (intr_code68[0]); i++)
680         intr_code68[i] = 0;
681
682     /* Initialize interrupt names. */
683     for (i = 0; i < sizeof (intr_names69) / sizeof (intr_names69[0]); i++)
684         intr_names69[i] = "Unknown";
685
686     /* Initialize interrupt codes. */
687     for (i = 0; i < sizeof (intr_code69) / sizeof (intr_code69[0]); i++)
688         intr_code69[i] = 0;
689
690     /* Initialize interrupt names. */
691     for (i = 0; i < sizeof (intr_names70) / sizeof (intr_names70[0]); i++)
692         intr_names70[i] = "Unknown";
693
694     /* Initialize interrupt codes. */
695     for (i = 0; i < sizeof (intr_code70) / sizeof (intr_code70[0]); i++)
696         intr_code70[i] = 0;
697
698     /* Initialize interrupt names. */
699     for (i = 0; i < sizeof (intr_names71) / sizeof (intr_names71[0]); i++)
700         intr_names71[i] = "Unknown";
701
702     /* Initialize interrupt codes. */
703     for (i = 0; i < sizeof (intr_code71) / sizeof (intr_code71[0]); i++)
704         intr_code71[i] = 0;
705
706     /* Initialize interrupt names. */
707     for (i = 0; i < sizeof (intr_names72) / sizeof (intr_names72[0]); i++)
708         intr_names72[i] = "Unknown";
709
710     /* Initialize interrupt codes. */
711     for (i = 0; i < sizeof (intr_code72) / sizeof (intr_code72[0]); i++)
712         intr_code72[i] = 0;
713
714     /* Initialize interrupt names. */
715     for (i = 0; i < sizeof (intr_names73) / sizeof (intr_names73[0]); i++)
716         intr_names73[i] = "Unknown";
717
718     /* Initialize interrupt codes. */
719     for (i = 0; i < sizeof (intr_code73) / sizeof (intr_code73[0]); i++)
720         intr_code73[i] = 0;
721
722     /* Initialize interrupt names. */
723     for (i = 0; i < sizeof (intr_names74) / sizeof (intr_names74[0]); i++)
724         intr_names74[i] = "Unknown";
725
726     /* Initialize interrupt codes. */
727     for (i = 0; i < sizeof (intr_code74) / sizeof (intr_code74[0]); i++)
728         intr_code74[i] = 0;
729
730     /* Initialize interrupt names. */
731     for (i = 0; i < sizeof (intr_names75) / sizeof (intr_names75[0]); i++)
732         intr_names75[i] = "Unknown";
733
734     /* Initialize interrupt codes. */
735     for (i = 0; i < sizeof (intr_code75) / sizeof (intr_code75[0]); i++)
736         intr_code75[i] = 0;
737
738     /* Initialize interrupt names. */
739     for (i = 0; i < sizeof (intr_names76) / sizeof (intr_names76[0]); i++)
740         intr_names76[i] = "Unknown";
741
742     /* Initialize interrupt codes. */
743     for (i = 0; i < sizeof (intr_code76) / sizeof (intr_code76[0]); i++)
744         intr_code76[i] = 0;
745
746     /* Initialize interrupt names. */
747     for (i = 0; i < sizeof (intr_names77) / sizeof (intr_names77[0]); i++)
748         intr_names77[i] = "Unknown";
749
750     /* Initialize interrupt codes. */
751     for (i = 0; i < sizeof (intr_code77) / sizeof (intr_code77[0]); i++)
752         intr_code77[i] = 0;
753
754     /* Initialize interrupt names. */
755     for (i = 0; i < sizeof (intr_names78) / sizeof (intr_names78[0]); i++)
756         intr_names78[i] = "Unknown";
757
758     /* Initialize interrupt codes. */
759     for (i = 0; i < sizeof (intr_code78) / sizeof (intr_code78[0]); i++)
760         intr_code78[i] = 0;
761
762     /* Initialize interrupt names. */
763     for (i = 0; i < sizeof (intr_names79) / sizeof (intr_names79[0]); i++)
764         intr_names79[i] = "Unknown";
765
766     /* Initialize interrupt codes. */
767     for (i = 0; i < sizeof (intr_code79) / sizeof (intr_code79[0]); i++)
768         intr_code79[i] = 0;
769
770     /* Initialize interrupt names. */
771     for (i = 0; i < sizeof (intr_names80) / sizeof (intr_names80[0]); i++)
772         intr_names80[i] = "Unknown";
773
774     /* Initialize interrupt codes. */
775     for (i = 0;
```

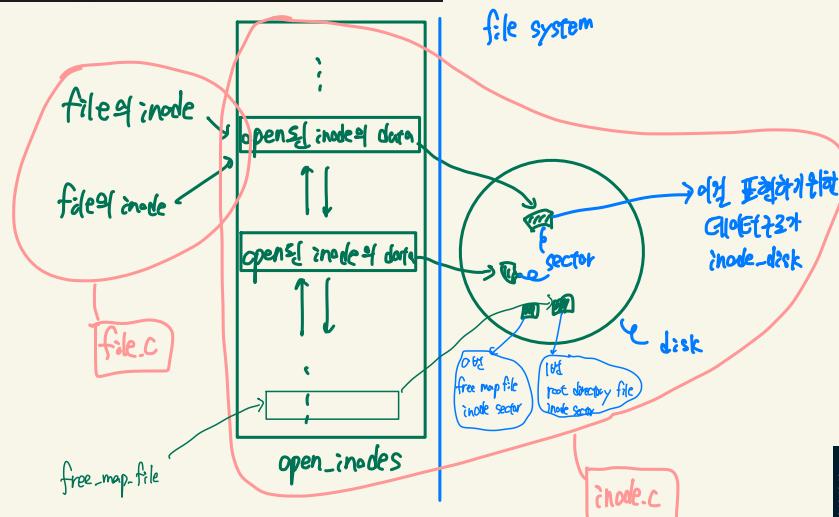
```

