Accessing a FAT File System from T-Engine

Introduction

There are various types of logical formats for disks and memory cards. On MS-DOS and Windows (excluding the NT line), they have adopted a logical format called FAT **[1]**. As for CD-ROM logical formats, ISO9660 is basic. Also, in the T-Engine development kits, they have adopted a logical format based on the BTRON specification, and they have realized a high level file system that possess a multi-link, hypertext structure. However, because FAT has also been widely adopted in various types of memory cards for embedded devices, such as digital cameras, there have been a lot of requests for FAT even with T-Engine, which is used for embedded device development. Accordingly, FAT (FAT12, FAT16, FAT32) and ISO9660 (Level 1) have recently been added to T-Engine development kits **[2]**. In this paper, we will explain these functions.

Method of Use

We carry out FAT file access in the following sequence.

(1) Load the UNIX (File) Emulator

The FAT file access function is realized by means of a subsystem called the T-Kernel Extension "UNIX (File) Emulator" **[3]**. Because this will be loaded at startup time if
lodspg unixemu
is specified inside the STARTUP.CMD initial startup file, there is no need to be particularly conscious of it

(2) File System Connection

- Preceding file access, it is necessary to call up attach() from inside the program and connect the file system.
  attach(device name, connection name, mode)
  - For the device name, if it's (the first partition) of a memory card of a PC Card slot, we specify "pca0"; if it's a USB connection,we specify "uda0" with a TC character string.
- For the connection name, we specify with an ASCII or Japanese EUC character string. As described below, it is used as the beginning of the bus name.
- For the mode, in a case where the logical format is FAT, we specify UX_MSDOS. Outside of this, we specify UX_BTRON in the case of the BTRON file system, and UX_CDROM in the case of a CD-ROM (ISO9660).

(3) File Access

As for file access APIs, APIs based on UNIX system calls, APIs based on ANSI C standard libraries, and POSIX-based directory read-in APIs can respectively be used (Table 1).
The path name that points to a file is the format "/connection name/directory name . . . /file name," and we specify this with an ASCII or Japanese language EUC character string. The punctuation character is '/', not '¥'. For example, the path name of the file cat.jpg, which is in the IMAGE directory under the directory DIR1 of the memory card connected with the connection name A becomes "/A/DIR1/IMAGE/cat.jpg."

(4) Cutting Off the File System

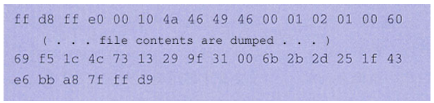
- When file access is completed, please cut off the file system by calling up detach() from inside the program.
  detach(device name, 0 or 1)
  - The device name is the same as that specified at the time of attach().
- When 1 is specified in the second argument, if possible, discharge the media.

Table 1 File Access APIs

UNIX-based API	ANSI C/POSIX-based API	Function
open	fopen, freopen, tmpfile	Opens a file
close	fclose	Closes a file
read	fread, getc, fgetc, fgets, fscanf, vfscanf	Reads in a file
write	fwrite, putc, fputc, fputs, fprintf, vfprintf	Writes in a file
lseek	fseek, fsetpos, rewind	Moves a file pointer
rename	rename	Changes a file name
unlink	remove	Deletes a file
	feof, ftell, fgetpos, ferror, clearerr	Detects each type of status
	fflush, setbuf, setvbuf, ungetc	Controls the buffer
getdents	opendir, readdir,closedir, rewinddir	Reads in a directory
mkdir, rmdir		Creates/deletes a directory
stat, lstat, fstat		Obtains file information

Sample Programs

We will now give three simple process base sample programs.



```
ff d8 ff e0 00 10 4a 46 49 46 00 01 02 01 00 60
    ( . . . file contents are dumped . . . )
69 f5 1c 4c 73 13 29 9f 31 00 6b 2b 2d 25 1f 43
e6 bb a8 7f ff d9
```

Figure 1. Example of the execution results of List 1

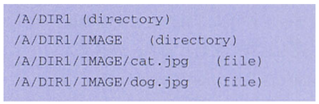
- List 1: Read in a File

  We connect the USB disk uda0 with connection name A, read in the file "/DIR1/IMAGE/cat.jpg" with fopen and fread, and do a hexadecimal dump (Fig. 1).

