Changed files

1. defs.h (line 123 [last line])

```
104 //PAGEBREAK: 16
105 // proc.c
106 int             cpuid(void);
107 void            exit(void);
108 int             fork(void);
109 int             growproc(int);
110 int             kill(int);
111 struct cpu*     mycpu(void);
112 struct proc*    myproc();
113 void            pinit(void);
114 void            procdump(void);
115 void            scheduler(void) __attribute__((noreturn));
116 void            sched(void);
117 void            setproc(struct proc*);
118 void            sleep(void*, struct spinlock*);
119 void            userinit(void);
120 int             wait(void);
121 void            wakeup(void*);
122 void            yield(void);
123 int             countTraps(void);
```

(line 123)

2. syscall.h

```
23 #define SYS_countTraps 22
```

3. user.h (line 26 [last line])

```
 4 // system calls
 5 int fork(void);
 6 int exit(void) __attribute__((noreturn));
 7 int wait(void);
 8 int pipe(int*);
 9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int countTraps(void);
```

4. proc.h (lines 52-53 [last lines])

```
52   int countSyscall[22];        // Syscall counter
53   int countTraps[20];          // Trap counter
```

5. usys.S

```
32 SYSCALL(countTraps)
```

6. syscall.c

```
106 extern int sys_countTraps(void);
```

```
130 [SYS_countTraps] sys_countTraps,
```

7. sysproc.c

```
93 int
94 sys_countTraps(void)
95 {
96   return countTraps();
97 }
```

8. proc.c

```
 9 #include "syscall.h"  // include header of syscall
10 #include "traps.h"    // include header of traps.h
```

```
12 struct {
13   struct spinlock lock;
14   struct proc proc[NPROC];
15 } ptable;
16
17 /**
18 * Initializes both countSyscall and countTraps arrays
19 * Assigns each element of the arrays to be 0
20 */
21 void assignArr(struct proc *p) {
22   for(int i = 0; i < 22; i++) {
23     p->countSyscall[i] = 0;
24   }
25   for(int j = 0; j < 20; j++) {
26     p->countTraps[j] = 0;
27   }
28 }
```

(lines 21-82)

```
29
30 /**
31 * array of names of syscall commands
32 */
33 static char *syscallNameCode[] = {
34   [SYS_fork]    "SYS_fork",
35   [SYS_exit]    "SYS_exit",
36   [SYS_wait]    "SYS_wait",
37   [SYS_pipe]    "SYS_pipe",
38   [SYS_read]    "SYS_read",
39   [SYS_kill]    "SYS_kill",
40   [SYS_exec]    "SYS_exec",
41   [SYS_fstat]   "SYS_fstat",
42   [SYS_chdir]   "SYS_chdir ",
43   [SYS_dup]     "SYS_dup",
44   [SYS_getpid]  "SYS_getpid",
45   [SYS_sbrk]    "SYS_sbrk",
46   [SYS_sleep]   "SYS_sleep",
47   [SYS_uptime]  "SYS_uptime",
48   [SYS_open]    "SYS_open",
49   [SYS_write]   "SYS_write",
50   [SYS_mknod]   "SYS_mknod",
51   [SYS_unlink]  "SYS_unlink",
52   [SYS_link]    "SYS_link",
53   [SYS_mkdir]   "SYS_mkdir",
54   [SYS_close]   "SYS_close",
55   [SYS_countTraps]    "SYS_countTraps"
56   };
```

```
57
58 /**
59 * array of names of traps
60 */
61 static char *trapNameCode[] = {
62   [T_DIVIDE]   "T_DIVIDE",
63   [T_DEBUG]    "T_DEBUG",
64   [T_NMI]      "T_NMI",
65   [T_BRKPT]    "T_BRKPT",
66   [T_OFLOW]    "T_OFLOW",
67   [T_BOUND]    "T_BOUND",
68   [T_ILLOP]    "T_ILLOP",
69   [T_DEVICE]   "T_DEVICE",
70   [T_DBLFLT]   "T_DBLFLT",
71   [T_TSS-1]    "T_TSS",
72   [T_SEGNP-1]  "T_SEGNP",
73   [T_STACK-1]  "T_STACK",
74   [T_GPFLT-1]  "T_GPFLT",
75   [T_PGFLT-1]  "T_PGFLT",
76   [T_FPERR-2]  "T_FPERR",
77   [T_ALIGN-2]  "T_ALIGN",
78   [T_MCHK-2]     "T_MCHK",
79   [T_SIMDERR-2]  "T_SIMDERR",
80   [18]           "T_SYSCALL",
81   [19]           "T_DEFAULT"
82   };
```

```
179   sp -= sizeof *p->context;
180   p->context = (struct context*)sp;
181   memset(p->context, 0, sizeof *p->context);
182
183   // assigns array of p
184   assignArr(p);
185
186   p->context->eip = (uint)forkret;
187
188   return p;
189 }
190
191 //PAGEBREAK: 32
192 // Set up first user process.
193 void
194 userinit(void)
```

