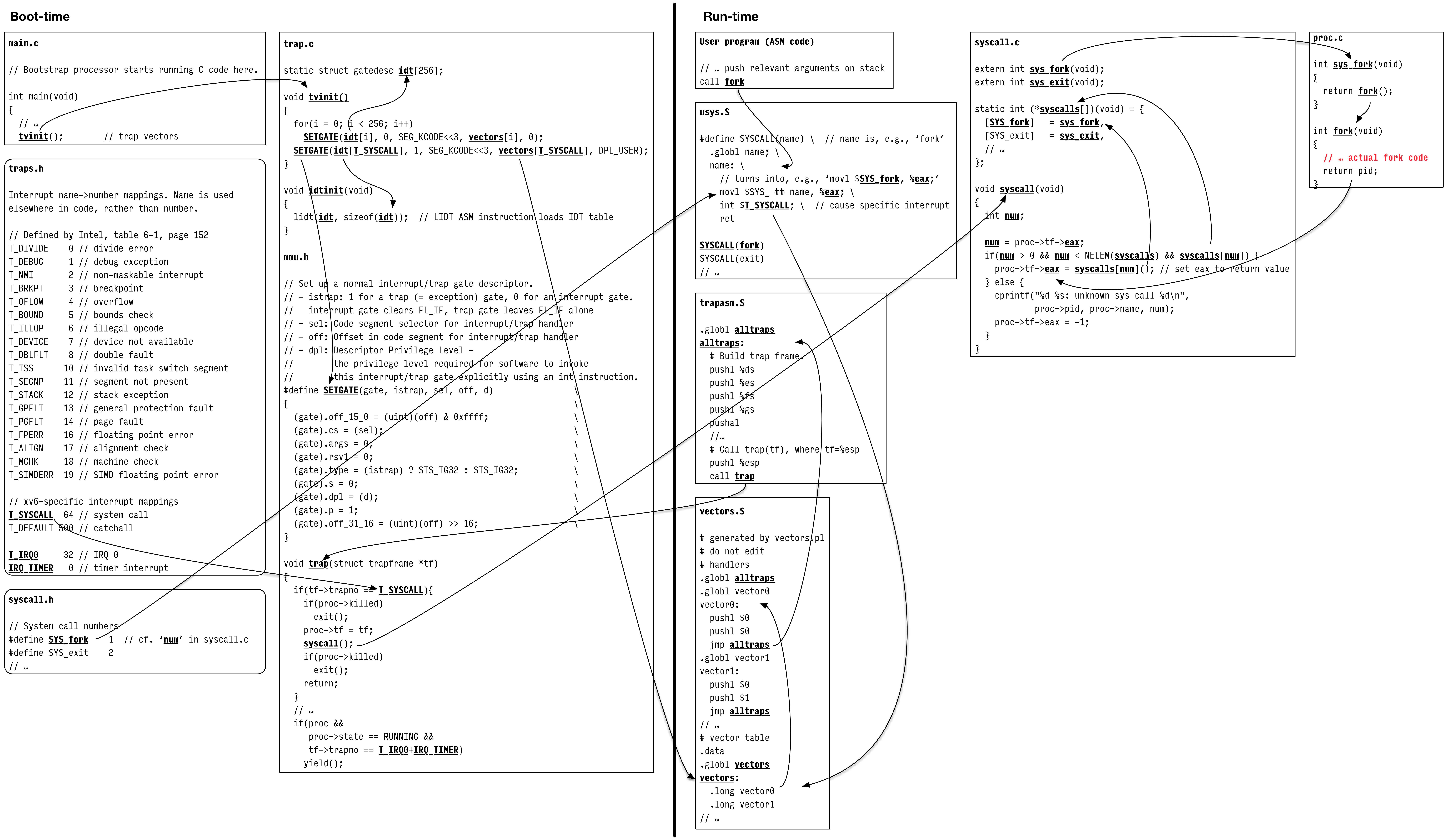

Boot-time

```
main.c

// Bootstrap processor starts running C code here.

int main(void)
{
  // …
  tvinit();        // trap vectors
```

```
traps.h

Interrupt name->number mappings. Name is used
elsewhere in code, rather than number.