// Task-State Segment (TSS)
// Instances of the TSS, an x86-specific structure, are used to
// store thread-specific information. The processor uses the TSS
// to store thread-specific information such as the stack pointer
// and interrupt handler. However, for various reasons including
// compatibility with legacy software, the processor also supports
// thread-local storage (TLS). In user mode (Ring 3), the processor
// conducts the load and save members of the current TSS to
// store thread-local data. In kernel mode (Ring 0), however, this
// is not possible. Instead, the kernel must create a TSS and initialize at least those fields, and
// then switch to kernel mode to use them.
//
// When an interrupt or an interrupt or trap gate
// occurs, the processor handles it by switching to the stack
// specified in the TSS. This is done via the instruction pushf.
// The processor then pushes the current stack pointer onto the stack
// specified in the TSS. It then pushes the current stack segment selector
// onto the stack specified in the TSS. Finally, it pushes the current
// stack limit onto the stack specified in the TSS. Thus, when the
// interrupt returns, the processor will always choose the stack
// specified in the TSS. This is how the kernel maintains its own stack
// (the calls to in_thread_schedule_tail in thread.c).
//
// See [IA32] 6.2.12 "Task-State Segment (TSS)" for a
// detailed description of the TSS. For more information about
// interrupt-handling procedures, see [IA32] 6.2.13 "Interrupt-handling
// procedures" for a description of when and
// why the processor switches context during an interrupt.
// struct tss {
//     uint32_t back_link; //0
//     uint32_t reserved; //1
//     uint32_t ss; //2
//     uint32_t es; //3
//     uint32_t cs; //4
//     uint32_t ss2; //5
//     uint32_t es2; //6
//     uint32_t cs2; //7
//     uint32_t ss3; //8
//     uint32_t es3; //9
//     uint32_t cs3; //10
//     uint32_t ss4; //11
//     uint32_t es4; //12
//     uint32_t cs4; //13
//     uint32_t ss5; //14
//     uint32_t es5; //15
//     uint32_t cs5; //16
//     uint32_t ss6; //17
//     uint32_t es6; //18
//     uint32_t cs6; //19
//     uint32_t ss7; //20
//     uint32_t es7; //21
//     uint32_t cs7; //22
//     uint32_t ss8; //23
//     uint32_t es8; //24
//     uint32_t cs8; //25
//     uint32_t ss9; //26
//     uint32_t es9; //27
//     uint32_t cs9; //28
//     uint32_t ss10; //29
//     uint32_t es10; //30
//     uint32_t cs10; //31
//     uint32_t ss11; //32
//     uint32_t es11; //33
//     uint32_t cs11; //34
//     uint32_t ss12; //35
//     uint32_t es12; //36
//     uint32_t cs12; //37
//     uint32_t ss13; //38
//     uint32_t es13; //39
//     uint32_t cs13; //40
//     uint32_t ss14; //41
//     uint32_t es14; //42
//     uint32_t cs14; //43
//     uint32_t ss15; //44
//     uint32_t es15; //45
//     uint32_t cs15; //46
//     uint32_t ss16; //47
//     uint32_t es16; //48
//     uint32_t cs16; //49
//     uint32_t ss17; //50
//     uint32_t es17; //51
//     uint32_t cs17; //52
//     uint32_t ss18; //53
//     uint32_t es18; //54
//     uint32_t cs18; //55
//     uint32_t ss19; //56
//     uint32_t es19; //57
//     uint32_t cs19; //58
//     uint32_t ss20; //59
//     uint32_t es20; //60
//     uint32_t cs20; //61
//     uint32_t ss21; //62
//     uint32_t es21; //63
//     uint32_t cs21; //64
//     uint32_t ss22; //65
//     uint32_t es22; //66
//     uint32_t cs22; //67
//     uint32_t ss23; //68
//     uint32_t es23; //69
//     uint32_t cs23; //70
//     uint32_t ss24; //71
//     uint32_t es24; //72
//     uint32_t cs24; //73
//     uint32_t ss25; //74
//     uint32_t es25; //75
//     uint32_t cs25; //76
//     uint32_t ss26; //77
//     uint32_t es26; //78
//     uint32_t cs26; //79
//     uint32_t ss27; //80
//     uint32_t es27; //81
//     uint32_t cs27; //82
//     uint32_t ss28; //83
//     uint32_t es28; //84
//     uint32_t cs28; //85
//     uint32_t ss29; //86
//     uint32_t es29; //87
//     uint32_t cs29; //88
//     uint32_t ss30; //89
//     uint32_t es30; //90
//     uint32_t cs30; //91
//     uint32_t ss31; //92
//     uint32_t es31; //93
//     uint32_t cs31; //94
//     uint32_t ss32; //95
//     uint32_t es32; //96
//     uint32_t cs32; //97
//     uint32_t ss33; //98
//     uint32_t es33; //99
//     uint32_t cs33; //100
//     uint32_t ss34; //101
//     uint32_t es34; //102
//     uint32_t cs34; //103
//     uint32_t ss35; //104
//     uint32_t es35; //105
//     uint32_t cs35; //106
//     uint32_t ss36; //107
//     uint32_t es36; //108
//     uint32_t cs36; //109
//     uint32_t ss37; //110
//     uint32_t es37; //111
//     uint32_t cs37; //112
//     uint32_t ss38; //113
//     uint32_t es38; //114
//     uint32_t cs38; //115
//     uint32_t ss39; //116
//     uint32_t es39; //117
//     uint32_t cs39; //118
//     uint32_t ss40; //119
//     uint32_t es40; //120
//     uint32_t cs40; //121
//     uint32_t ss41; //122
//     uint32_t es41; //123
//     uint32_t cs41; //124
//     uint32_t ss42; //125
//     uint32_t es42; //126
//     uint32_t cs42; //127
//     uint32_t ss43; //128
//     uint32_t es43; //129
//     uint32_t cs43; //130
//     uint32_t ss44; //131
//     uint32_t es44; //132
//     uint32_t cs44; //133
//     uint32_t ss45; //134
//     uint32_t es45; //135
//     uint32_t cs45; //136
//     uint32_t ss46; //137
//     uint32_t es46; //138
//     uint32_t cs46; //139
//     uint32_t ss47; //140
//     uint32_t es47; //141
//     uint32_t cs47; //142
//     uint32_t ss48; //143
//     uint32_t es48; //144
//     uint32_t cs48; //145
//     uint32_t ss49; //146
//     uint32_t es49; //147
//     uint32_t cs49; //148
//     uint32_t ss50; //149
//     uint32_t es50; //150
//     uint32_t cs50; //151
//     uint32_t ss51; //152
//     uint32_t es51; //153
//     uint32_t cs51; //154
//     uint32_t ss52; //155
//     uint32_t es52; //156
//     uint32_t cs52; //157
//     uint32_t ss53; //158
//     uint32_t es53; //159
//     uint32_t cs53; //160
//     uint32_t ss54; //161
//     uint32_t es54; //162
//     uint32_t cs54; //163
//     uint32_t ss55; //164
//     uint32_t es55; //165
//     uint32_t cs55; //166
//     uint32_t ss56; //167
//     uint32_t es56; //168
//     uint32_t cs56; //169
//     uint32_t ss57; //170
//     uint32_t es57; //171
//     uint32_t cs57; //172
//     uint32_t ss58; //173
//     uint32_t es58; //174
//     uint32_t cs58; //175
//     uint32_t ss59; //176
//     uint32_t es59; //177
//     uint32_t cs59; //178
//     uint32_t ss60; //179
//     uint32_t es60; //180
//     uint32_t cs60; //181
//     uint32_t ss61; //182
//     uint32_t es61; //183
//     uint32_t cs61; //184
//     uint32_t ss62; //185
//     uint32_t es62; //186
//     uint32_t cs62; //187
//     uint32_t ss63; //188
//     uint32_t es63; //189
//     uint32_t cs63; //190
//     uint32_t ss64; //191
//     uint32_t es64; //192
//     uint32_t cs64; //193
//     uint32_t ss65; //194
//     uint32_t es65; //195
//     uint32_t cs65; //196
//     uint32_t ss66; //197
//     uint32_t es66; //198
//     uint32_t cs66; //199
//     uint32_t ss67; //200
//     uint32_t es67; //201
//     uint32_t cs67; //202
//     uint32_t ss68; //203
//     uint32_t es68; //204
//     uint32_t cs68; //205
//     uint32_t ss69; //206
//     uint32_t es69; //207
//     uint32_t cs69; //208
//     uint32_t ss70; //209
//     uint32_t es70; //210
//     uint32_t cs70; //211
//     uint32_t ss71; //212
//     uint32_t es71; //213
//     uint32_t cs71; //214
//     uint32_t ss72; //215
//     uint32_t es72; //216
//     uint32_t cs72; //217
//     uint32_t ss73; //218
//     uint32_t es73; //219
//     uint32_t cs73; //220
//     uint32_t ss74; //221
//     uint32_t es74; //222
//     uint32_t cs74; //223
//     uint32_t ss75; //224
//     uint32_t es75; //225
//     uint32_t cs75; //226
//     uint32_t ss76; //227
//     uint32_t es76; //228
//     uint32_t cs76; //229
//     uint32_t ss77; //230
//     uint32_t es77; //231
//     uint32_t cs77; //232
//     uint32_t ss78; //233
//     uint32_t es78; //234
//     uint32_t cs78; //235
//     uint32_t ss79; //236
//     uint32_t es79; //237
//     uint32_t cs79; //238
//     uint32_t ss80; //239
//     uint32_t es80; //240
//     uint32_t cs80; //241
//     uint32_t ss81; //242
//     uint32_t es81; //243
//     uint32_t cs81; //244
//     uint32_t ss82; //245
//     uint32_t es82; //246
//     uint32_t cs82; //247
//     uint32_t ss83; //248
//     uint32_t es83; //249
//     uint32_t cs83; //250
//     uint32_t ss84; //251
//     uint32_t es84; //252
//     uint32_t cs84; //253
//     uint32_t ss85; //254
//     uint32_t es85; //255
//     uint32_t cs85; //256
//     uint32_t ss86; //257
//     uint32_t es86; //258
//     uint32_t cs86; //259
//     uint32_t ss87; //260
//     uint32_t es87; //261
//     uint32_t cs87; //262
//     uint32_t ss88; //263
//     uint32_t es88; //264
//     uint32_t cs88; //265
//     uint32_t ss89; //266
//     uint32_t es89; //267
//     uint32_t cs89; //268
//     uint32_t ss90; //269
//     uint32_t es90; //270
//     uint32_t cs90; //271
//     uint32_t ss91; //272
//     uint32_t es91; //273
//     uint32_t cs91; //274
//     uint32_t ss92; //275
//     uint32_t es92; //276
//     uint32_t cs92; //277
//     uint32_t ss93; //278
//     uint32_t es93; //279
//     uint32_t cs93; //280
//     uint32_t ss94; //281
//     uint32_t es94; //282
//     uint32_t cs94; //283
//     uint32_t ss95; //284
//     uint32_t es95; //285
//     uint32_t cs95; //286
//     uint32_t ss96; //287
//     uint32_t es96; //288
//     uint32_t cs96; //289
//     uint32_t ss97; //290
//     uint32_t es97; //291
//     uint32_t cs97; //292
//     uint32_t ss98; //293
//     uint32_t es98; //294
//     uint32_t cs98; //295
//     uint32_t ss99; //296
//     uint32_t es99; //297
//     uint32_t cs99; //298
//     uint32_t ss100; //299
//     uint32_t es100; //300
//     uint32_t cs100; //301
//     uint32_t ss101; //302
//     uint32_t es101; //303
//     uint32_t cs101; //304
//     uint32_t ss102; //305
//     uint32_t es102; //306
//     uint32_t cs102; //307
//     uint32_t ss103; //308
//     uint32_t es103; //309
//     uint32_t cs103; //310
//     uint32_t ss104; //311
//     uint32_t es104; //312
//     uint32_t cs104; //313
//     uint32_t ss105; //314
//     uint32_t es105; //315
//     uint32_t cs105; //316
//     uint32_t ss106; //317
//     uint32_t es106; //318
//     uint32_t cs106; //319
//     uint32_t ss107; //320
//     uint32_t es107; //321
//     uint32_t cs107; //322
//     uint32_t ss108; //323
//     uint32_t es108; //324
//     uint32_t cs108; //325
//     uint32_t ss109; //326
//     uint32_t es109; //327
//     uint32_t cs109; //328
//     uint32_t ss110; //329
//     uint32_t es110; //330
//     uint32_t cs110; //331
//     uint32_t ss111; //332
