COUNTER2

1. Counter. C program to generate counters for html documents. This program runs as a cgi-bin script and is called by an httpd daemon. By default the routine will initialize a counter to zero, but you can initialize it to some other value. Also an option to NOT increment the counter can be used.

This program was based on a "buggy" perl script that did the same thing. It was buggy because files on our busy server were occasionally reset to zero. I believe this was caused by a race condition between multiple copies of the program accessing the counter file simultaneously. By using POSIX record locking this problem has not re-occurred.

Modified on May 11, 1997 to do permission checking. This was do to the fact that I was getting over a million hits a week, and someone in management was notified that lots of porno sites were using it. So, to keep the service going, I had to limit access to selected sites.

2. This program is written in WEB, a preprocessor for C or Pascal. This style of programming is called "Literate Programming." For Further information see the paper *Literate Programming*, by Donald Knuth in *The Computer Journal*, Vol 27, No. 2, 1984; or the book *Weaving a Program: Literate Programming in* WEB by Wayne Sewell, Van Nostrand Reinhold, 1989. Another good source of information is the Usenet group *comp.programming.literate*. It has information on new tools and a FAQ (Frequently Asked Questions).

3. Everything is top down. Here is the first macro section that will then define all the other sections.

```
  ⟨ Program 4 ⟩
```

4. Here is the top down structure of this program.

```
⟨ Program 4 ⟩ ≡
  ⟨ Global # includes 5 ⟩
  ⟨ Global structures 6 ⟩
  ⟨ Global variables 7 ⟩
  ⟨ Utility functions 41 ⟩
  ⟨ Main 9 ⟩
```

This code is used in section 3.

5. I need these standard include files.

```
⟨ Global # includes 5 ⟩ ≡
#include <stdio.h>
#include <stdlib.h>
```

See also sections 19, 26, and 42.

This code is used in section 4.

6. Self explanatory. Standard structure for getting environment variables when in a cgi-bin script.

```
⟨ Global structures 6 ⟩ ≡
  typedef struct {
    char name[128];
    char val[128];
  } entry;
```

This code is used in section 4.

7. Prototypes. These are routines that came with NCSA's httpd software.

```
⟨ Global variables 7 ⟩ ≡
  void getword (char *word, char * line , char *stop ) ;
  char x2c(char *what);
  void unescape_url(char *url);
  void plustospace(char *str);
```

See also sections 8, 10, and 37.

This code is used in section 4.

8. Here I define all the environment varaibles that are supposed to be available to cgi-bin scripts. I also added `REFERER_URL` to check who is using the program. (This also meant that I had to slightly modify the file `cgi.c` of the NCSA httpd source to support a new environment variable.)

```
#define NUM_EVARS 17
⟨ Global variables 7 ⟩ +≡
  static char *enames[ ] = {"SERVER_SOFTWARE", "SERVER_NAME", "GATEWAY_INTERFACE",
      "SERVER_PROTOCOL", "SERVER_PORT", "REQUEST_METHOD", "HTTP_ACCEPT", "PATH_INFO",
      "PATH_TRANSLATED", "SCRIPT_NAME", "QUERY_STRING", "REMOTE_HOST", "REMOTE_ADDR",
      "REMOTE_USER", "CONTENT_TYPE", "CONTENT_LENGTH", "REFERER_URL"};
```

9. Here is the main routine. Everything is modularized using literate programming modules which are supposed to be self-documenting.

```
⟨ Main 9 ⟩ ≡
  void main(int argc, char *argv[ ]){ entry entries[100];
      register int x, m = 0;
      int i;
      char *cl;
      ⟨ Local variables 16 ⟩
      ⟨ Check access privileges of referer 11 ⟩⟨ Decode QUERY_STRING 13 ⟩
      ⟨ Convert options to uppercase 14 ⟩
      ⟨ Set options 15 ⟩
      ⟨ Open file 17 ⟩
    finis: ⟨ Generate bitmap 27 ⟩
      exit(0); }
```

This code is used in section 4.