List 1 Read in a File

```
/*
  TRONWARE Vol. 84 Recording
  T-Engine FAT File Access Sample Programs
  Sample 1: Read in a File
  Copyright (C) 2003 Personal Media Corporation
*/

#include <basic.h>
#include <stdio.h>
#include <btron/unixemu.h>

W main( W ac, TC *av[] )
{
    int r, i; FILE *f; UB buf[16];
    const TC devnm[] = {L"uda0"};         /* Device name (TC character string) */
#define CONNM "A"                         /* Connection name */

    r = attach( devnm, CONNM, UX_MSDOS ); /* Connection */
    if (r < 0) goto e1;

    f = fopen( "/" CONNM "/DIR1/IMAGE/cat.jpg", "r" );
    if (f == NULL) goto e2;
    while( (r = fread( buf, sizeof(UB), sizeof(buf), f )) > 0 ) {
        for( i = 0; i < r; i++ ) printf( "%02x ", buf[i] );
        printf( "\n" );
    }
    fclose( f );

e2: detach( devnm, 0 );                  /* Cut-off */
e1: return 0;
}
```

Translator's note: The characters "uda0" highlighted in red above are simulated TRON Code characters; they are not actually written with TRON Code. The same applies to the lists below. Please keep this in mind if you plan to cut and paste the source code in these lists.



```
/A/DIR1 (directory)
/A/DIR1/IMAGE   (directory)
/A/DIR1/IMAGE/cat.jpg   (file)
/A/DIR1/IMAGE/dog.jpg   (file)
```

Figure 2. Example of the execution results of List 2

- List 2: Recursively Search the Directories

  We recursively search the directories on the disk by means of opendir and readdir and output a path name (Fig 2).

List 2: Recursively Search the Directories

```
/*
  TRONWARE Vol. 84 Recording
  T-Engine FAT File Access Sample Programs
  Sample 2: Recursively Search the Directory
  Copyright (C) 2003 Personal Media Corporation
*/

#include <basic.h>
#include <stdio.h>
#include <string.h>
#include <btron/unixemu.h>
#include <unix/dirent.h>

static void list( char *path )
{
    DIR *f; struct dirent *d; char *t;

    for( t = path; *t != '\0'; t++ );    /* Find the end of the path name */
    f = opendir( path );
    if (f == NULL) return;
    while( (d = readdir( f )) != NULL ) {
        *t = '/';
        strcpy( t + 1, d -> d_name );
        if (d -> d_type == DT_DIR) {     /* In case of a directory */
            if (strcmp( d -> d_name, "." ) == 0
             || strcmp( d -> d_name, ".." ) == 0) {
                                         /* Read over . and .. */
            } else {
                printf( "%s\t(directory)\n", path );
                list( path );            /* Recursively follows the directory */
            }
        } else {                         /* In the case of a file */
            printf( "%s\t(file)\n", path );
        }
    }
    closedir( f );
    *t = '\0';
}

W main( W ac, TC *av[] )
{
    int r; char buf[512];
    const TC devnm[] = {L"uda0"};        /* Device name (TC character string) */
#define CONNM "A"                        /* Connection name */

    r = attach( devnm, CONNM, UX_MSDOS ); /* Connection */
    if (r < 0) goto e1;

    strcpy( buf, "/" CONNM );
    list( buf );

    detach( devnm, 0 );                  /* Cut-off */
e1: return 0;
}
```

- List 3: Display a JPEG Image

  We read in the JPEG file "/DIR1/IMAGE/cat.jpg" with open and read, convert it into a 65,536 color bitmap using the image format conversion library of T-Shell, and display it on the screen by means of the T-Shell display primitives (T-Shell required).