//     uint32_t es111; //333
//     uint32_t cs111; //334
//     uint32_t ss112; //335
//     uint32_t es112; //336
//     uint32_t cs112; //337
//     uint32_t ss113; //338
//     uint32_t es113; //339
//     uint32_t cs113; //340
//     uint32_t ss114; //341
//     uint32_t es114; //342
//     uint32_t cs114; //343
//     uint32_t ss115; //344
//     uint32_t es115; //345
//     uint32_t cs115; //346
//     uint32_t ss116; //347
//     uint32_t es116; //348
//     uint32_t cs116; //349
//     uint32_t ss117; //350
//     uint32_t es117; //351
//     uint32_t cs117; //352
//     uint32_t ss118; //353
//     uint32_t es118; //354
//     uint32_t cs118; //355
//     uint32_t ss119; //356
//     uint32_t es119; //357
//     uint32_t cs119; //358
//     uint32_t ss120; //359
//     uint32_t es120; //360
//     uint32_t cs120; //361
//     uint32_t ss121; //362
//     uint32_t es121; //363
//     uint32_t cs121; //364
//     uint32_t ss122; //365
//     uint32_t es122; //366
//     uint32_t cs122; //367
//     uint32_t ss123; //368
//     uint32_t es123; //369
//     uint32_t cs123; //370
//     uint32_t ss124; //371
//     uint32_t es124; //372
//     uint32_t cs124; //373
//     uint32_t ss125; //374
//     uint32_t es125; //375
//     uint32_t cs125; //376
//     uint32_t ss126; //377
//     uint32_t es126; //378
//     uint32_t cs126; //379
//     uint32_t ss127; //380
//     uint32_t es127; //381
//     uint32_t cs127; //382
//     uint32_t ss128; //383
//     uint32_t es128; //384
//     uint32_t cs128; //385
//     uint32_t ss129; //386
//     uint32_t es129; //387
//     uint32_t cs129; //388
//     uint32_t ss130; //389
//     uint32_t es130; //390
//     uint32_t cs130; //391
//     uint32_t ss131; //392
//     uint32_t es131; //393
//     uint32_t cs131; //394
//     uint32_t ss132; //395
//     uint32_t es132; //396
//     uint32_t cs132; //397
//     uint32_t ss133; //398
//     uint32_t es133; //399
//     uint32_t cs133; //400
//     uint32_t ss134; //401
//     uint32_t es134; //402
//     uint32_t cs134; //403
//     uint32_t ss135; //404
//     uint32_t es135; //405
//     uint32_t cs135; //406
//     uint32_t ss136; //407
//     uint32_t es136; //408
//     uint32_t cs136; //409
//     uint32_t ss137; //410
//     uint32_t es137; //411
//     uint32_t cs137; //412
//     uint32_t ss138; //413
//     uint32_t es138; //414
//     uint32_t cs138; //415
//     uint32_t ss139; //416
//     uint32_t es139; //417
//     uint32_t cs139; //418
//     uint32_t ss140; //419
//     uint32_t es140; //420
//     uint32_t cs140; //421
//     uint32_t ss141; //422
//     uint32_t es141; //423
//     uint32_t cs141; //424
//     uint32_t ss142; //425
//     uint32_t es142; //426
//     uint32_t cs142; //427
//     uint32_t ss143; //428
//     uint32_t es143; //429
//     uint32_t cs143; //430
//     uint32_t ss144; //431
//     uint32_t es144; //432
//     uint32_t cs144; //433
//     uint32_t ss145; //434
//     uint32_t es145; //435
//     uint32_t cs145; //436
//     uint32_t ss146; //437
//     uint32_t es146; //438
//     uint32_t cs146; //439
//     uint32_t ss147; //440
//     uint32_t es147; //441
//     uint32_t cs147; //442
//     uint32_t ss148; //443
//     uint32_t es148; //444
//     uint32_t cs148; //445
//     uint32_t ss149; //446
//     uint32_t es149; //447
//     uint32_t cs149; //448
//     uint32_t ss150; //449
//     uint32_t es150; //450
//     uint32_t cs150; //451
//     uint32_t ss151; //452
//     uint32_t es151; //453
//     uint32_t cs151; //454
//     uint32_t ss152; //455
//     uint32_t es152; //456
//     uint32_t cs152; //457
//     uint32_t ss153; //458
//     uint32_t es153; //459
//     uint32_t cs153; //460
//     uint32_t ss154; //461
//     uint32_t es154; //462
//     uint32_t cs154; //463
//     uint32_t ss155; //464
//     uint32_t es155; //465
//     uint32_t cs155; //466
//     uint32_t ss156; //467
//     uint32_t es156; //468
//     uint32_t cs156; //469
//     uint32_t ss157; //470
//     uint32_t es157; //471
//     uint32_t cs158; //472
//     uint32_t ss159; //473
//     uint32_t es159; //474
//     uint32_t cs159; //475
//     uint32_t ss160; //476
//     uint32_t es160; //477
//     uint32_t cs160; //478
//     uint32_t ss161; //479
//     uint32_t es161; //480
//     uint32_t cs161; //481
//     uint32_t ss162; //482
//     uint32_t es162; //483
//     uint32_t cs162; //484
//     uint32_t ss163; //485
//     uint32_t es163; //486
//     uint32_t cs163; //487
//     uint32_t ss164; //488
//     uint32_t es164; //489
//     uint32_t cs164; //490
//     uint32_t ss165; //491
//     uint32_t es165; //492
//     uint32_t cs165; //493
//     uint32_t ss166; //494
//     uint32_t es166; //495
//     uint32_t cs166; //496
//     uint32_t ss167; //497
//     uint32_t es167; //498
//     uint32_t cs168; //499
//     uint32_t ss168; //500
//     uint32_t es169; //501
//     uint32_t cs169; //502
//     uint32_t ss170; //503
//     uint32_t es171; //504
//     uint32_t cs172; //505
//     uint32_t ss173; //506
//     uint32_t es174; //507
//     uint32_t cs175; //508
//     uint32_t ss176; //509
//     uint32_t es177; //510
//     uint32_t cs178; //511
//     uint32_t ss179; //512
//     uint32_t es180; //513
//     uint32_t cs181; //514
//     uint32_t ss182; //515
//     uint32_t es183; //516
//     uint32_t cs184; //517
//     uint32_t ss185; //518
//     uint32_t es186; //519
//     uint32_t cs187; //520
//     uint32_t ss188; //521
//     uint32_t es189; //522
//     uint32_t cs190; //523
//     uint32_t ss191; //524
//     uint32_t es192; //525
//     uint32_t cs193; //526
//     uint32_t ss194; //527
//     uint32_t es195; //528
//     uint32_t cs196; //529
//     uint32_t ss197; //530
//     uint32_t es198; //531
//     uint32_t cs199; //532
//     uint32_t ss200; //533
//     uint32_t es201; //534
//     uint32_t cs202; //535
//     uint32_t ss203; //536
//     uint32_t es204; //537
//     uint32_t cs205; //538
//     uint32_t ss206; //539
//     uint32_t es207; //540
//     uint32_t cs208; //541
//     uint32_t ss209; //542
//     uint32_t es210; //543
//     uint32_t cs211; //544
//     uint32_t ss212; //545
//     uint32_t es213; //546
//     uint32_t cs214; //547
//     uint32_t ss215; //548
//     uint32_t es216; //549
//     uint32_t cs217; //550
//     uint32_t ss218; //551
//     uint32_t es219; //552
//     uint32_t cs220; //553
//     uint32_t ss221; //554
//     uint32_t es222; //555
//     uint32_t cs223; //556
//     uint32_t ss224; //557
//     uint32_t es225; //558
//     uint32_t cs226; //559
//     uint32_t ss227; //560
//     uint32_t es228; //561
//     uint32_t cs229; //562
//     uint32_t ss230; //563
//     uint32_t es231; //564
//     uint32_t cs232; //565
//     uint32_t ss233; //566
//     uint32_t es234; //567
//     uint32_t cs235; //568
//     uint32_t ss236; //569
//     uint32_t es237; //570
//     uint32_t cs238; //571
//     uint32_t ss239; //572
//     uint32_t es240; //573
//     uint32_t cs241; //574
//     uint32_t ss242; //575
//     uint32_t es243; //576
//     uint32_t cs244; //577
//     uint32_t ss245; //578
//     uint32_t es246; //579
//     uint32_t cs247; //580
//     uint32_t ss248; //581
//     uint32_t es249; //582
//     uint32_t cs250; //583
//     uint32_t ss251; //584
//     uint32_t es252; //585
//     uint32_t cs253; //586
//     uint32_t ss254; //587
//     uint32_t es255; //588
//     uint32_t cs256; //589
//     uint32_t ss257; //590
//     uint32_t es258; //591
//     uint32_t cs259; //592
//     uint32_t ss260; //593
//     uint32_t es261; //594
//     uint32_t cs262; //595
//     uint32_t ss263; //596
//     uint32_t es264; //597
//     uint32_t cs265; //598
//     uint32_t ss266; //599
//     uint32_t es267; //600
//     uint32_t cs268; //601
//     uint32_t ss269; //602
//     uint32_t es270; //603
//     uint32_t cs271; //604
//     uint32_t ss272; //605
//     uint32_t es273; //606
//     uint32_t cs274; //607
//     uint32_t ss275; //608
//     uint32_t es276; //609
//     uint32_t cs277; //610
//     uint32_t ss278; //611
//     uint32_t es279; //612
//     uint32_t cs280; //613
//     uint32_t ss281; //614
//     uint32_t es282; //615
//     uint32_t cs283; //616
//     uint32_t ss284; //617
//     uint32_t es285; //618
//     uint32_t cs286; //619
//     uint32_t ss287; //620
//     uint32_t es288; //621
//     uint32_t cs289; //622
//     uint32_t ss290; //623
//     uint32_t es291; //624
//     uint32_t cs292; //625
//     uint32_t ss293; //626
//     uint32_t es294; //627
//     uint32_t cs295; //628
//     uint32_t ss296; //629
//     uint32_t es297; //630
//     uint32_t cs298; //631
//     uint32_t ss299; //632
//     uint32_t es299; //633
//     uint32_t cs299; //634
//     uint32_t ss299; //635
//     uint32_t es299; //636
//     uint32_t cs299; //637
//     uint32_t ss299; //638
//     uint32_t es299; //639
//     uint32_t cs299; //640
//     uint32_t ss299; //641
//     uint32_t es299; //642
//     uint32_t cs299; //643
//     uint32_t ss299; //644
//     uint32_t es299; //645
//     uint32_t cs299; //646
//     uint32_t ss299; //647
//     uint32_t es299; //648
//     uint32_t cs299; //649
//     uint32_t ss299; //650
//     uint32_t es299; //651
//     uint32_t cs299; //652
//     uint32_t ss299; //653
//     uint32_t es299; //654
//     uint32_t cs299; //655
//     uint32_t ss299; //656
//     uint32_t es299; //657
//     uint32_t cs299; //658
//     uint32_t ss299; //659
//     uint32_t es299; //660
//     uint32_t cs299; //661
//     uint32_t ss299; //662
//     uint32_t es299; //663
//     uint32_t cs299; //664
//     uint32_t ss299; //665
//     uint32_t es299; //666
//     uint32_t cs299; //667
//     uint32_t ss299; //668
//     uint32_t es299; //669
//     uint32_t cs299; //670
//     uint32_t ss299; //671
//     uint32_t es299; //672
//     uint32_t cs299; //673
//     uint32_t ss299; //674
//     uint32_t es299; //675
//     uint32_t cs299; //676
//     uint32_t ss299; //677
//     uint32_t es299; //678
//     uint32_t cs299; //679
//     uint32_t ss299; //680
//     uint32_t es299; //681
//     uint32_t cs299; //682
//     uint32_t ss299; //683
//     uint32_t es299; //684
//     uint32_t cs299; //685
//     uint32_t ss299; //686
//     uint32_t es299; //687
//     uint32_t cs299; //688
//     uint32_t ss299; //689
//     uint32_t es299; //690
//     uint32_t cs299; //691
//     uint32_t ss299; //692
//     uint32_t es299; //693
//     uint32_t cs299; //694
//     uint32_t ss299; //695
//     uint32_t es299; //696
//     uint32_t cs299; //697
//     uint32_t ss299; //698
//     uint32_t es299; //699
//     uint32_t cs299; //700
//     uint32_t ss299; //701
//     uint32_t es299; //702
//     uint32_t cs299; //703
//     uint32_t ss299; //704
//     uint32_t es299; //705
//     uint32_t cs299; //706
//     uint32_t ss299; //707
//    
```