10. List of permitted sites.

```
#define P_NUM 9
⟨ Global variables 7 ⟩ +≡
  static char *p_sites[ ] = {".mil/", ".edu/", ".org/", "www.parentinglaw.com/",
      "www.americasnet.com/", "www.mmu.ac.uk/", "home.earthlink.net/", ".bienlogic.com/",
      "www.vol.it/"};
  char *cl, *cl2, *ptr;
  int priv_found;
```

11.

```
⟨ Check access privileges of referer 11 ⟩ ≡
#if defined (DEBUG)
  printf("Checking␣access␣privileges␣of␣counter␣requester.\n");
#endif
  cl = getenv("REFERER_URL");
  if (cl ≡ Λ) {
#if defined (DEBUG)
    printf("␣Whoa␣man!!␣Your␣httpd␣does␣not␣support␣REFERER_URL.\n");
#endif
    cl = (char *) malloc(80);
    strcpy(cl, "http://white.nosc.mil/~evansjr/referer/");
  }
#if defined (DEBUG)
  printf("cl␣=␣%s.\n", cl);
#endif
  ⟨ Parse URL 12 ⟩
  priv_found = FALSE;
  for (i = 0; i < P_NUM; i++) {
#if defined (DEBUG)
    printf("Comparing␣'%s'␣with␣'%s'.\n", cl, p_sites[i]);
#endif
    if ((ptr = strstr(cl, p_sites[i])) ≠ Λ) {
      priv_found = TRUE;
      i = P_NUM;
    }
  }
  if (priv_found ≡ FALSE) {
    counter = 0;
    do_invisible = FALSE;
#if defined (DEBUG)
    printf("NO␣PRIVELEGES!\n");
#endif
    goto finis;
  }
```

This code is used in section 9.

12.

```
⟨ Parse URL 12 ⟩ ≡
#if defined (DEBUG)
  printf("Parsing␣URL␣%s.\n", cl);
#endif
  cl2 = strstr(cl, "http://");
  if (cl2 ≡ Λ) {
#if defined (DEBUG)
    printf("␣Whoa␣man!!␣No␣http␣string␣in␣REFERER_URL.\n");
#endif
  }
  ptr = (cl2 + 7);
  for (i = 0; i < 200; i++) {
    cl[i] = ptr[i];
    if (ptr[i] ≡ '/') {
      cl[i + 1] = Λ;
      i = 200;
    }
  }
```

This code is used in section 11.

13. Stolen from `query.c`. However I modified *getword*( ) because it is evidently “bad-form” to use character in html files.

```
⟨ Decode QUERY_STRING 13 ⟩ ≡
  cl = getenv("QUERY_STRING");
  if (cl ≡ Λ) {
    printf("No␣query␣information␣to␣decode.\n");
    exit(1);
  }
#if defined (DEBUG)
  printf("Your␣query␣string␣is␣%s.\n", cl);
#endif
  for (x = 0; cl[0] ≠ '\0'; x++) {
    m = x;
    getword(entries[x].val, cl, "&:;");
    plustospace(entries[x].val);
    unescape_url(entries[x].val);
    getword(entries[x].name, entries[x].val, "=");
  }
```

This code is used in section 9.

14. Make options case insensitive.

```
⟨ Convert options to uppercase 14 ⟩ ≡
  for (x = 1; x ≤ m; x++) {
    for (i = 0; entries[x].name[i] ≠ '\0'; i++) {
      c = toupper(entries[x].name[i]);
      entries[x].name[i] = (char) c;
    }
    for (i = 0; entries[x].val[i] ≠ '\0'; i++) {
      c = toupper(entries[x].val[i]);
      entries[x].val[i] = (char) c;
    }
  }
```

This code is used in section 9.

