Embedded Linux Conference
November 5-7, 2012
Barcelona, Spain

Texas Instruments
Linux Development Center

FFSB and IOzone

File system Benchmarking Tools, Features and Internals

Keshava Munegowda , Sourav Poddar
Texas Instruments (India) Pvt Ltd
Bangalore.

Dr. G T Raju
Professor and Head, Computer Science and Engineering Department,
R N S Institute of Technology, Bangalore, India.

Agenda

- FFSB overview
- FFSB profile files
- FFSB internals
- IOzone overview
- IOzone internals
- Porting of FFSB to ARM based platforms
- Porting of IOzone to ARM based platforms
- Example Results
- FFSB v/s Iozone Summary
- References

FFSB overview

- ***F***lexible ***F***ile ***S***ystem ***B***enchmarking tool
- Uses the Pthreads for the performance benchmarking
- Uses profile files as input
  - Example: ffsb <profile_file>
- Limits the maximum time of benchmarking
- Multiple file systems to benchmark
- Flexible probablities for read, write, append, delete etc operations

Profile files of FFSB

- Input to FFSB tool to specify
  - Global options
  - File systems
    - Options per file system
  - Thread Groups
    - Options per Thread group

Global Options of Profile files

- time
  - Duration of Benchmarking operations
- Directio
  - File open, read and writes are made with the option “O_DIRECT”.
  - No Buffers are used by kernel; direct device read/writes are performed.
- Alignio
  - All buffered opertions are aligned to 4K boundaries
- callout
  - Specifies an external command to be executed before the Benchmarking starts.

File system options of profile files

- location
  - Mounted directory path (/media/..) of file system
- num_files
  - Number of files (start files) to exist before the benchmarking
- num_dirs
  - Number of directories to exist before the benchmarking
- max_filesize
  - Maximum file size of the start files
- min_filesize
  - Minimum file size of the start files
- create_blocksize
  - Size of the data block to be used to while writing a data to start files
  - Default size is 4K
- Reuse
  - Reuse if files are already existing

File system options of profile files cont....

- agefs
  - Aging of the file system is enabled
  - One threadgroup description should follow this agefs option
    - For example

```
agefs=1
[threadgroup0]
        num_threads=1
        write_blocksize=4096
        write_size=64m
        write_weight=1
[end0]
```