```

10 /* Identifies an inode. */
11 #define INODE_MAGIC 0x494e4f44
12
13 /* On-disk inode.
14  | Must be exactly BLOCK_SECTOR_SIZE bytes long. */
15 struct inode_disk
16 {
17     block_sector_t start;           /* First data sector. */
18     off_t length;                 /* File size in bytes. */
19     unsigned magic;               /* Magic number. */
20     uint32_t unused[125];         /* Not used. */
21 };
22
23 /* Returns the number of sectors to allocate for an inode SIZE
24  | bytes long. */
25 static inline size_t
26 bytes_to_sectors (off_t size)
27 {
28     return DIV_ROUND_UP (size, BLOCK_SECTOR_SIZE);
29 }
30
31 /* In-memory inode. */
32 struct inode
33 {
34     struct list_elem elem;          /* Element in inode list. */
35     block_sector_t sector;          /* Sector number of disk location. */
36     int open_cnt;                  /* Number of openers. */
37     bool removed;                  /* True if deleted, false otherwise. */
38     int deny_write_cnt;            /* 0: writes ok, >0: deny writes. */
39     struct inode_disk data;        /* Inode content. */
40 };
41
42 /* Returns the block device sector that contains byte offset POS
43  | within INODE.
44  | Returns -1 if INODE does not contain data for a byte at offset
45  | POS. */
46 static block_sector_t
47 byte_to_sector (const struct inode *inode, off_t pos)
48 {
49     ASSERT (inode != NULL);
50     if (pos < inode->data.length)
51         return inode->data.start + pos / BLOCK_SECTOR_SIZE;
52     else
53         return -1;
54 }
55
56 /* List of open inodes, so that opening a single inode twice
57  | returns the same 'struct inode'. */
58 static struct list open_inodes;

```



Syscall_handler & INTR_HANDLER