15. Here I set the options for the returned counter. At a minimum there must be the counter file name. If COUNT is set and the file does not exist, the initial count value is set to COUNT. By default INCR is assumed to be true, but if it is set to false then no auto-incrementing is done.

```
#define TRUE 1
#define FALSE 0

⟨ Set options 15 ⟩ ≡
  do_incr = TRUE;
  do_count = FALSE;
  do_reverse = FALSE;
  do_invisible = FALSE;
  for (x = 0; x ≤ m; x++) {
    if ((strcmp(entries[x].name, "COUNT")) ≡ 0) {
      counter = atoi(entries[x].val);
      do_count = TRUE;
    }
    else if ((strcmp(entries[x].name, "INCR")) ≡ 0) {
      if ((strcmp(entries[x].val, "FALSE")) ≡ 0) {
        do_incr = FALSE;
      }
    }
    else if ((strcmp(entries[x].name, "REVERSE")) ≡ 0) {
      if ((strcmp(entries[x].val, "TRUE")) ≡ 0) {
        do_reverse = TRUE;
      }
    }
    else if ((strcmp(entries[x].name, "INVISIBLE")) ≡ 0) {
      if ((strcmp(entries[x].val, "TRUE")) ≡ 0) {
        do_invisible = TRUE;
      }
    }
  }
```

This code is used in section 9.

16. Here I define local variables to handle the counter options.

```
⟨ Local variables 16 ⟩ ≡
  int do_incr, do_count, do_reverse, do_invisible;
  int c, fid;
```

See also sections 24, 29, and 34.

This code is used in section 9.

17. Opening and closing files is complicated. We need to avoid race conditions as multiple copies of this routine could be running. I use standard advisory locking (POSIX) to prevent this from happening.

```
⟨ Open file 17 ⟩ ≡
  ⟨ Stat file first 18 ⟩
  ⟨ Open appropriately 20 ⟩
  ⟨ Lock file 21 ⟩
  ⟨ Update counter prn 22 ⟩
  ⟨ Unlock and close file 23 ⟩
```

This code is used in section 9.

18. First I determine if the file even exists.

```
⟨ Stat file first 18 ⟩ ≡
  strcpy(working_dir, COUNTER_DIR);
  cfile = strcat(working_dir, entries[0].name);
  if ((stat(cfile, &buf)) < 0) {
    file_exists = FALSE;
  }
  else {
    file_exists = TRUE;
  }
```

This code is used in section 17.

19. The macro `COUNTER_DIR` is defined in `counter.h`. It is system specific so I have removed it from this code so that systems not having a `WEB` preprocessor can recompile the code.

```
⟨ Global # includes 5 ⟩ +≡
#include "counter.h"
```

20. Here I open the file.

```
⟨ Open appropriately 20 ⟩ ≡
  if (file_exists) {
    if ((fd = fopen(cfile, "r+")) ≡ Λ) {
      printf("Unable␣to␣open␣counter␣file␣%s\n", cfile);
      exit(1);
    }
  }
  else {
    if ((fd = fopen(cfile, "w")) ≡ Λ) {
      printf("Unable␣to␣open␣counter␣file␣%s\n", cfile);
      exit(1);
    }
  }
  fid = fileno(fd);
```

This code is used in section 17.

21. If I am just reading the counter I just need a shared read lock. If I am also updating the lock I need an exclusive write lock.

```
⟨ Lock file 21 ⟩ ≡
  if (do_incr) WRITE_LOCK(fid);
  else READ_LOCK(fid);
```

This code is used in section 17.

22.

```
⟨ Update counter prn 22 ⟩ ≡
  if ((¬file_exists) ∧ (do_count)) {
    fprintf(fd, "%d\n", counter);
  }
  else {
    fscanf(fd, "%d", &counter);
    if (do_incr) {
      counter++;
      fseek(fd, SEEK_SET, 0);
      fprintf(fd, "%d\n", counter);
    }
  }
```

This code is used in section 17.

23.

```
⟨ Unlock and close file 23 ⟩ ≡
  UN_LOCK(fid);
  fclose(fd);
```

This code is used in section 17.