// Defined by Intel, table 6-1, page 152
T_DIVIDE    0 // divide error
T_DEBUG     1 // debug exception
T_NMI       2 // non-maskable interrupt
T_BRKPT     3 // breakpoint
T_OFLOW     4 // overflow
T_BOUND     5 // bounds check
T_ILLOP     6 // illegal opcode
T_DEVICE    7 // device not available
T_DBLFLT    8 // double fault
T_TSS      10 // invalid task switch segment
T_SEGNP    11 // segment not present
T_STACK    12 // stack exception
T_GPFLT    13 // general protection fault
T_PGFLT    14 // page fault
T_FPERR    16 // floating point error
T_ALIGN    17 // alignment check
T_MCHK     18 // machine check
T_SIMDERR  19 // SIMD floating point error

// xv6-specific interrupt mappings
T_SYSCALL  64 // system call
T_DEFAULT 500 // catchall

T_IRQ0     32 // IRQ 0
IRQ_TIMER   0 // timer interrupt
```

```
syscall.h

// System call numbers
#define SYS_fork    1  // cf. 'num' in syscall.c
#define SYS_exit    2
// …
```

```
trap.c

static struct gatedesc idt[256];

void tvinit()
{
  for(i = 0; i < 256; i++)
    SETGATE(idt[i], 0, SEG_KCODE<<3, vectors[i], 0);
  SETGATE(idt[T_SYSCALL], 1, SEG_KCODE<<3, vectors[T_SYSCALL], DPL_USER);
}

void idtinit(void)
{
  lidt(idt, sizeof(idt));  // LIDT ASM instruction loads IDT table
}

mmu.h

// Set up a normal interrupt/trap gate descriptor.
// - istrap: 1 for a trap (= exception) gate, 0 for an interrupt gate.
//   interrupt gate clears FL_IF, trap gate leaves FL_IF alone
// - sel: Code segment selector for interrupt/trap handler
// - off: Offset in code segment for interrupt/trap handler
// - dpl: Descriptor Privilege Level -
//        the privilege level required for software to invoke
//        this interrupt/trap gate explicitly using an int instruction.
#define SETGATE(gate, istrap, sel, off, d)                \
{                                                         \
  (gate).off_15_0 = (uint)(off) & 0xffff;                 \
  (gate).cs = (sel);                                      \
  (gate).args = 0;                                        \
  (gate).rsv1 = 0;                                        \
  (gate).type = (istrap) ? STS_TG32 : STS_IG32;           \
  (gate).s = 0;                                           \
  (gate).dpl = (d);                                       \
  (gate).p = 1;                                           \
  (gate).off_31_16 = (uint)(off) >> 16;                   \
}

void trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(proc->killed)
      exit();
    proc->tf = tf;
    syscall();
    if(proc->killed)
      exit();
    return;
  }
  // …
  if(proc &&
     proc->state == RUNNING &&
     tf->trapno == T_IRQ0+IRQ_TIMER)
    yield();
```

Run-time

```
User program (ASM code)

// … push relevant arguments on stack
call fork
```

```
usys.S

#define SYSCALL(name) \  // name is, e.g., 'fork'
  .globl name; \
  name: \
    // turns into, e.g., 'movl $SYS_fork, %eax;'
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \  // cause specific interrupt
    ret

SYSCALL(fork)
SYSCALL(exit)
// …
```

```
trapasm.S

.globl alltraps
alltraps:
  # Build trap frame.
  pushl %ds
  pushl %es
  pushl %fs
  pushl %gs
  pushal
  //…
  # Call trap(tf), where tf=%esp
  pushl %esp
  call trap
```

```
vectors.S

# generated by vectors.pl
# do not edit
# handlers
.globl alltraps
.globl vector0
vector0:
  pushl $0
  pushl $0
  jmp alltraps
.globl vector1
vector1:
  pushl $0
  pushl $1
  jmp alltraps
// …
# vector table
.data
.globl vectors
vectors:
  .long vector0
  .long vector1
// …
```

```
syscall.c

extern int sys_fork(void);
extern int sys_exit(void);

static int (*syscalls[])(void) = {
  [SYS_fork]    = sys_fork,
  [SYS_exit]    = sys_exit,
  // …
};

void syscall(void)
{
  int num;

  num = proc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    proc->tf->eax = syscalls[num](); // set eax to return value
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            proc->pid, proc->name, num);
    proc->tf->eax = -1;
  }
}
```

```
proc.c

int sys_fork(void)
{
  return fork();
}

int fork(void)
{
  // … actual fork code
  return pid;
}
```