```

9 void
10 syscall_init (void)
11 {
12     intr_register_int (0x30, 3, INTR_ON, syscall_handler, "syscall");
13 }

```

struct block* fs_device

```

8 /* A block device. */
9 struct block
10 {
11     struct list_elem elem;          /* Element in all_blocks. */
12     const char name[16];           /* Block device name. */
13     enum block_type type;          /* Type of block device. */
14     block_sector_t size;            /* Size in sectors. */
15
16     const struct block_operations *ops; /* Driver operations. */
17     void *aux;                      /* Extra data owned by driver. */
18
19     unsigned long long read_cnt;    /* Number of sectors read. */
20     unsigned long long write_cnt;   /* Number of sectors written. */
21 };
22

```

free_map_file

```

32
33 /* Sets up the timer to interrupt TIMER_FREQ times per second,
34  | and registers the corresponding interrupt. */
35 void
36 timer_init (void)
37 {
38     pit_configure_channel (0, 2, TIMER_FREQ);
39     intr_register_ext (0x20, timer_interrupt, "8254 Timer");
40 }

```

src/usaprog/syscall.C

- Syscall_init
- Syscall_handler

접근.

접근
접근

process.c

- process_Wait

Syscall Number 틀

Src/lib/cos/Syscall.C, syscall.h

```
22  /* Projects 2 and later. */
23  void halt (void) NO_RETURN;
24  void exit (int status) NO_RETURN;
25  pid_t exec (const char *file);
26  int wait (pid_t);
27  bool create (const char *file, unsigned initial_size);
28  bool remove (const char *file);
29  int open (const char *file);
30  int filesize (int fd);
31  int read (int fd, void *buffer, unsigned length);
32  int write (int fd, const void *buffer, unsigned length);
33  void seek (int fd, unsigned position);
34  unsigned tell (int fd);
35  void close (int fd);
```

```
100 // Timer interrupt Handler */
101 static void
102 timer_interrupt (struct intr_frame *args UNUSED)
103 {
104     /* Called by the timer interrupt handler at each timer tick.
105      * Thus, this function runs in an external interrupt context. */
106     void
107     (*thread_tick) (void);
108
109     int ticks;
110
111     /* Get the thread tick. */
112     ticks = args->ticks;
113
114     /* Update statistics. */
115     if (ticks > 0)
116         idle_ticks++;
117     else
118         user_ticks++;
119
120     #ifdef USRSPIN
121     if (!(*args->esp)) /* If no page dir. */
122         user_ticks++;
123     #endif
124
125     /* Increment kernel ticks. */
126     kernel_ticks++;
127
128     /* Perform pre-emption. */
129     if ((*thread_ticks >= TIME_SLICE)
130         _int_yield_on_return ();
131 }
132
133 /* Yield processing of an external interrupt. Directs the
134  * interrupt handler to yield to a new process just before
135  * returning from the interrupt. May not be called at any other
136  * time. */
137 void
138 _int_yield_on_return (void)
139 {
140     ASSERT (_intr_context != 0);
141     _yield_on_return = 1;
142 }
```

TIME_SLICE가
끝나면 thread.yield 한다.
이繰り返す。

```

30 // Function for interrupt handlers, timer, and exceptions. This
31 // function is called by the assembly language interrupt stubs
32 // int->handler. PWORD describes the interrupt and the
33 // interrupt number. The interrupt number is
34 // valid for timer and exceptions.
35 void (*int_handler)(struct INT_FRAME *frame)
36 {
37     struct INT_FRAME *frame;
38     int handler;
39
40     /* External interrupts are special.
41      You can handle one at a time (so interrupts must be off)
42      when you handle them. If you enable another interrupt
43      while handling an external interrupt, handler cannot sleep. */
44     if (frame->int->vector == 0x00 || frame->vec_no == 0x20)
45     {
46         /* If we're here, then we're in an interrupt
47         ASSERT (frame->level == INTL_LEVEL);
48         ASSERT (INT_FRAME::INT);
49
50         if (external)
51             yield_on_return = false;
52     }
53
54     /* If there's no handler, then we're in an interrupt
55     header = int_handlers[frame->vec_no];
56     if (header == NULL)
57     {
58         /* If frame->vec_no == 0x27 || frame->vec_no == 0x2f)
59
60         /* There is no handler, but this interrupt can trigger
61         spontaneously due to a hardware fault or hardware
62         condition. Set up a default
63         else
64     }
65
66     unexpected_interrupt (frame);
67
68     /* Complete the processing of an external interrupt. */
69     if (external)
70     {
71
72         /* If we're here, then we're in an interrupt
73         ASSERT (frame->level == INTL_LEVEL);
74         ASSERT (INT_FRAME::INT);
75
76         if (frame->vec_no == 0x20)
77             ps->end_of_interrupt (frame->vec_no);
78
79         if (yield_on_return)
80             thread_yield (i);
81     }
82 }

```

optional register #12.

@ **int_handlers** 26 번째
Callees에 대해서는 주석을 주면 좋다!

30 /* Interrupt handler functions for each interrupt,
31 static int_handler_func _INT_HANDLER (INTL_CNT)

```
17 /* Registers external interrupt VEC_NO to invoke HANDLER, which  
18 is named NAME for debugging purposes. The handler will  
19 execute with interrupts disabled. */  
20 void  
21 intr_register_ext (uint8_t vec_no, intr_handler_func handle,  
22 const char *name)  
23 {  
24     ASSERT (vec_no >= 0x20 && vec_no <= 0x2f);  
25     register_handler (vec_no, 0, INTLVL_low, handle, name);  
26 }  
27  
28 /* Registers internal interrupt VEC_NO to invoke HANDLER, which  
29 is named NAME for debugging purposes. The interrupt handler  
30 will be invoked with interrupt status LEVEL.  
31  
32 The handler will have descriptor privilege level DPL, meaning  
33 that it can be invoked immediately when the processor  
34 exits the current task. The interrupt will be triggered by the  
35 user mode to invoke the interrupt and DPL#0 prevents such  
36 invocation. Faults and exceptions that occur in user mode  
37 will be handled by the interrupt descriptor. See section  
38 [TASX-2018] sections 4.5.1 "Privilege Levels", and 4.8.5.1  
39 "Accessing Nonconforming Code Segments" for further  
40 discussion. */  
41 void  
42 intr_register_int (uint8_t vec_no, int dpl, enum intr_level_level  
43                   int_handler_func handle, const char *name)  
44 {  
45     ASSERT (vec_no >= 0x20 && vec_no <= 0x2f);  
46     register_handler (vec_no, dpl, level, handle, name);  
47 }
```

② register 된다

0x30에 Syscall_handler 힘수를 등록시켜둔다

등록 how?

- syscall-init 헤더에서
intr-register-int 험수를 호출하고
• 는 register-handler 험수를 호출한다.
• 는 intt-handlers에 데코더를 저장시킨다.

怎么了？

나중에 interrupt_handler에서 인터럽트 번호를 알고 (0x30이겠지)
syscall_handler 함수를 호출하는 것임.

아직 흐름의 시방법에 어려움이 있을 때

"구현할 것