List 3: Display a JPEG Image

```
/*
  TRONWARE Vol. 84 Recording
  T-Engine FAT File Access Sample Programs
  Sample 3: Display a JPEG Image
  Copyright (C) 2003 Personal Media Corporation
*/

#include <basic.h>
#include <stdlib.h>
#include <btron/unixemu.h>
#include <unix/fcntl.h>
#include <unix/unistd.h>
#include <btron/dp.h>
#include <btron/libimg.h>

static int file_read( UB *buf, UW reqsize, int *fd )
{
    return read( *fd, buf, reqsize );
}

W main( W ac, TC *av[] )
{
    int r, fd; BMP bmp; IMG_BMP ib; GID gid; RECT rect;
    IMG_COMPACT comp = { LIBIMG_METHOD_JPEG, LIBIMG_JPEG_READ_OPT_1_1_SIZE };
    CSPEC cspec = { DA_COLOR_RGB, {0x0b05, 0x0506, 0x0005, 0}, NULL };
    const TC scr[] = {L"SCREEN"};
    const TC devnm[] = {L"uda0"};        /* Device name (TC character string) */
#define CONNM "A"                        /* Connection name */

    r = attach( devnm, CONNM, UX_MSDOS ); /* Connection */
    if (r < 0) goto e1;

    /* read in the JPEG file and convert it into a bitmap */
    fd = open( "/" CONNM "/DIR1/IMAGE/cat.jpg", O_RDONLY );
    if (fd < 0) goto e2;
    bmp.pixbits = 0x1010;                /* 16 bit = 65536 colors */
    ib.color_spec = &cspec; ib.bmap = &bmp;
    ib.funcptr = file_read; ib.flushptr = NULL; ib.io_src = &fd;
    libimg_rea_bmp( &ib, &comp );
    close( fd );

    /* Display bitmap on the screen */
    gid = b_gopn_dev( scr, NULL );
    b_gset_bcs( gid, &cspec );
    b_gget_fra( gid, &rect );
    b_grst_bmp( gid, &rect, &bmp, &bmp.bounds, NULL,
                G_STORE | G_CVFORM | G_CVCOLOR );
    b_gcls_env( gid );
    free( bmp.baseaddr[0] );

e2: detach( devnm, 0 );                  /* Cut-off */
e1: return 0;
}
```

Application Example: JPEG Viewer

As an Application Example, we have created a JPEG image viewer (the programs lists are recorded onto the attached CD-ROM).

- **Program Outline**

- This program automatically detects the memory being inserted and commences read-in. For insertion detection of the device, we use the T-Kernel Extension Event Manager.
- In the application in List 2, we recursively search the directories on the memory card and read in all the JPEG files.
- In the application in List 3, we convert the read-in JPEG files into bitmaps and retain them.
- For touch panel and button input also, we use the T-Kernel Extension Event Manager.
- In a case where the LCD screen is vertically arranged and the images are horizontally arranged, we display by rotating 90 degrees using the b_grot_bmp API of T-Shell.

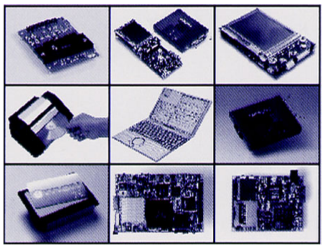


Figure 3. List mode of the JPEG image viewer

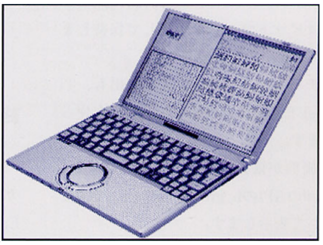


Figure 4. Basic mode of the JPEG image viewer

- **Operation**
- There are two modes. In the list mode, we display by reducing the images into 3 x 3 =9 (Fig. 3). In the basic mode, we display one image (Fig. 4).
- By means of the "←" and "→" buttons, we move to the next page or preceding page.
- When we touch the image in the list mode, that image expands and we view in the basic mode.
- When we push the "O" button, we move to the list mode.