24.

```
⟨ Local variables 16 ⟩ +≡
  int file_exists;
  struct stat buf;
  FILE *fd;
  int counter = 0;
  char *cfile, working_dir[80];
```

25. Bitmap Generation.

26. These are needed in the following code.

```
⟨ Global # includes 5 ⟩ +≡
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <ctype.h>
#include <string.h>
#include <math.h>
```

27. Here I generate the bitmap. The macro MINLEN is the minimum number of characters to generate. If the number requires more than `MINLEN` characters, then it generates them.

```
#define MINLEN 6

⟨ Generate bitmap 27 ⟩ ≡
  ⟨ Order digits 28 ⟩
  if (¬do_invisible) {
    ⟨ Write digits 30 ⟩
  }
  else {
    ⟨ Write out null image 32 ⟩
  }
```

This code is used in section 9.

28. Here I order the digits.

```
⟨ Order digits 28 ⟩ ≡
  if (counter ≡ 0) {
    x = 1;
  }
  else {
    x = ((int) floor(log10(counter))) + 1;
  }
  i = 0;
  if (x < MINLEN) {
    for (i = 0; i < (MINLEN − x); i++) {
      digits[i] = 0;
    }
    numdigits = MINLEN;
  }
  else {
    numdigits = x;
  }
  newc = counter;
  while (newc > 0) {
    x−−;
    if (x > 0) powr = ((int) pow(10.0, x));
    else powr = 1;
    digits[i] = newc/powr;
    newc = newc − (digits[i++] ∗ powr);
  }
```

This code is used in section 27.

29.

```
⟨ Local variables 16 ⟩ +≡
  int newc, powr, numdigits, digits[10];
  int j, kntr, numbytes, ind;
```

30.

```
⟨ Write digits 30 ⟩ ≡
  ⟨ Write x-bitmap header 31 ⟩
  ⟨ Write out first 3 lines of filler 33 ⟩
  ⟨ Write out 10 lines of digits 35 ⟩
  ⟨ Write out last 3 lines of filler 36 ⟩
```

This code is used in section 27.

31. Standard ascii header for x-bitmap.

```
⟨ Write x-bitmap header 31 ⟩ ≡
  printf("Content-type:␣image/x-xbitmap\n\n");
  printf("#define␣count_width␣%d\n#define␣count_height␣16\n", numdigits ∗ 8);
  printf("static␣char␣count_bits[]␣=␣{\n");
```

This code is used in section 30.

32. Standard ascii header for x-bitmap.

```
⟨ Write out null image 32 ⟩ ≡
  printf("Content-type:␣image/x-xbitmap\n\n");
  printf("#define␣count_width␣1\n#define␣count_height␣1\n");
  printf("static␣char␣count_bits[]␣=␣{\n");
  if (do_reverse) printf("0x00};\n");
  else printf("0xff};\n");
```

This code is used in section 27.

33.

```
⟨ Write out first 3 lines of filler 33 ⟩ ≡
  kntr = 0;
  if (¬do_reverse) filler = "0xff";
  else filler = "0x00";
  numbytes = 16 ∗ 8 ∗ numdigits;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < numdigits; j++) {
      printf("%s,", filler);
      kntr++;
      if ((kntr % 8) ≡ 0) printf("\n");
    }
  }
```

This code is used in section 30.

34.

```
⟨ Local variables 16 ⟩ +≡
  char ∗filler;
```

35.

```
⟨ Write out 10 lines of digits 35 ⟩ ≡
  if (do_reverse) {
    for (i = 0; i < 10; i++) {
      for (j = 0; j < numdigits; j++) {
        ind = digits[j] * 10 + i;
        printf("%s,", ibitstream[ind]);
        kntr++;
        if ((kntr % 8) ≡ 0) printf("\n");
      }
    }
  }
  else {
    for (i = 0; i < 10; i++) {
      for (j = 0; j < numdigits; j++) {
        ind = digits[j] * 10 + i;
        printf("%s,", bitstream[ind]);
        kntr++;
        if ((kntr % 8) ≡ 0) printf("\n");
      }
    }
  }
```

This code is used in section 30.