```
15 static void
16 syscall_handler (struct intr_frame *f UNUSED)
17 {
18     printf ("system call!\n");
19     thread_exit ();
20 }
```

Page faults are an exception. They are treated the same way as memory faults, but they will not be treated as segment virtual faults.

Refer to [IA32-DM] section 4.3.3, "Execution and Interruption of User Programs" for more information about the use of page faults.

Virtual Page Faults

```

/* These exceptions can be raised by a user program,
 * e.g., via the INT3, INTN, INTL, or INTQ instructions;
 * register_error(0x0, 0x0, INT3, NULL, "INT3 Exception");
 * register_error(0x0, 0x0, INTN, NULL, "INTN Exception");
 * register_error(0x0, 0x0, INTL, NULL, "INTL Exception");
 * register_error(0x0, 0x0, INTQ, NULL, "INTQ Exception");
 */
void handleVirtualPageFault()
{
    /* These exceptions can be raised by the CPU from
     * checking them via the INTn instructions. They can still be
     * treated as virtual page faults if the CPU has the privilege
     * to do so.
     */
    register_error(0x0, 0x0, INT3, NULL, "CPU Divide Error");
    register_error(0x0, 0x0, INTN, NULL, "CPU Non-Page Fault");
    register_error(0x0, 0x0, INTL, NULL, "CPU General Protection");
    register_error(0x0, 0x0, INTQ, NULL, "CPU Stack Bound");
    register_error(0x0, 0x0, INTN, NULL, "CPU Floating-Point");
    register_error(0x0, 0x0, INTL, NULL, "CPU Non-Present");
    register_error(0x0, 0x0, INTQ, NULL, "CPU Alignment");
    register_error(0x0, 0x0, INTN, NULL, "CPU Thread");
    register_error(0x0, 0x0, INTL, NULL, "CPU General Protection");
    register_error(0x0, 0x0, INTQ, NULL, "CPU Stack Bound");
}

```

A few exceptions can be handled with interrupts instead of exceptions. The interrupt number is used to identify the exception. This is done to avoid conflicts between the interrupt numbers assigned to different exceptions.

```

15 /* Initializes the file system module.
16 | | If FORMAT is true, reformats the file system. */
17 void
18 filesys_init (bool format)
19 {
20     fs_device = block_get_role (BLOCK_FILESYSTEM);
21     if (fs_device == NULL)
22         PANIC ("No file system device found, can't initialize file system.");
23
24     inode_init ();
25     free_map_init ();
26
27     if (format)
28         do_format ();
29
30     free_map_open ();
31 }

```

```

56 /* List of open inodes, so that opening a single inode twice
57 | | returns the same 'struct inode'. */
58 static struct list open_inodes;
59
60 /* Initializes the inode module. */
61 void
62 inode_init (void)
63 {
64     list_init (&open_inodes);
65 }

```

test 2 failed msg 2% off write 2%.

```

20
21 void
22 msg (const char *format, ...)
23 {
24     va_list args;
25
26     if (quiet)
27         return;
28     va_start (args, format);
29     vmsg (format, args, "\n");
30     va_end (args);
31 }

```

```

11 static void
12 vmsg (const char *format, va_list args, const char *suffix)
13 {
14     /* No go to some trouble to stuff the entire message into a
15      * single buffer and output it in a single system call, because
16      * that'll (typically) ensure that it gets sent to the console
17      * atomically. Otherwise kernel messages like "foo: exit(0)"
18      * can end up being interleaved if we're unlucky. */
19     static char buf[1024];
20
21     sprintf (buf, strlen (buf), "(%s ", test_name);
22     vsprintf (buf + strlen (buf), strlen (buf) - strlen (buf), format, args);
23     strcpy (buf + strlen (buf), suffix);
24     write (STDOUT_FILENO, buf, strlen (buf));
25 }

```

```

281 void exit(int status)
282 {
283     struct list_elem *temp;
284     struct child_pcb *my_pcbs;
285     struct list *parent_child_pcbs_list = &thread_current()->parent->child_pcbs_list;
286     int my_tid = thread_current()->tid;
287     int exit_status = status;
288
289     for (temp = list_begin(parent_child_pcbs_list); temp != list_end(parent_child_pcbs_list); temp = list_next(temp))
290     {
291         my_pcbs = list_entry(temp, struct child_pcb, elem);
292         if (my_pcbs->tid == my_tid)
293             my_pcbs->exit_status = status;
294     }
295     thread_exit();
296 }

```

```

288 /* Deschedules the current thread and destroys it. Never
289  * returns to the caller. */
290 void
291 thread_exit (void)
292 {
293     ASSERT (!intr_context ());
294
295 #ifdef USERPROG
296     process_exit ();
297 #endif
298
299     /* Remove thread from all threads list, set our status to dying,
300      * and schedule another process. That process will destroy us
301      * when it calls thread_schedule_tail(). */
302     intr_disable ();
303     list_remove (&thread_current()->allelem);
304     thread_current ()->status = THREAD_DYING;
305     schedule ();
306     NOT_REACHED ();
307 }

```

```

156 void
157 process_exit (void)
158 {
159     struct thread scur = thread_current ();
160     uint32_t *pd;
161
162     if (cur->exit_status == -100) //unchanged, terminate on weird case
163     exit (-1);
164
165 // lab2 : remove unnecessaries
166 printf ("ws: exit(%d)\n", cur->name, cur->exit_status);
167 struct fd_elem *temp;
168 struct list_elem *e;
169 struct list_elem *to_remove;
170 struct child_pcb *tmp;
171 struct list *fd_list_to_close = &thread_current()->fd_list;
172 struct list *child_pcbs_list_to_free = &thread_current()->child_pcbs_list;
173
174 // close all connected fd
175 e = list_begin(fd_list_to_close);
176 while (e != list_end(fd_list_to_close))
177 {
178     temp = list_entry(e, struct fd_elem, elem);
179     e = list_next(e);
180     close (temp->fd_number);
181 }
182
183 // allow writes to exe_file and close it!
184 if (thread_current()->exe_file)
185 {
186     file_allow_write (thread_current()->exe_file);
187     file_close (thread_current()->exe_file);
188 }
189
190 // free child_pcbs_list
191 if (list_empty (child_pcbs_list_to_free))
192 {
193     e = list_pop_front (child_pcbs_list_to_free);
194     tmp = list_entry (e, struct child_pcb, elem);
195     free (tmp);
196 }
197
198 // check if current thread's parent was waiting for this thread's exit
199 // if (thread_current()->parent->waiting_tid == thread_current()->tid)
200 {
201     sema_up (&thread_current()->parent->is_child_finished); //yes your child is now exiting and finished
202     thread_current()->parent->waiting_tid = -1;
203 }
204
205
206 // Destroy the current process's page directory and switch back
207 // to the kernel-only page directory.
208 pd = cur->pagedir;
209 if (pd != NULL)
210 {
211     /* Correct ordering here is crucial. We must set
212      * cur->pagedir to NULL before switching page directories,
213      * so that a timer interrupt can't switch back to the
214      * process page directory. We must activate the base page
215      * directory before destroying the process's page
216      * directory, or our active page directory will be one
217      * that's been freed (and cleared). */
218     cur->pagedir = NULL;
219     pagedir_activate (pd);
220     pagedir_destroy (pd);
221 }
222 }

```