Application Example: JPEG Viewer

```
/*
  TRONWARE Vol. 84 Recording
  T-Engine FAT File Access Sample Programs
  Sample 4: JPEG Viewer
  Copyright (C) 2003 Personal Media Corporation
*/

#include <basic.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <queue.h>
#include <btron/event.h>
#include <btron/proctask.h>
#include <btron/unixemu.h>
#include <unix/dirent.h>
#include <unix/fcntl.h>
#include <unix/unistd.h>
#include <btron/dp.h>
#include <btron/libimg.h>

#define Debug(s) printf s ;

static const TC devnm[] = {L"uda0"}; /* Device name (TC character string) */
#define CONNM "A" /* Connection name */
static CSPEC CSpec = { DA_COLOR_RGB, {0x0b05, 0x0506, 0x0005, 0}, NULL };
static const int PixBits = 0x1010;
static int ScreenWidth, ScreenHeight;
static int WorkWidth, WorkHeight;
static GID gid0, wgid[2];
static BMP wbmp[2];

#define GapX 2
#define GapY GapX
#define FullSize( rect ) rect.c.left = 0, rect.c.right = ScreenWidth, rect.c.top = 0, rect.c.bottom = ScreenHeight
#define Piece( rect, i, k, mcol, mrow, GapX, GapY ) rect.c.left = ScreenWidth / mcol * (mcol - (i) - 1), rect.c.right = ScreenWidth / mcol * (mcol - (i)) - GapX, rect.c.top = ScreenHeight / mrow * (k), rect.c.bottom = ScreenHeight / mrow * ((k) + 1) - GapY
#define RightLine( rect, i, k, mcol, mrow, GapX, GapY ) rect.c.left = (rect.c.right = ScreenWidth / mcol * (mcol - (i))) - GapX, rect.c.top = ScreenHeight / mrow * (k), rect.c.bottom = ScreenHeight / mrow * ((k) + 1)
#define BottomLine( rect, i, k, mcol, mrow, GapX, GapY ) rect.c.left = ScreenWidth / mcol * (mcol - (i) - 1), rect.c.right = ScreenWidth / mcol * (mcol - (i)), rect.c.top = (rect.c.bottom = ScreenHeight / mrow * ((k) + 1)) - GapY
#define RightMargin( rect, mcol, mrow ) rect.c.left = ScreenWidth / mcol * mcol, rect.c.right = ScreenWidth, rect.c.top = 0, rect.c.bottom = ScreenHeight
#define BottomMargin( rect, mcol, mrow ) rect.c.left = 0, rect.c.right = ScreenWidth, rect.c.top = ScreenHeight / mrow * mrow, rect.c.bottom = ScreenHeight

#define KeyLeft 0x6b /* < */
#define KeyRight 0x6c /* > */
#define KeyOK 0x6e /* O */
#define KeyCan 0x6f /* X */

static struct node {
  QUEUE queue;
  BMP bmp;
  char name[128];
} head;
#define QueNext( q ) ((struct node *)(((q) -> queue).next))
#define QuePrev( q ) ((struct node *)(((q) -> queue).prev))

static enum { one, multi } mode;
static int mcol, mrow; /* Number of divisions of the list display */
static struct node *current, *corner;

/* Search for the bitmap of the name specified from the queue */
static struct node *bmp_search( const char *name )
{
  struct node *q;
  const char *a, *b;
  int n;
  n = strlen(name);
  for( q = QueNext( &head ); q != &head; q = QueNext( q ) ) {
    a = q -> name; b = name;
    if (memcmp( a, b, n + 1 ) == 0) {
      return q;
    }
  }
  return NULL;
}

/* Register the bitmap in the queue */
static struct node *bmp_regist( const char *name, BMP *pbmp )
{