36.

```
⟨ Write out last 3 lines of filler 36 ⟩ ≡
  for (j = 0; j < (3 * numdigits − 1); j++) {
    printf("%s,", filler);
    kntr++;
    if ((kntr % 8) ≡ 0) printf("\n");
  }
  if (do_reverse) printf("0x00};\n");
  else printf("0xff};\n");
```

This code is used in section 30.

37.

```
⟨ Global variables 7 ⟩ +≡
  static char *bitstream[] = {"0xc3","0x99","0x99","0x99","0x99","0x99","0x99","0x99","0x99",
      "0xc3","0xcf","0xc7","0xcf","0xcf","0xcf","0xcf","0xcf","0xcf","0xcf","0xc7","0xc3",
      "0x99","0x9f","0x9f","0xcf","0xe7","0xf3","0xf9","0xf9","0x81","0xc3","0x99","0x9f",
      "0x9f","0xc7","0x9f","0x9f","0x9f","0x99","0xc3","0xcf","0xcf","0xc7","0xc7","0xcb",
      "0xcb","0xcd","0x81","0xcf","0x87","0x81","0xf9","0xf9","0xf9","0xc1","0x9f","0x9f",
      "0x9f","0x99","0xc3","0xc7","0xf3","0xf9","0xf9","0xc1","0x99","0x99","0x99","0x99",
      "0xc3","0x81","0x99","0x9f","0x9f","0xcf","0xcf","0xe7","0xe7","0xf3","0xf3","0xc3",
      "0x99","0x99","0x99","0xc3","0x99","0x99","0x99","0x99","0xc3","0xc3","0x99","0x99",
      "0x99","0x99","0x83","0x9f","0x9f","0xcf","0xe3"};
  static char *ibitstream[] = {"0x3c","0x66","0x66","0x66","0x66","0x66","0x66","0x66","0x66",
      "0x3c","0x30","0x38","0x30","0x30","0x30","0x30","0x30","0x30","0x30","0x38","0x3c",
      "0x66","0x60","0x60","0x30","0x18","0x0c","0x06","0x06","0x7e","0x3c","0x66","0x60",
      "0x60","0x38","0x60","0x60","0x60","0x66","0x3c","0x30","0x30","0x38","0x38","0x34",
      "0x34","0x32","0x7e","0x30","0x78","0x7e","0x06","0x06","0x06","0x3e","0x60","0x60",
      "0x60","0x66","0x3c","0x38","0x0c","0x06","0x06","0x3e","0x66","0x66","0x66","0x66",
      "0x3c","0x7e","0x66","0x60","0x60","0x30","0x30","0x18","0x18","0x0c","0x0c","0x3c",
      "0x66","0x66","0x66","0x3c","0x66","0x66","0x66","0x66","0x3c","0x3c","0x66","0x66",
      "0x66","0x66","0x7c","0x60","0x60","0x30","0x1c"};
```

38. Record Locking.

39. Here are some macro definitions taken from W. R. Stevens' wonderful book, *Advanced Programming in the Unix Environment*. They provide a clean interface to the `fcntl` function, which is the POSIX.1 method of locking records.

```
#define read_lock(fd, offset, whence, len)  lock_reg(fd, F_SETLK, F_RDLCK, offset, whence, len)
#define readw_lock(fd, offset, whence, len)  lock_reg(fd, F_SETLKW, F_RDLCK, offset, whence, len)
#define write_lock(fd, offset, whence, len)  lock_reg(fd, F_SETLK, F_WRLCK, offset, whence, len)
#define writew_lock(fd, offset, whence, len)  lock_reg(fd, F_SETLKW, F_WRLCK, offset, whence, len)
#define un_lock(fd, offset, whence, len)  lock_reg(fd, F_SETLK, F_UNLCK, offset, whence, len)
```