```
pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
pass tests/userprog/sc-bad-sp
pass tests/userprog/sc-bad-arg
pass tests/userprog/sc-boundary
pass tests/userprog/sc-boundary-2
pass tests/userprog/sc-boundary-3
pass tests/userprog/halt
pass tests/userprog/exit
pass tests/userprog/create-normal
pass tests/userprog/create-empty ✓
pass tests/userprog/create-null
pass tests/userprog/create-long
pass tests/userprog/create-exists
pass tests/userprog/create-bound
FAIL tests/userprog/open-normal
pass tests/userprog/open-missing
FAIL tests/userprog/open-boundary
pass tests/userprog/open-empty
pass tests/userprog/open-null
pass tests/userprog/open-bad-ptr
FAIL tests/userprog/open-twice
pass tests/userprog/close-normal
pass tests/userprog/close-twice
pass tests/userprog/close-stdin
pass tests/userprog/close-stdout
pass tests/userprog/close-bad-fd
pass tests/userprog/read-normal
FAIL tests/userprog/read-bad-ptr
FAIL tests/userprog/read-boundary
FAIL tests/userprog/read-zero
pass tests/userprog/read-stdin
pass tests/userprog/read-had-fd
FAIL tests/userprog/write-normal
FAIL tests/userprog/write-bad-ptr
FAIL tests/userprog/write-boundary
FAIL tests/userprog/write-zero
pass tests/userprog/write-stdin
pass tests/userprog/write-bad-fd
pass tests/userprog/exec-once
pass tests/userprog/exec-arg
pass tests/userprog/exec-bound
pass tests/userprog/exec-bound-2
pass tests/userprog/exec-bound-3
pass tests/userprog/exec-multiple
FAIL tests/userprog/exec-missing
pass tests/userprog/exec-bad-ptr
pass tests/userprog/wait-simple
pass tests/userprog/wait-twice
pass tests/userprog/wait-killed
pass tests/userprog/wait-bad-pid
pass tests/userprog/multi-recuse
FAIL tests/userprog/multi-child-fd
FAIL tests/userprog/rox-simple
FAIL tests/userprog/rox-child
FAIL tests/userprog/rox-multichild
pass tests/userprog/bad-read
pass tests/userprog/bad-write
pass tests/userprog/bad-read2
pass tests/userprog/bad-write2
pass tests/userprog/bad-jump
pass tests/userprog/bad-jump2
FAIL tests/userprog/no-vm/multi-oom
pass tests/filesys/base/lg-create
pass tests/filesys/base/lg-full
pass tests/filesys/base/lg-random
pass tests/filesys/base/lg-seq-block
pass tests/filesys/base/lg-seq-random
pass tests/filesys/base/sm-create
pass tests/filesys/base/sm-full
pass tests/filesys/base/sm-random
pass tests/filesys/base/sm-seq-block
pass tests/filesys/base/sm-seq-random
FAIL tests/filesys/base/sm-seq-random
FAIL tests/filesys/base/syn-read
pass tests/filesys/base/syn-remove
FAIL tests/filesys/base/syn-write
```

18 of 80 tests failed.

```
pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
pass tests/userprog/sc-bad-sp
pass tests/userprog/sc-bad-arg
pass tests/userprog/sc-boundary
pass tests/userprog/sc-boundary-2
FAIL tests/userprog/sc-boundary-3
pass tests/userprog/matt
pass tests/userprog/exit
pass tests/userprog/create-normal
pass tests/userprog/create-empty
FAIL tests/userprog/create-null
FAIL tests/userprog/create-bad-ptr
pass tests/userprog/create-long
pass tests/userprog/create-exists
pass tests/userprog/create-bound
FAIL tests/userprog/open-normal
pass tests/userprog/open-missing
FAIL tests/userprog/open-boundary
pass tests/userprog/open-empty
FAIL tests/userprog/open-null
FAIL tests/userprog/open-bad-ptr
FAIL tests/userprog/open-twice
pass tests/userprog/close-normal
pass tests/userprog/close-twice
pass tests/userprog/close-stdin
pass tests/userprog/close-stdout
pass tests/userprog/close-bad-fd
pass tests/userprog/read-normal
FAIL tests/userprog/read-bad-ptr
FAIL tests/userprog/read-boundary
FAIL tests/userprog/read-zero
pass tests/userprog/read-stdin
pass tests/userprog/read-bad-fd
FAIL tests/userprog/write-normal
FAIL tests/userprog/write-bad-ptr
FAIL tests/userprog/write-boundary
FAIL tests/userprog/write-zero
pass tests/userprog/write-stdin
pass tests/userprog/write-bad-fd
FAIL tests/userprog/exec-once
pass tests/userprog/exec-arg
pass tests/userprog/exec-bound
FAIL tests/userprog/exec-bound-2
FAIL tests/userprog/exec-bound-3
pass tests/userprog/exec-multiple
FAIL tests/userprog/exec-missing
FAIL tests/userprog/exec-bad-ptr
pass tests/userprog/wait-simple
pass tests/userprog/wait-twice
FAIL tests/userprog/wait-killed
pass tests/userprog/wait-bad-pid
pass tests/userprog/multi-recuse
FAIL tests/userprog/multi-child-fd
FAIL tests/userprog/rox-simple
FAIL tests/userprog/rox-child
FAIL tests/userprog/rox-multichild
pass tests/userprog/bad-read
pass tests/userprog/bad-write
pass tests/userprog/bad-read2
pass tests/userprog/bad-write2
pass tests/userprog/bad-jump
pass tests/userprog/bad-jump2
FAIL tests/userprog/no-vm/multi-oom
pass tests/filesys/base/lg-create
pass tests/filesys/base/lg-full
pass tests/filesys/base/lg-random
pass tests/filesys/base/lg-seq-block
pass tests/filesys/base/lg-seq-random
pass tests/filesys/base/sm-create
pass tests/filesys/base/sm-full
pass tests/filesys/base/sm-random
pass tests/filesys/base/sm-seq-block
pass tests/filesys/base/sm-seq-random
FAIL tests/filesys/base/syn-read
pass tests/filesys/base/syn-remove
FAIL tests/filesys/base/syn-write
```

25 of 80 tests failed.

D.30. 12월 26일. case 1, 2, 3
created of check_for_error

```
pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
pass tests/userprog/sc-bad-sp
pass tests/userprog/sc-bad-arg
pass tests/userprog/sc-boundary
pass tests/userprog/sc-boundary-2
pass tests/userprog/sc-boundary-3
pass tests/userprog/halt
pass tests/userprog/exit
```

```
pass tests/userprog/create-normal
pass tests/userprog/create-empty
FAIL tests/userprog/create-null
```

```
FAIL tests/userprog/create-bad-ptr
pass tests/userprog/create-long
pass tests/userprog/create-exists
pass tests/userprog/create-bound
```

```
FAIL tests/userprog/open-normal
pass tests/userprog/open-missing
FAIL tests/userprog/open-boundary
pass tests/userprog/open-empty
FAIL tests/userprog/open-null
FAIL tests/userprog/open-bad-nt
FAIL tests/userprog/open-twice
```

```
pass tests/userprog/close-normal
pass tests/userprog/close-twice
pass tests/userprog/close-stdin
pass tests/userprog/close-stdout
pass tests/userprog/close-bad-fd
```

```
FAIL tests/userprog/read-bad-ptr
FAIL tests/userprog/read-boundary
FAIL tests/userprog/read-zero
pass tests/userprog/read-stdin
pass tests/userprog/read-bad-fd
```

```
FAIL tests/userprog/write-normal
FAIL tests/userprog/write-bad-ptr
FAIL tests/userprog/write-boundary
FAIL tests/userprog/write-zero
pass tests/userprog/write-stdin
pass tests/userprog/write-bad-fd
```

```
FAIL tests/userprog/exec-once
pass tests/userprog/exec-arg
pass tests/userprog/exec-bound
pass tests/userprog/exec-bound-2
```

```
FAIL tests/userprog/exec-bound-3
pass tests/userprog/exec-multiple
FAIL tests/userprog/exec-missing
FAIL tests/userprog/exec-bad-ptr
```

```
pass tests/userprog/wait-simple
pass tests/userprog/wait-twice
pass tests/userprog/wait-killed
pass tests/userprog/wait-bad-pid
```

```
pass tests/userprog/multi-recuse
FAIL tests/userprog/multi-child-fd
FAIL tests/userprog/rox-simple
```

```
FAIL tests/userprog/rox-child
FAIL tests/userprog/rox-multichild
```

```
pass tests/userprog/bad-read
pass tests/userprog/bad-write
pass tests/userprog/bad-read2
```

```
pass tests/userprog/bad-write2
pass tests/userprog/bad-jump
pass tests/userprog/bad-jump2
```

```
FAIL tests/userprog/no-vn/multi-oom
pass tests/filesys/base/lg-create
```

```
pass tests/filesys/base/lg-full
pass tests/filesys/base/lg-random
```

```
pass tests/filesys/base/lg-seq-block
pass tests/filesys/base/lg-seq-random
```