  int k; struct node *q;
  k = (pbmp -> rowbytes) * (pbmp -> bounds.c.bottom - pbmp -> bounds.c.top);
  q = (struct node *)malloc( sizeof(struct node) + k );
  if (q == NULL) return NULL;

  memcpy( &(q -> bmp), pbmp, sizeof(BMP) );
  memcpy( q + 1, pbmp -> baseaddr[0], k );
  (q -> bmp).baseaddr[0] = (UB*)(q + 1);
  memcpy( q -> name, name, sizeof(q -> name) );
  QueInsert( (QUEUE*)q, (QUEUE*)&head );

  return q;
}

/* Release of the queue */
static void bmp_gc( void )
{
  struct node *q, *r;
  for( q = QueNext( &head ); q != &head; q = r ) {
    r = QueNext( q );
    QueRemove( (QUEUE*)q );
    free( q );
  }
}
```

```
/* Display of one page */
static void page_put( struct node *q )
{
  RECT rect;
  int i, k, m, n;
  struct node *p;

  current = q;
  switch (mode) {
  case one :
    FullSize( rect );
    if (q == &head) {
      b_gfil_rec( gid0, rect, BLACK100, 0, G_STORE );
    } else {
      b_grst_bmp( gid0, &rect, &(q -> bmp), &((q -> bmp).bounds),
                  NULL, G_STORE | G_CVFORM | G_CVCOLOR );
    }
    break;

  case multi :
    p = q;
    if (p != &head) {
      for( m = -1, p = q; p != &head; p = QuePrev( p ), m++ );
      for( n = m, p = q; p != &head; p = QueNext( p ), n++ );
      /* Present position m in [0, ... ,n - 1] */
      p = &head;
      for( i = (m / (mcol * mrow)) * (mcol * mrow); i >= 0; i-- ) {
        p = QueNext( p );
      }
    }
    corner = p;

    for( i = 0; i < mcol; i++ ) {
      for( k = 0; k < mrow; k++ ) {
        Piece( rect, i, k, mcol, mrow, GapX, GapY );
        if (p == &head) {
          b_gfil_rec( gid0, rect, BLACK100, 0, G_STORE );
        } else {
          b_grst_bmp( gid0, &rect, &(p -> bmp), &((p -> bmp).bounds),
                      NULL, G_STORE | G_CVFORM | G_CVCOLOR );
          p = QueNext( p );
        }
        RightLine( rect, i, k, mcol, mrow, GapX, GapY );
        b_gfil_rec( gid0, rect, BLACK100, 0, G_STORE );
        BottomLine( rect, i, k, mcol, mrow, GapX, GapY );
        b_gfil_rec( gid0, rect, BLACK100, 0, G_STORE );
      }
    }
    RightMargin( rect, mcol, mrow );
    b_gfil_rec( gid0, rect, BLACK100, 0, G_STORE );
    BottomMargin( rect, mcol, mrow );
    b_gfil_rec( gid0, rect, BLACK100, 0, G_STORE );
    break;
  }
}

/* Next page */
static void page_next( void )
{
  struct node *q, *r; int k;
  switch (mode) {
  case one :
    q = QueNext( current );
    if (q == &head) { q = QueNext( q ); }
    page_put( q );
    break;
  case multi :
    for( k = mrow * mcol, q = current;
         --k >= 0 && (r = QueNext( q )) != &head;
         q = r );
    if (k > 0) q = QueNext( &head );
    page_put( q );
    break;
  }
}

/* Previous page */
static void page_prev( void )
{
  struct node *q, *r; int k;
  switch (mode ) {
  case one :
    q = QuePrev( current );
    if (q == &head) { q = QuePrev( q ); }
    page_put( q );
    break;
  case multi :
    for( k = mrow * mcol, q = current;