(Line 184; at the bottom of method ‘alloproc();)

```
325   // Parent might be sleeping in wait().
326   wakeup1(curproc->parent);
327
328   /**
329   * adds up both syscalls and trap calls
330   * from parent and child process
331   */
332   int i;
333   for(i = 0; i < 22; i++) {
334     curproc->parent->countSyscall[i] = curproc->parent->countSyscall[i] + curproc->countSyscall[i];
335     if(i < 20) {
336       curproc->parent->countTraps[i] = curproc->parent->countTraps[i] + curproc->countTraps[i];
337     }
338   }
339
340   // Pass abandoned children to init.
341   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
342     if(p->parent == curproc){
343       p->parent = initproc;
344       if(p->state == ZOMBIE)
345         wakeup1(initproc);
346     }
347   }
```

(lines 332-338; bottom of ‘exit()’ method)

```
621 /**
622 * This function is our system call. It keeps track of both system calls and traps calls.
623 * At first, it counts their occurrence from the current process.
624 * -> This is saved in countTraps[] and countSyscall[] arrays.
625 * Then, it displays them to the shell screen.
626 * After that, it shows the names of each call in both countTraps[] and countSyscall[] arrays,
627 * and the number of their occurrences in a process next to them.
628 * At last, it shows how many times the user process has been trapped to the OS in general.
629 */
630 int
631 countTraps(void) {
632   int totalSyscall = 0;
633   int totalTraps = 0;
634
635   for(int i = 0; i < 20; i++) {
636     totalTraps += myproc()->countTraps[i];
637   }
638
639   cprintf("\n*-------------------------------*\n");
640   cprintf("| Total number of traps = %d\t|\n", totalTraps);
641   cprintf("|-------------------------------|\n");
642   cprintf("| Trap Name \t| Occurrences\t|\n");
643   cprintf("|-------------------------------|\n");
644
645   for(int j = 0; j < 20; j++) {
646     if(myproc()->countTraps[j] != 0) {
647       cprintf("| [%s]\t| %d\t\t|\n", trapNameCode[j], myproc()->countTraps[j]);
648     }
649   }
650   cprintf("*-------------------------------*\n");
651
652   for(int m = 0; m < 22; m++) {
653     totalSyscall += myproc()->countSyscall[m];
654   }
655
656   cprintf("\n*---------------------------------------*\n");
657   cprintf("| Total number of system calls = %d\t|\n", totalSyscall);
658   cprintf("|---------------------------------------|\n");
659   cprintf("| Syscall Name \t\t| Occurrences\t|\n");
660   cprintf("|---------------------------------------|\n");
661
```

```
662   for(int n = 0; n < 22; n++) {
663     if(myproc()->countSyscall[n] != 0) {
664       if(n == 21) { // if the call is [SYS_countTraps] ; does a bit of formatting
665         cprintf("| [%s] \t| %d\t\t|\n", syscallNameCode[n + 1], myproc()->countSyscall[n]);
666       }
667       else {
668         cprintf("| [%s] \t\t| %d\t\t|\n", syscallNameCode[n + 1], myproc()->countSyscall[n]);
669       }
670     }
671   }
672   cprintf("*---------------------------------------*\n");
673 
674   cprintf("\n*---------------------------------------*\n");
675   if(totalTraps +totalSyscall < 10) {
676     cprintf("| Total number of all calls = %d\t\t|\n", totalTraps + totalSyscall);
677   }
678   else {
679     cprintf("| Total number of all calls = %d\t|\n", totalTraps + totalSyscall);
680   }
681   cprintf("*---------------------------------------*\n\n");
682 
683   return 22;
684 }
```