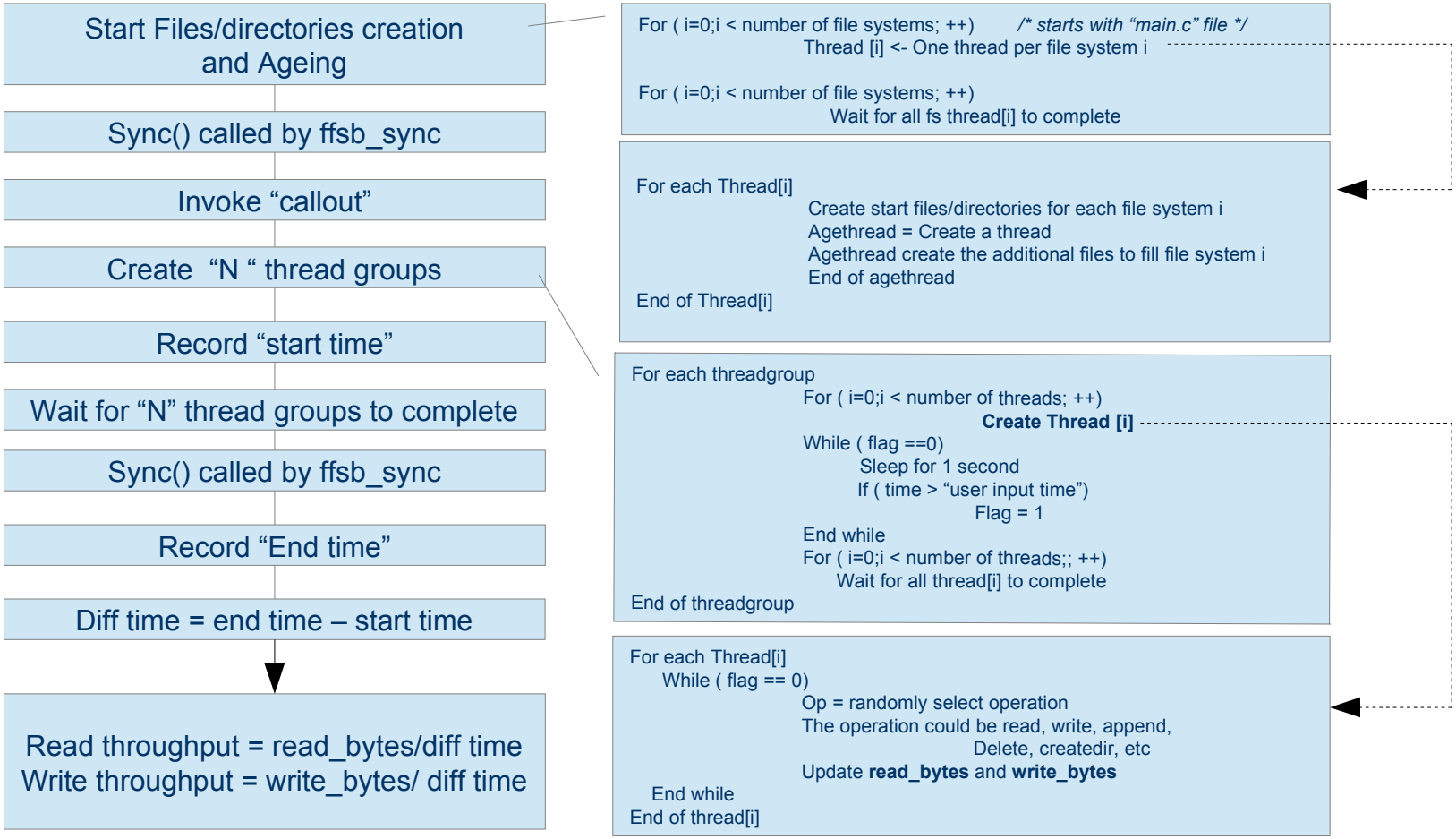
- desired_util
  - File system utilization factor to determine the file system aging
  - This value is (Number of used blocks/ Total number of blocks of file system)
    - For example, desired_util=0.02 means age the file system until 20% is full
- age_blocksize
  - Block size for the file creation
  - Default size is 4K

Thread group options of profile files

- Bindfs
  - specifies the file system mount point on which this thread group operate
- num_threads
  - Number of threads to execute for the benchmarking
- write_size
  - Amount of data written to the file ***in every iteration*** of write performance benchmarking
- read_size
  - Amount of data read from the file ***in every iteration*** of read performance benchmarking
- write_blocksize
  - Block size in bytes used while writing data to benchmark file
- read_blocksize
  - Block size in bytes used while reading data to benchmark file
- write_weight, read_weight, append_weight, delete_weight
  - Probability weight values of the file system operations
- op_delay
  - Delay between each operation

FFSB internals

version: ffsb-6.0-rc2



IOzone

- No profile files
- Options are provided along with IOzone command
  - Example : **iozone -a -I -n 64M -g 64M -r 4k -i 0 -i 1 -b <excel.file>**
    - **a : Auto mode , receives the user input file sizes**
    - **I : use Direct I/O**
    - **n : minimum file size**
    - **g: maximum file size**
    - **r : read/write block size**
    - **i <test number> : test numbers 0 – write/rewrite , 1- read/re-read**
- Modes supports : **Default mode, Auto mode, Througput test.**
- No file system Ageing
- Guaranteed file system operation; No Random selection of opeartion
- No Execution time limitation
- Excel format output