         --k >= 0 && (r = QuePrev( q )) != &head;
         q = r );
    if (k > 0) q = QuePrev( &head );
    page_put( q );
    break;
  }
}

/* Rotation/shrinking of bitmap */
static void set_bmp( BMP *pbmp )
{
  int w, h; PNT pnt = {0, 0};
  w = (pbmp -> bounds.c.right) - (pbmp -> bounds.c.left);
  h = (pbmp -> bounds.c.bottom) - (pbmp -> bounds.c.top);
  if ((w < h) == (WorkWidth < WorkHeight)) {
    b_grst_bmp( wgid[0], &(wbmp[0].bounds), pbmp, &(pbmp -> bounds), NULL, G_STORE );
  } else {
    b_grst_bmp( wgid[1], &(wbmp[1].bounds), pbmp, &(pbmp -> bounds), NULL, G_STORE );
    b_grot_bmp( wgid[1], wgid[0], &(wbmp[1].bounds), &pnt, 90*3, NULL, G_STORE );
  }
}

static int file_read( UB *buf, UW reqsize, int *fd )
{
  return read( *fd, buf, reqsize );
}

/* Read in a JPEG file and convert it into a bitmap */
static int jpeg_bmp( const char* path, BMP *pbmp )
{
  int fd, er; IMG_BMP img_bmp;
  IMG_COMPACT comp = { LIBIMG_METHOD_JPEG, LIBIMG_JPEG_READ_OPT_1_1_SIZE };

  fd = open( path, O_RDONLY );
  if (fd < 0) return fd;

  pbmp -> pixbits = PixBits;
  img_bmp.color_spec = &CSpec;
  img_bmp.bmap = pbmp;
  img_bmp.funcptr = file_read;
  img_bmp.flushptr = NULL;
  img_bmp.io_src = &fd;
  er = libimg_rea_bmp( &img_bmp, &comp );

  close( fd );

  return er;
}

/* Investigate whether or not the character string ends in .jpg */
static int match( const char *name )
{
  const char *p; int k;
  for( p = name, k = 0; *p; k++, p++ );
  return (k >= 4
          && p[-4] == '.'
          && toupper(p[-3]) == 'J'
          && toupper(p[-2]) == 'P'
          && toupper(p[-1]) == 'G' );
}

/* Recursively search for all the .jpg files at the bottom of the path,
  and convert them into bitmaps */
static void list( char *path )
{
  DIR *f; struct dirent *d; char *t;
  static BMP bmp; struct node *q;

  for( t = path; *t != '\0'; t++ ); /* Search for the tail of the path name */
  f = opendir( path );
  if (f == NULL) return;
  while( (d = readdir( f )) != NULL ) {
    *t = '/';
    strcpy( t + 1, d -> d_name );
    if (d -> d_type == DT_DIR) { /* In the case of a directory */
      if (strcmp( d -> d_name, "." ) == 0
          || strcmp( d -> d_name, ".." ) == 0) {
        /* Skip over reading . and .. */
      } else {
        list( path ); /* Recursively follow the directory */
      }
    } else if (match( d -> d_name )) {/* In the case of a *.jpg file*/
      Debug(( "%s\n", path ));
      if (bmp_search( path ) != NULL) {
      } else if (jpeg_bmp( path, &bmp ) >= 0) {
        set_bmp( &bmp );
        free( bmp.baseaddr[0] );
        q = bmp_regist( path, &(wbmp[0]) );
        if (q != NULL) {
          page_put( q );
        }
      }
    }
  }
  closedir( f );
  *t = '\0';
}

/* Read in card */
static int read_card( void )
{
  int r; static char buf[512];

  r = attach( devnm, CONNM, UX_MSDOS ); /* Connection */
  if (r < 0) return r;

  strcpy( buf, "/" CONNM );
  list( buf );

