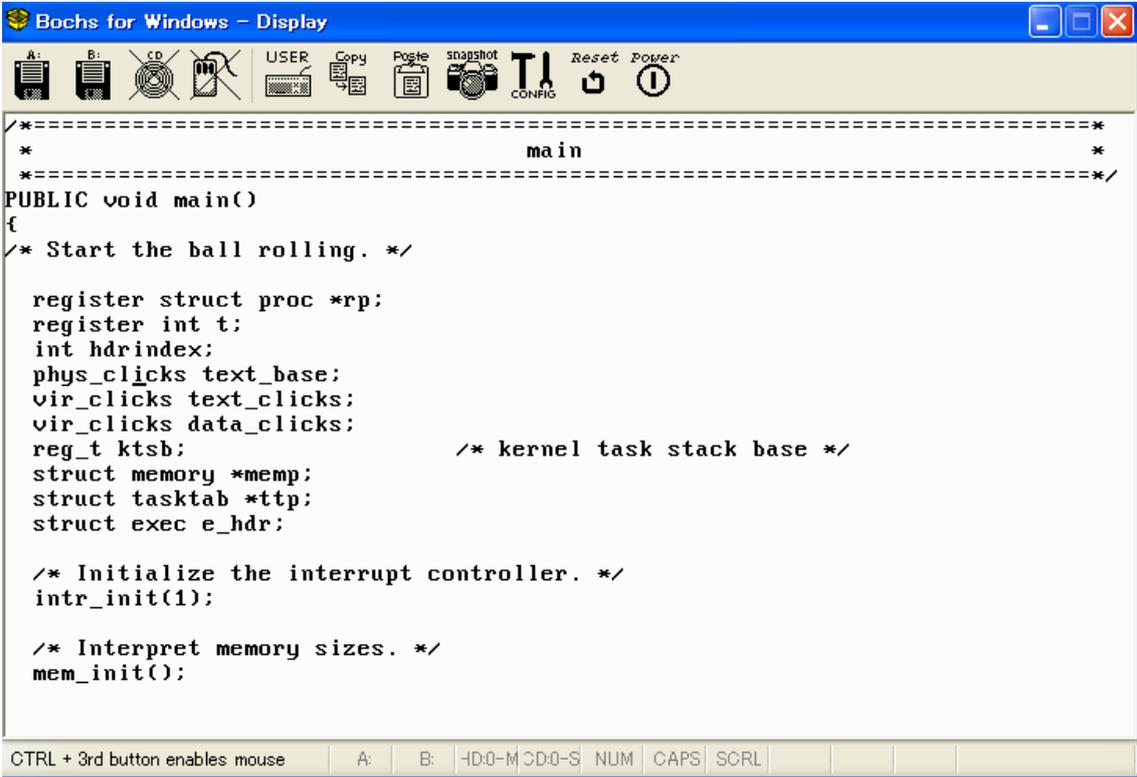


待機プロセス

以下は、`/usr/src/kernel` の `main()` プログラムの先頭部分です。



```
Bochs for Windows - Display
=====
*                                     main                                     *
=====
PUBLIC void main()
{
/* Start the ball rolling. */

register struct proc *rp;
register int t;
int hdrindex;
phys_clicks text_base;
vir_clicks text_clicks;
vir_clicks data_clicks;
reg_t ktsb;                                     /* kernel task stack base */
struct memory *memp;
struct tasktab *ttp;
struct exec e_hdr;

/* Initialize the interrupt controller. */
intr_init(1);

/* Interpret memory sizes. */
mem_init();
}
```

ここでは、まず、`struct tasktab *ttp;` で、`tasktab` 構造体へのポインタを定義しています。

「構造体へのポインタ」というのは、「構造体」というスーツケース（カバン）の番号をポイントするもの（指し示すもの）、何か、一塊のデータが格納されている「インスタンスのようなもの」の、どれかを指している、と考えてください。