IOzone : Default mode

- Command:
  - Iozone -b <excel.file>
- Performs read/re-read, write/re-write, fread/re-fread, fwrite/re-fwrite etc with
  - 512 KB size
  - Record size : 4K
- User can specify the DIRECT_IO, record size too
- Input minumum/maximum file size are not considered.
- Always the fixed size : 512KB

IOzone : Auto Mode

- Command:
  - Iozone -a -n 4M -g 64M -r 4K -i 0 -i 1 -b <excel.file>
  - a : Automode
- Automode can set minimum and maximum file size
- Repeat the test from minimum size file to maximum size file

IOzone : Auto Mode Internals

- Function : auto_test() [file : iozone.c , version: 3.397]
- For (i= <min_file_size> ; i < Max_file_size>; i*=2) {

```
if ( <r>  ( record size> defined)
    perform_test(i, r);  /* constant record size */
else
    for (rec = 4; k<= i; k*2) {
      perform_test( i, rec)   /* varying record size */
    }
}
```

- The perform_test will be read, write, fread, fwrite, etc

IOzone : Auto Mode Internals

Perform test example: write_perf_test [file: iozone.c]

fd = create file iozone.tmp /* default file name to use if "-f" option not used */

/* DIRECT_IO can be flag will be set while creating a file */

nb = total size to write / record size

start time = record the time

```
for( i=0; i< nb; i++) {
    write(fd, buffer, record size);
}
```

if ("e" is supplied options of iozone) then

fsync(fd)

end time = record the time

diff time = end time – start time

write rate = total size to write / diff time

IOzone: Throughput Test Mode

- Command:
  - Iozone -T -t < value> -I <value> -u <value> -b <excel.file>
  - T : use pthread
    - If <-T> option is not provided, child process (fork()) will be used.
  - I : Minimum number of threads/process
  - u : Maximum number of threads/process
  - t : number of threads/process
    - If -t is provided, options I and u are ignored.
- Uses 512KB file size for write/read
- If <I> and <u> options provided , then
  - Count = <u> - <I> +1 ; Count iterations performed

    For (i=0, k= < I > ;i < count; i++, k++)

    throughput test is performed with k threads /* function: throughput_test will be invoked*/

IOzone: Throughput Test Mode internals

Function: **throughput_test** [file: iozone.c] {

1. start time = record time
2. for (i = 0; i< num_threads; i++) {
   create thread[i] with the function thread_write_test
   }
3. for (i = 0; i< num_threads; i++) {
   wait for thread[i] to complete
   }
4. end time = record time
5. diff time = end time – start time;
6. troughput = total bytes written by num_threads / diff time
7. print parent throughput

....

/* create threads for “read” and wait for them to complete, repeat steps 1 to 7 for read operation */

....

/* create threads for fwrite and wait for them to complete, repeat steps 1 to 7 for fwrite operation */

.....

.....

}

IOzone: Throughput Test internals

Function: **thread_write_test** (thread number) [file: iozone.c]{

fd = create file iozone.DUMMY. < thread number>

/* default file name to use if "-f" option not used */

/* DIRECT_IO can be flag will be set while creating a file */

nb = 512 KB / record size

start time = record the time

for(i=0; i< nb; i++) {

write(fd, buffer, record size);

}

if ("e" is supplied options of iozone) then

fysnc(fd)

end time = record the time

diff time = end time – start time

write rate = total size to write / diff time

print the "Write rate/throuput"

}

IOzone : Cross Compiling for ARM based SOC - OMAP

- Copy the ARM libraries to BusyBox library directory
  - Sudo cp -rf * <...>/arm-2010q1/arm-none-linux-gnueabi/libc/lib/* <...>/busybox/lib
- Modify {CC} and {GCC} values in the Makefile of Iozone source directory
  - CC = arm-none-linux-gnueabi-gcc
  - GCC = arm-none-linux-gnueabi-gcc
    - These compiler strings should indicate the ARM cross compiler
- Run “make linux” command in the iozone source folder
  - The iozone source folder will be generally <..>/iozone_408/src/current
- Copy the generated “iozone” executable file to busybox file system