  detach( devnm, 0 ); /* Cut-off */
  return 0;
}

/* Event processing */
static void evtp( void )
{
  int i, k; EVENT evt; struct node *d; static KeyTab keytab;

  keytab.keymax = 256; keytab.kctmax = 1;
  keytab.kctsel[0] = 0;
  for(i = 0; i < keytab.keymax; i++) keytab.kct[i] = i;
  b_set_ktb(&keytab);
  b_chg_emk( EM_DEVICE | EM_KEYDWN | EM_BUTDWN );

  for(;;) {
    switch (b_get_evt( EM_ALL, &evt, CLR )) {
    case EV_DEVICE : /* Device event */
      Debug(("EV_DEVICE %d\n", evt.data.dev.kind));
      if (evt.data.dev.kind == 1) { /* Insertion */
        read_card();
      }
      break;
    case EV_KEYDWN : /* Key down event */
      Debug(("EV_KEYDWN %#x\n", evt.data.key.keytop));
      switch(evt.data.key.keytop) {
      case KeyLeft :
        page_prev();
        break;
      case KeyRight :
        page_next();
        break;
      case KeyOK :
        switch (mode) {
        case one :
          if (current != &head) {
            mode = multi;
            page_put( current );
          }
          break;
        }
        break;
      case KeyCan :
        return; /* Terminate */
        break;
      }
      break;
    case EV_BUTDWN : /* Event when touch panel is touched */
      Debug(("EV_BUTDWN\n"));
      switch (mode) {
      case one :
        i = 2 * evt.pos.x / ScreenWidth;
        if (i == 0) {
          page_prev();
        } else {
          page_next();
        }
        break;
      case multi :
        k = (mrow * evt.pos.y / ScreenHeight)
          + (mcol - 1 - (mcol * evt.pos.x / ScreenWidth)) * mrow;
        for( i = k, d = corner; --i >= 0; d = QueNext( d ) ) {
          if (d == &head) break;
        }
        if (d != &head) {
          mode = one;
          page_put( d );
        }
        break;
      }
      break;
    }
  }
}

W main( W ac, TC *av[] )
{
  int i;
  const TC scr[] = {L"SCREEN"};
  DEV_SPEC devspec;

  /*Initialization */
  QueInit( (QUEUE*)&head );
  mode = multi; mcol = mrow = 3;
  current = corner = &head;

  b_gdsp_ptr( 0 ); /* Pointer elimination */
  gid0 = b_gopn_dev( (TC*)scr, NULL );
  b_gset_bcs( gid0, &CSpec );
  b_gget_spc( (TC*)scr, &devspec );
  ScreenWidth = devspec.hpixels; ScreenHeight = devspec.vpixels;
  WorkWidth = ScreenWidth; WorkHeight = ScreenHeight;
  for(i = 0; i < 2; i++) {
    wbmp[i].planes = 1;
    wbmp[i].pixbits = PixBits;
    wbmp[i].rowbytes = sizeof(UH) *
      (wbmp[i].bounds.c.right = (i ? WorkHeight : WorkWidth));
    wbmp[i].bounds.c.bottom = (i ? WorkWidth : WorkHeight);
    wbmp[i].bounds.c.left = wbmp[i].bounds.c.top = 0;
    wbmp[i].baseaddr[0] = (UB*)malloc( WorkWidth * WorkHeight * (PixBits >> 8) );
    wgid[i] = b_gopn_mem( NULL, &(wbmp[i]), (B*)&CSpec );
  }

  /* Event processing */
  read_card();
  evtp();

  /* Clean up afterward */
  bmp_gc();
  for(i = 0; i < 2; i++) {
    b_gcls_env( wgid[i] );
    free( wbmp[i].baseaddr[0] );
  }
  b_gcls_env( gid0 );
  return 0;
}
```