```
pass tests/filesys/base/sm-create
pass tests/filesys/base/sm-full
```

```
pass tests/filesys/base/sm-random
pass tests/filesys/base/sm-seq-block
```

```
pass tests/filesys/base/sm-seq-random
FAIL tests/filesys/base/syn-read
```

```
pass tests/filesys/base/syn-remove
FAIL tests/filesys/base/syn-write
```

```
25 of 80 tests failed.
```

```
pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
pass tests/userprog/sc-bad-sp
pass tests/userprog/sc-bad-arg
pass tests/userprog/sc-boundary
pass tests/userprog/sc-boundary-2
pass tests/userprog/sc-boundary-3
pass tests/userprog/halt
pass tests/userprog/exit
```

```
pass tests/userprog/create-normal
pass tests/userprog/create-empty
FAIL tests/userprog/create-null
```

```
FAIL tests/userprog/create-bad-ptr
pass tests/userprog/create-long
pass tests/userprog/create-exists
pass tests/userprog/create-bound
```

```
FAIL tests/userprog/open-normal
pass tests/userprog/open-missing
FAIL tests/userprog/open-boundary
```

```
pass tests/userprog/open-null
```

```
FAIL tests/userprog/open-bad-nt
```

```
FAIL tests/userprog/open-twice
```

```
pass tests/userprog/close-normal
```

```
pass tests/userprog/close-twice
```

```
pass tests/userprog/close-stdin
```

```
pass tests/userprog/close-stdout
```

```
pass tests/userprog/close-bad-fd
```

```
FAIL tests/userprog/read-bad-ptr
```

```
FAIL tests/userprog/read-boundary
```

```
FAIL tests/userprog/read-zero
```

```
pass tests/userprog/read-stdin
```

```
pass tests/userprog/read-bad-fd
```

```
FAIL tests/userprog/write-normal
```

```
FAIL tests/userprog/write-bad-ptr
```

```
FAIL tests/userprog/write-boundary
```

```
FAIL tests/userprog/write-zero
```

```
pass tests/userprog/write-stdin
```

```
pass tests/userprog/write-bad-fd
```

```
FAIL tests/userprog/exec-once
```

```
pass tests/userprog/exec-arg
```

```
pass tests/userprog/exec-bound
```

```
pass tests/userprog/exec-bound-2
```

```
FAIL tests/userprog/exec-bound-3
```

```
pass tests/userprog/exec-multiple
```

```
FAIL tests/userprog/exec-missing
```

```
FAIL tests/userprog/exec-bad-ptr
```

```
• FAIL tests/userprog/wait-simple
```

```
• FAIL tests/userprog/wait-twice
```

```
pass tests/userprog/wait-killed
```

```
pass tests/userprog/wait-bad-pid
```

```
• FAIL tests/userprog/multi-recuse
```

```
FAIL tests/userprog/multi-child-fd
```

```
FAIL tests/userprog/rox-simple
```

```
FAIL tests/userprog/rox-child
```

```
FAIL tests/userprog/rox-multichild
```

```
pass tests/userprog/bad-read
```

```
pass tests/userprog/bad-write
```

```
pass tests/userprog/bad-read2
```

```
pass tests/userprog/bad-write2
```

```
pass tests/userprog/bad-jump
```

```
pass tests/userprog/bad-jump2
```

```
FAIL tests/userprog/no-vn/multi-oom
```

```
pass tests/filesys/base/lg-create
```

```
pass tests/filesys/base/lg-full
```

```
pass tests/filesys/base/lg-random
```

```
pass tests/filesys/base/lg-seq-block
```

```
pass tests/filesys/base/lg-seq-random
```

```
pass tests/filesys/base/sm-create
```

```
pass tests/filesys/base/sm-full
```

```
pass tests/filesys/base/sm-random
```

```
pass tests/filesys/base/sm-seq-block
```

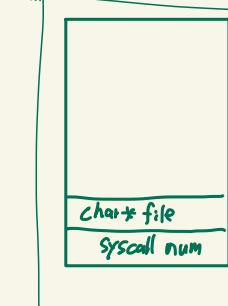
```
pass tests/filesys/base/sm-seq-random
```

```
FAIL tests/filesys/base/syn-read
```

```
pass tests/filesys/base/syn-remove
```

```
FAIL tests/filesys/base/syn-write
```

```
23 of 80 tests failed.
```



ESP에 파일 주소를 넣어야 하는데
(char*x) (esp+4)
ESP에 숫자를 넣어야 하는데
(int) (esp+4)

정상동작해야하는게 안되네.

예상...

exit code 충돌 문제 있음.

추가 check_address 넣으니 안됨.

```

pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
pass tests/userprog/sc-bad-sp
pass tests/userprog/sc-bad-arg
pass tests/userprog/sc-boundary
pass tests/userprog/sc-boundary-2
pass tests/userprog/sc-boundary-3
pass tests/userprog/halt
pass tests/userprog/exit
pass tests/userprog/create-normal
pass tests/userprog/create-empty
pass tests/userprog/create-null
pass tests/userprog/create-bad-pr
pass tests/userprog/create-long
pass tests/userprog/create-exists
pass tests/userprog/create-bound
FAIL tests/userprog/open-normal
pass tests/userprog/open-missing
FAIL tests/userprog/open-boundary
pass tests/userprog/open-empty
pass tests/userprog/open-null
pass tests/userprog/open-bad-pr
FAIL tests/userprog/open-twice
pass tests/userprog/close-normal
pass tests/userprog/close-twice
pass tests/userprog/stdin
pass tests/userprog/stdout
pass tests/userprog/close-stdout
pass tests/userprog/close-bad-fd
pass tests/userprog/read-normal
FAIL tests/userprog/read-bad-pr
FAIL tests/userprog/read-boundary
FAIL tests/userprog/read-zero
pass tests/userprog/read-stdout
pass tests/userprog/read-bad-fd
FAIL tests/userprog/write-normal
FAIL tests/userprog/write-bad-pr
FAIL tests/userprog/write-boundary
FAIL tests/userprog/write-zero
pass tests/userprog/write-stdin
pass tests/userprog/write-bad-fd
pass tests/userprog/exec-once
pass tests/userprog/exec-arg
pass tests/userprog/exec-bound
pass tests/userprog/exec-bound-2
FAIL tests/userprog/exec-bound-3
pass tests/userprog/exec-multiple
FAIL tests/userprog/exec-missing
FAIL tests/userprog/exec-hard-ntr
FAIL tests/userprog/wait-simple
FAIL tests/userprog/wait-twice
pass tests/userprog/wait-killed
pass tests/userprog/wait-bad-pid
FAIL tests/userprog/multi-recuse
FAIL tests/userprog/multi-child-fd
FAIL tests/userprog/rox-simple
FAIL tests/userprog/rox-child
FAIL tests/userprog/multichild
pass tests/userprog/bad-read
pass tests/userprog/bad-write
pass tests/userprog/bad-read2
pass tests/userprog/bad-write2
pass tests/userprog/bad-jump
pass tests/userprog/bad-jump2
FAIL tests/userprog/no-vm/multi-oom
pass tests/filesys/base/lg-create
pass tests/filesys/base/lg-full
pass tests/filesys/base/lg-random
pass tests/filesys/base/lg-seq-block
pass tests/filesys/base/lg-seq-random
pass tests/filesys/base/sm-create
pass tests/filesys/base/sm-full
pass tests/filesys/base/sm-random
pass tests/filesys/base/sm-seq-block
pass tests/filesys/base/sm-seq-random
FAIL tests/filesys/base/syn-read
pass tests/filesys/base/syn-remove
FAIL tests/filesys/base/syn-write
23 of 80 tests failed.

exec02
check02
23 → 21 → 20 → 16 → 0
21 → 20 → 16 → 0
20 → 16 → 0
16 → 0
0

```

exec02
check02
23 → 21 → 20 → 16 → 0

21 → 20 → 16 → 0

20 → 16 → 0

16 → 0

0