この少し下に、

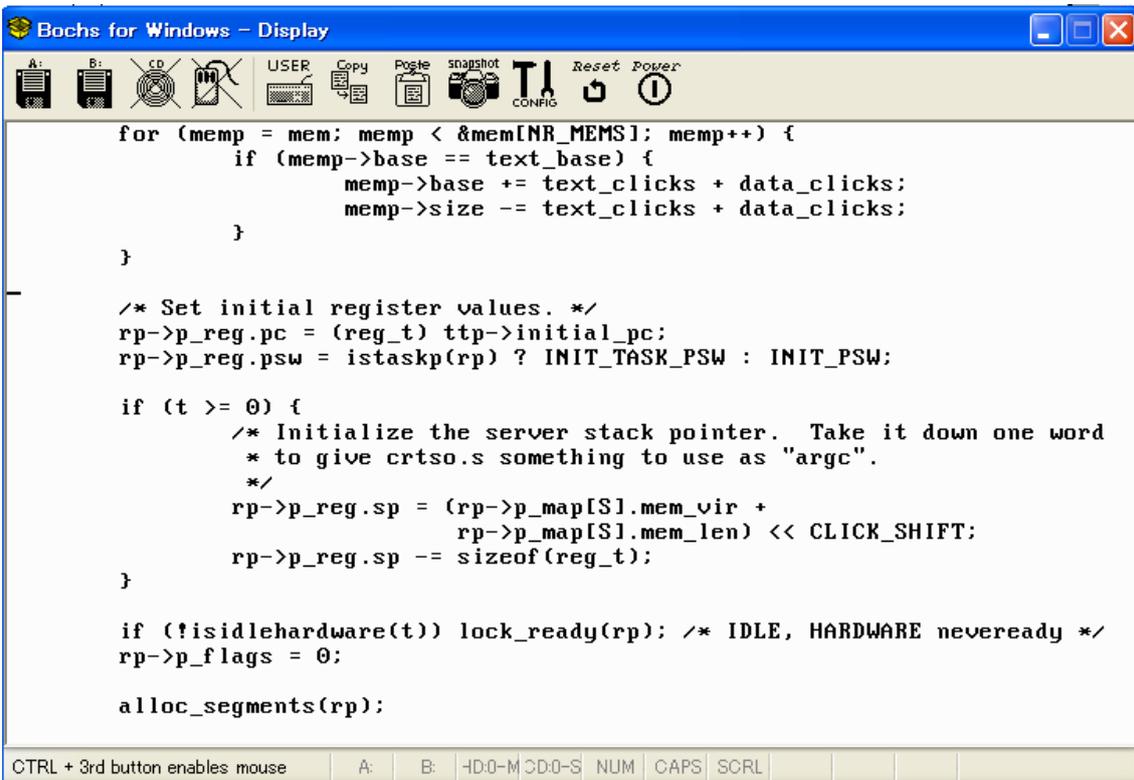
```
/* Set up proc table entries for tasks and servers. The stacks of the
 * kernel tasks are initialized to an array in data space. The stacks
 * of the servers have been added to the data segment by the monitor, so
 * the stack pointer is set to the end of the data segment. All the
 * processes are in low memory on the 8086. On the 386 only the kernel
 * is in low memory, the rest is loaded in extended memory.
 */
```

このように書かれた部分があります。このコメントの先頭に、

`Set up proc table entries for tasks and servers.`

と書かれています。

さらにこの少し下、113行目を見ると、



```
for (memp = mem; mem < &mem[NR_MEMS]; memp++) {
    if (memp->base == text_base) {
        memp->base += text_clicks + data_clicks;
        memp->size -= text_clicks + data_clicks;
    }
}

/* Set initial register values. */
rp->p_reg.pc = (reg_t) ttp->initial_pc;
rp->p_reg.psw = istaskp(rp) ? INIT_TASK_PSW : INIT_PSW;

if (t >= 0) {
    /* Initialize the server stack pointer. Take it down one word
     * to give crtso.s something to use as "argc".
     */
    rp->p_reg.sp = (rp->p_map[S].mem_vir +
                  rp->p_map[S].mem_len) << CLICK_SHIFT;
    rp->p_reg.sp -= sizeof(reg_t);
}

if (!isidlehardware(t)) lock_ready(rp); /* IDLE, HARDWARE neverready */
rp->p_flags = 0;

alloc_segments(rp);
```

rp は、proc 構造体へのポインタですが、この pc という構造体メンバに、tasktab 構造体の initial\_pc というメンバ変数の値を代入しています。