Conclusion

As is in List 3 or the Application Example, we can read in via the USB and process on T-Engine JPEG images taken with a digital camera. At that time, the FAT file access we introduced on this occasion and the powerful image processing functions of T-Shell are useful. Also, because we can access via the same methods even a CD-ROM connected on the USB, we can easily realize such things as an electronic book, for example. Please by all means make use of T-Engine for which the range of applications is expanding more and more.

Reference

TE-FAT-Readme

```
-----------------------------------------------------
              TRONWARE VOL. 84 Recording
       T-Engine FAT File Access Sample Programs
     Copyright (C) 2003 Personal Media Corporation
-----------------------------------------------------
Sample Program Outline
```

```
test1 : File Read-in

  Connect USB disk uda0 with the connection name A,
  read in the file /DIR1/IMAGE/cat.jpg with fopen, fread, and
  do a hexadecimal dump.

test2 : Recursively Search the Directories

  By means of opendir, readdir, recursively search
  all the directories on a disk and output a path name.

test3 : Display of a JPEG Image (T-Shell required)

  Read in with open and read the JPEG file /DIR1/IMAGE/cat.jpg,
  using the T-Shell image format conversion library, convert it into a 65,536 color bitmap,
  and by means of T-Engine display primitives display it on the screen.

test 4 : JPEG Image Viewer (T-Shell required)

      Designed so that it runs on the minimal configuration of T-Shell. As for T-Shell,
      please load in advance only the display primitives with lodspg dp.

  [Program Outline]
  •Automatically detects that the memory card has been inserted, and begins read-in.
    For device insertion detection, we use the Event Manager of T-Kernel Extension.
  •In the application of test2, we recursively search all the directories on the memory card,
    and we read in all the *.jpg files.
  •In the application of test3, the read-in JPEG files
    are converted into bitmaps and preserved.
  •For the input detection of the touch panel and buttons also,
    we use the Event Manager of T-Kernel Extension.
  •In a case where the LCD screen is vertically arranged and the images are horizontally arranged,
    we display by rotating 90 degrees with the b_grot_bmp API of T-Shell.

  [Operations]
  •There are two modes.
    In the list mode, we display by shrinking the images into 3 x 3 = 9.
    In the basic mode, we display one image.
  •We move the next page/previous page with the [←][→]  buttons.
  •When we touch an image in the list mode, we expand that image and display it in the basic mode.
  •When we press the [O] button, we move into list mode.
  •When we press the [X] button, we terminate.

Directory Configuration

 tw84fat --- src -+- test1.C   File read-in
                  |
                  +- test2.C   Recursively search the directories
                  |
                  +- test3.C   JPEG image display
                  |
                  +- test4.C   JPEG image viewer
                  |
                  +- Makefile  Make file

Make/Execute Methods

Referencing the following examples, perform make in the development environment on Linux,
and execute on top of T-Engine.

In the Case of test1, test2

Linux$ cd tw84fat
Linux$ mkdir sh7727            Create a directory for use on sh7727
Linux$ cd sh7727
Linux$ ln -s ../src/Makefile   Paste a symbolic link in Makefile
Linux$ gmake test2             Make test2

Linux$ $BD/etc/gterm -3 -l/dev/ttyS0  Start up the terminal software gterm

    Turn on the T-Engine power supply and start up the CLI.

[/SYS]% att pca0 p             Here, we use the work disk on the PC Card
[/SYS]% cd /p
[/p]% recv -d test2            Transmit to the disk the test2 execution file

    Connect to T-Engine via the USB the memory card in
    which the directories and files are stored in FAT format.

[/p]% test2                    Execute test2

In the case of test3, test4

Linux$ cd tw84fat
Linux$ mkdir sh7727            Create a directory for use on sh7727
Linux$ cd sh7727
Linux$ ln -s ../src/Makefile   Paste a symbolic link in Makefile
Linux$ gmake test4             Make test4

Linux$ $BD/etc/gterm -3 -l/dev/ttyS0  Start up the terminal software gterm

    Turn on the T-Engine power supply and start up the CLI.
    Without inserting the PC Card, start up from the ROM disk,
    and do not load T-Shell yet.

    Insert into the PC card slot the T-Shell startup disk.

[/SYS]% att pca0 p              Connect to the T-Shell startup disk
[/SYS]% cd /p
[/p]% lodspg dp                 Load the display primitives

[/p]% recv -d test4             Transmit the test4 execution onto the disk

    Connect to T-Engine via the USB the memory card in
    which the directories and files are stored in FAT format.

[/p]% test4                     Execute test4
```

[1] What we call a File Allocation Table (FAT) is originally the name of a data structure for managing the status of use of each cluster on a disk, but in a broad sense, it is also used as the name of a logic format. There are three forms, FAT12, FAT16, and FAT32, in keeping with the number of clusters that can be handled.

[2] At the time this paper was written, compatibility with the SH7751R, VR5500, VR4131, and ARM920-MX1 versions have been released; those who are registered users can download from the user support Web site.

[3] The name "UNIX (file) emulator" is derived from the fact that a file access API based on a UNIX system call is employed. At the present time, UNIX functions outside of file access have not been implemented.

The above article on T-Engine programming appeared on pages 94-97 in Vol. 84 of *TRONWARE* . It was translated and loaded onto this Web page with the permission of Personal Media Corporation.