9. Trap.c

```
11 // Interrupt descriptor table (shared by all CPUs).
12 struct gatedesc idt[256];
13 extern uint vectors[];  // in vectors.S: array of 256 entry pointers
14 struct spinlock tickslock;
15 uint ticks;
16 
17 /**
18 * Checks which trap is called based on the current process's trap number.
19 * The function then increments that trap number by one each time to give us information at the end about
20 * what types of traps have occurred and the number of times they (traps) have occurred.
21 */
22 void
23 check_trap_func(struct trapframe *tf)
24 {
25   if(tf->trapno == T_DIVIDE) {
26     myproc()->countTraps[0] = myproc()->countTraps[0] + 1;
27   }
28   else if(tf->trapno == T_DEBUG) {
29     myproc()->countTraps[1] = myproc()->countTraps[1] + 1;
30   }
31   else if(tf->trapno == T_NMI) {
32     myproc()->countTraps[2] = myproc()->countTraps[2] + 1;
33   }
34   else if(tf->trapno == T_BRKPT) {
35     myproc()->countTraps[3] = myproc()->countTraps[3] + 1;
36   }
37   else if(tf->trapno == T_OFLOW) {
38     myproc()->countTraps[4] = myproc()->countTraps[4] + 1;
39   }
40   else if(tf->trapno == T_BOUND) {
41     myproc()->countTraps[5] = myproc()->countTraps[5] + 1;
42   }
43   else if(tf->trapno == T_ILLOP) {
44     myproc()->countTraps[6] = myproc()->countTraps[6] + 1;
45   }
46   else if(tf->trapno == T_DEVICE) {
47     myproc()->countTraps[7] = myproc()->countTraps[7] + 1;
48   }
49   else if(tf->trapno == T_DBLFLT) {
50     myproc()->countTraps[8] = myproc()->countTraps[8] + 1;
51   }
```

```
52    else if(tf->trapno == T_TSS) {
53      myproc()->countTraps[9] = myproc()->countTraps[9] + 1;
54    }
55    else if(tf->trapno == T_SEGNP) {
56      myproc()->countTraps[10] = myproc()->countTraps[10] + 1;
57    }
58    else if(tf->trapno == T_STACK) {
59      myproc()->countTraps[11] = myproc()->countTraps[11] + 1;
60    }
61    else if(tf->trapno == T_GPFLT) {
62      myproc()->countTraps[12] = myproc()->countTraps[12] + 1;
63    }
64    else if(tf->trapno == T_PGFLT) {
65      myproc()->countTraps[13] = myproc()->countTraps[13] + 1;
66    }
67    else if(tf->trapno == T_FPERR) {
68      myproc()->countTraps[14] = myproc()->countTraps[14] + 1;
69    }
70    else if(tf->trapno == T_ALIGN) {
71      myproc()->countTraps[15] = myproc()->countTraps[15] + 1;
72    }
73    else if(tf->trapno == T_MCHK) {
74      myproc()->countTraps[16] = myproc()->countTraps[16] + 1;
75    }
76    else if(tf->trapno == T_SIMDERR) {
77      myproc()->countTraps[17] = myproc()->countTraps[17] + 1;
78    }
79    else if(tf->trapno == T_SYSCALL) {
80      myproc()->countTraps[18] = myproc()->countTraps[18] + 1;
81    }
82    else if(tf->trapno == T_DEFAULT) {
83      myproc()->countTraps[19] = myproc()->countTraps[19] + 1;
84    }
85  }
86
87  void
88  tvinit(void)
89  {
```

(lines 22-85)

```
105 //PAGEBREAK: 41
106 void
107 trap(struct trapframe *tf)
108 {
109   if(tf->trapno == T_SYSCALL){
110     if(myproc()->killed)
111       exit();
112
113     /** Both system call and trap number are handled.
114      *  If a trap number exists, then check_trap_func()
115      *  is invoked and the new trap call increments
116      *  the occurrence of that trap by 1 in array countTraps[20].
117      *  If the eax is valid, then the system call occurrence
118      *  is incremented by 1in array countSyscall[22].
119     */
120     int num;
121     int i;
122     num = tf->eax;
123     i = num - 1;
124     myproc()->tf = tf;
125     myproc()->countSyscall[i] = myproc()->countSyscall[i] + 1;
126     check_trap_func(myproc()->tf);
127
128     syscall();
129     if(myproc()->killed)
130       exit();
131     return;
132   }
```

(line 120 - 126)

10. Makefile (lines 184-185 ; line 255)

```
166 .PRECIOUS: %.o
167
168 UPROGS=\
169         _cat\
170         _echo\
171         _forktest\
172         _grep\
173         _init\
174         _kill\
175         _ln\
176         _ls\
177         _mkdir\
178         _rm\
179         _sh\
180         _stressfs\
181         _usertests\
182         _wc\
183         _zombie\
184         _test_case1\
185         _test_case2\
186
187 fs.img: mkfs README $(UPROGS)
188         ./mkfs fs.img README $(UPROGS)
189
190 -include *.d
191
192 clean:
```

```
246 # CUT HERE
247 # prepare dist for students
248 # after running make dist, probably want to
249 # rename it to rev0 or rev1 or so on and then
250 # check in that version.
251
252 EXTRA=\
253         mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
254         ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
255         printf.c umalloc.c test_case1.c test_case2.c\
256         README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
257         .gdbinit.tmpl gdbutil\
258
259 dist:
260         rm -rf dist
261         mkdir dist
262         for i in $(FILES); \
263         do \
264                 grep -v PAGEBREAK $$i >dist/$$i; \
265         done
266         sed '/CUT HERE/,$$d' Makefile >dist/Makefile
267         echo >dist/runoff.spec
268         cp $(EXTRA) dist
```