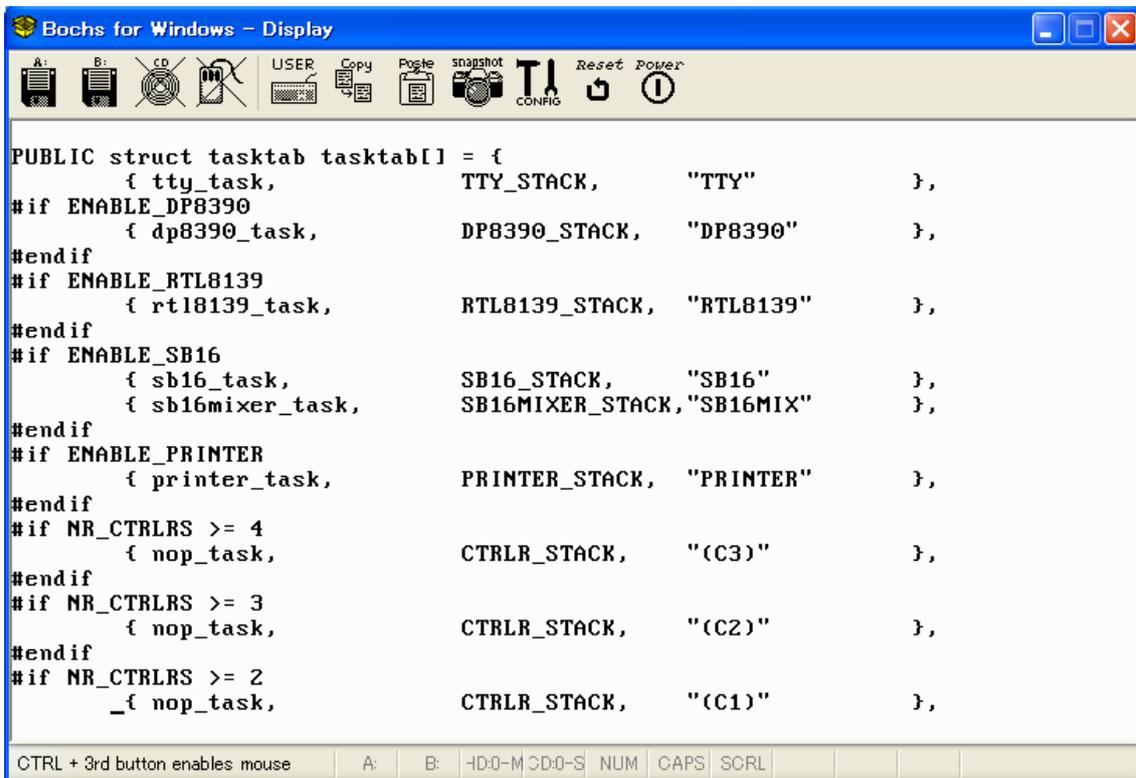
PC は、Program Counter の略で、CPU が次にどの命令を実行するか、命令語の番地を示すレジスタのことです。また、PSW は Program Status Word の略で、その命令を実行している時の CPU の状態（一つ前の計算結果で桁上がりがあったか、など）を保存している CPU のレジスタです。

rp は、プロセス群というたくさんの「カバン」を「指差す」ことのできるポインタですが、後でこの rp は一つずつが「起動」されます。これらに値を代入する、ということは、「新たにプロセスとして起動すべきプログラム」がどこにあるかを、予め登録しているということになります。

言葉を換えると、初期化処理の中で MINIX が MINIX 自身として起動していくプロセスが、ここで準備されている、ということになります。

この、tasktab の initial\_pc という変数には、どんな値が格納されているのでしょうか。

答えは `table.c` の 82 行目から書かれています。



```
Bochs for Windows - Display
A: B: CD USER Copy Paste snapshot CONFIG Reset Power
PUBLIC struct tasktab tasktab[] = {
    { tty_task,          TTY_STACK,      "TTY"          },
#if ENABLE_DP8390
    { dp8390_task,      DP8390_STACK,  "DP8390"      },
#endif
#if ENABLE_RTL8139
    { rtl8139_task,     RTL8139_STACK, "RTL8139"     },
#endif
#if ENABLE_SB16
    { sb16_task,        SB16_STACK,    "SB16"        },
    { sb16mixer_task,  SB16MIXER_STACK, "SB16MIX"    },
#endif
#if ENABLE_PRINTER
    { printer_task,    PRINTER_STACK, "PRINTER"     },
#endif
#if NR_CTRLRS >= 4
    { nop_task,        CTRLR_STACK,   "(C3)"        },
#endif
#if NR_CTRLRS >= 3
    { nop_task,        CTRLR_STACK,   "(C2)"        },
#endif
#if NR_CTRLRS >= 2
    { nop_task,        CTRLR_STACK,   "(C1)"        },
};
CTRL + 3rd button enables mouse  A: B: HD0-M DD0-S NUM CAPS SCRL
```

`type.h` を見てみます。

```
struct tasktab {
    task_t *initial_pc;
    int stksize;
    char name[8];
};
```

`tasktab` 配列の最初の引数は `initial_pc` ですが、2 番目は `stksize` であり、3 番目は `name`(名前)です。

この名前は、どこかで確認することができます。この先は、授業での説明をよく聞いてください。