40. Here are some macros of my own devising. Since I am always locking the entire file, I need only one parameter. The other parameters never change. I use W. R. Stevens' macros defined previously.

```
#define READ_LOCK(fd)  readw_lock(fd, 0, SEEK_SET, 0)
#define WRITE_LOCK(fd)  writew_lock(fd, 0, SEEK_SET, 0)
#define UN_LOCK(fd)  un_lock(fd, 0, SEEK_SET, 0)
```

41. Here is W. R. Stevens' `lock_reg` function which provides a nice interface to the `fcntl` function.

```
⟨ Utility functions 41 ⟩ ≡
  int lock_reg(int fd, int cmd, int type, off_t offset, int whence, off_t len)
  {
    struct flock lock;

    lock.l_type = type;    /* F_RDLCK, F_WRLCK, F_UNLCK */
    lock.l_start = offset;    /* byte offset, relative to l_whence */
    lock.l_whence = whence;    /* SEEK_SET, SEEK_CUR, SEEK_END */
    lock.l_len = len;    /* #bytes (0 means to EOF) */
    return (fcntl(fd, cmd, &lock));
  }
```

See also section 44.

This code is used in section 4.

42. The types and constants used for record locking are defined in `fcntl.h`.

```
⟨ Global # includes 5 ⟩ +≡
#include <fcntl.h>
```

43. Utility functions.

44. This is a function taken from `util.c` which came with the NCSA Httpd distribution. I have modified to allow more then one stop character.

```
⟨ Utility functions 41 ⟩ +≡
  void getword (char *word, char * line , char *stop ) { int x = 0, y, kntr;
  char *ptr; ptr = strpbrk ( line , stop ) ; if (ptr) { kntr = ptr − line ; } else { kntr = strlen ( line ) ;
        } for (x = 0; x < kntr; x++) { word[x] =
  line [x] ;
  } word[kntr] = '\0'; if ( line [x] ) ++x;
  y = 0; while ( line [y++] = line [x++] ) ; }
```

45. Index. Here is a list of the identifiers used, and where they appear. Underlined entries indicate the place of definition.

⟨ Check access privileges of referer 11 ⟩ Used in section 9.

⟨ Convert options to uppercase 14 ⟩ Used in section 9.

⟨ Decode `QUERY_STRING` 13 ⟩ Used in section 9.

⟨ Generate bitmap 27 ⟩ Used in section 9.

⟨ Global structures 6 ⟩ Used in section 4.

⟨ Global variables 7, 8, 10, 37 ⟩ Used in section 4.

⟨ Global # **include**s 5, 19, 26, 42 ⟩ Used in section 4.

⟨ Local variables 16, 24, 29, 34 ⟩ Used in section 9.

⟨ Lock file 21 ⟩ Used in section 17.

⟨ Main 9 ⟩ Used in section 4.

⟨ Open appropriately 20 ⟩ Used in section 17.

⟨ Open file 17 ⟩ Used in section 9.

⟨ Order digits 28 ⟩ Used in section 27.

⟨ Parse URL 12 ⟩ Used in section 11.

⟨ Program 4 ⟩ Used in section 3.

⟨ Set options 15 ⟩ Used in section 9.

⟨ Stat file first 18 ⟩ Used in section 17.

⟨ Unlock and close file 23 ⟩ Used in section 17.

⟨ Update counter prn 22 ⟩ Used in section 17.

⟨ Utility functions 41, 44 ⟩ Used in section 4.

⟨ Write digits 30 ⟩ Used in section 27.

⟨ Write out 10 lines of digits 35 ⟩ Used in section 30.

⟨ Write out first 3 lines of filler 33 ⟩ Used in section 30.

⟨ Write out last 3 lines of filler 36 ⟩ Used in section 30.

⟨ Write out null image 32 ⟩ Used in section 27.

⟨ Write x-bitmap header 31 ⟩ Used in section 30.