```
/* -*- linux-c -*- ------------------------------------------------------- *
 *
 *   Copyright (C) 1991, 1992 Linus Torvalds
 *   Copyright 2007 rPath, Inc. - All Rights Reserved
 *   Copyright 2009 Intel Corporation; author H. Peter Anvin
 *
 *   This file is part of the Linux kernel, and is made available under
 *   the terms of the GNU General Public License version 2.
 *
 * ----------------------------------------------------------------------- */

/*
 * Memory detection code
 */

#include "boot.h"

#define SMAP	0x534d4150	/* ASCII "SMAP" */

static int detect_memory_e820(void)
{
	int count = 0;
	struct biosregs ireg, oreg;
	struct e820entry *desc = boot_params.e820_map;
	static struct e820entry buf; /* static so it is zeroed */

	initregs(&ireg);
	ireg.ax  = 0xe820;
	ireg.cx  = sizeof buf;
	ireg.edx = SMAP;
	ireg.di  = (size_t)&buf;

	/*
	 * Note: at least one BIOS is known which assumes that the
	 * buffer pointed to by one e820 call is the same one as
	 * the previous call, and only changes modified fields.  Therefore,
	 * we use a temporary buffer and copy the results entry by entry.
	 *
	 * This routine deliberately does not try to account for
	 * ACPI 3+ extended attributes.  This is because there are
	 * BIOSes in the field which report zero for the valid bit for
	 * all ranges, and we don't currently make any use of the
	 * other attribute bits.  Revisit this if we see the extended
	 * attribute bits deployed in a meaningful way in the future.
	 */

	do {
		intcall(0x15, &ireg, &oreg);
		ireg.ebx = oreg.ebx; /* for next iteration... */

		/* BIOSes which terminate the chain with CF = 1 as opposed
		   to %ebx = 0 don't always report the SMAP signature on
		   the final, failing, probe. */
		if (oreg.eflags & X86_EFLAGS_CF)
			break;

		/* Some BIOSes stop returning SMAP in the middle of
		   the search loop.  We don't know exactly how the BIOS
		   screwed up the map at that point, we might have a
		   partial map, the full map, or complete garbage, so
		   just return failure. */
		if (oreg.eax != SMAP) {
			count = 0;
			break;
		}

		*desc++ = buf;
		count++;
	} while (ireg.ebx && count < ARRAY_SIZE(boot_params.e820_map));

	return boot_params.e820_entries = count;
}
```

```
74  static int detect_memory_e801(void)
75  {
76          struct biosregs ireg, oreg;
77
78          initregs(&ireg);
79          ireg.ax = 0xe801;
80          intcall(0x15, &ireg, &oreg);
81
82          if (oreg.eflags & X86_EFLAGS_CF)
83                  return -1;
84
85          /* Do we really need to do this? */
86          if (oreg.cx || oreg.dx) {
87                  oreg.ax = oreg.cx;
88                  oreg.bx = oreg.dx;
89          }
90
91          if (oreg.ax > 15*1024) {
92                  return -1;          /* Bogus! */
93          } else if (oreg.ax == 15*1024) {
94                  boot params.alt mem k = (oreg.dx << 6) + oreg.ax;
95          } else {
96                  /*
97                   * This ignores memory above 16MB if we have a memory
98                   * hole there.  If someone actually finds a machine
99                   * with a memory hole at 16MB and no support for
100                  * 0E820h they should probably generate a fake e820
101                  * map.
102                  */
103                 boot params.alt mem k = oreg.ax;
104         }
105
106         return 0;
107 }
108
109 static int detect memory 88(void)
110 {
111         struct biosregs ireg, oreg;
112
113         initregs(&ireg);
114         ireg.ah = 0x88;
115         intcall(0x15, &ireg, &oreg);
116
117         boot_params.screen_info.ext_mem_k = oreg.ax;
118
119         return -(oreg.eflags & X86_EFLAGS_CF); /* 0 or -1 */
120 }
121
122 int detect memory(void)
123 {
124         int err = -1;
125
126         if (detect_memory_e820() > 0)
127                 err = 0;
128
129         if (!detect_memory_e801())
130                 err = 0;
131
132         if (!detect_memory_88())
133                 err = 0;
134
135         return err;
136 }
```