FFSB : Cross Compiling for ARM based SOC - OMAP

- Copy the ARM libraries to BusyBox library directory
  - Sudo cp -rf * <...>/arm-2010q1/arm-none-linux-gnueabi/libc/lib/* <...>/busybox/lib
- Modify {CC} and {GCC} values in the Makefile of FFSB source directory
  - CC = arm-none-linux-gnueabi-gcc
  - GCC = arm-none-linux-gnueabi-gcc
    - These compiler strings should indicate the ARM cross compiler
- Run “make” command in the FFSB source folder
  - The FFSB source folder will be generally <..>/ffsb-6.0-rc2/
- Copy the generated “ffsb” executable file to busybox file system

Example Results of IOzone and FFSB

	IOzone	FFSB
Read	3.8GB/sec	3.9 GB/sec
Write	45.5MB/sec	5.5MB/sec
Read [DIRECT_IO]	7.3MB/sec	6.88MB/sec
write [DIRECT_IO]	497KB/sec	503KB/sec
Write [with fsync()]	5.7 MB/sec	6MB/sec

- **Performance of Buffer write of IOzone is Higher than buffered write of FFSB**
- One of the reasons could be the FFSB uses sync() after the write opeation and before collecting the time stamp; the **ffsb_sync()[file: main.c]** function calles the sync() system call.
- In FFSB, write sync and write gives almost same performance
- The IOzone does not invoke the sync() before collecting the time stamp after the write operation. Function **write_perf_test [file: iozone.c]** does not use **sync()** function call.
- Usage of **sync()** function in **write_perf_test** , yields the average write performance of 10 MB/sec with IOzone tool i.e, Reduced from 45MB/sec to 10 MB/sec.

Test Setup details	
System	Dell latitude with Intel core i7
RAM	4GB
Linux version	3.2
Storage device	4 GB Transcend thumb drive
File system used in storage device	FAT32
FFSB version	6.0.rc2
IOzone version	3.397
File size	64MB
Write/read block size	4K

FFSB profile file example

```
# Example 1
time=200
directio=0
[filesystem0]
                location=/media/4GB
                num_files=1
                num_dirs=1
                max_filesize=64m
                min_filesize=64m
[end0]
[threadgroup0]
                num_threads=1
                write_blocksize=4096
                write_size=64m
                write_weight=1
                read_weight=0
[end0]
[threadgroup1]
                num_threads=1
                read_blocksize=4096
                read_size=64m
                write_weight=0
                read_weight=1
[end1]

---------- output ----------
Read Throughput: 3.86GB/sec
Write Throughput: 5.96MB/sec
```

```
# Example 2
time=300
directio=0
[filesystem0]
                location=/media/4GB
                num_files=2
                num_dirs=1
                max_filesize=64m
                min_filesize=64m
[end0]

[threadgroup0]
                num_threads=2
                write_blocksize=4096
                write_size=64m
                write_weight=1
                read_blocksize=4096
                read_size=64m
                read_weight=2
[end0]

---------- output ----------
Read Throughput: 13MB/sec
Write Throughput: 6.13MB/sec
```

- **Example 1 and Example 2 uses the same number of threads**
- **Example 2 has the higher read weights than write weights**
- **The read performance of the example2 is lower than example1.**
- **Using a seperate/ dedicated threads for read and write operations gives the better throughtput values.**
- **In Example 1, its observed that a read thread reads more data with in 200 seconds.**

FFSB v/s IOzone

Feature	FFSB	IOzone
Execution	Only Pthread	Current process, child process and pthreads also
Benchmark options/selectives	Profile files	Options provided along with command
directio	yes	yes
MS Execel format output	no	yes
File system operations execution order	Random and based on input weight order	Linear
File system Aging	yes	no
Execution time limit	yes	no

References

- FFSB url: http://sourceforge.net/projects/ffsb/.
- Iozone url: http://www.iozone.org/.

Questions

Queries and Feedback

- Keshava Munegowda
  - keshava_mgowda@ti.com
  - keshava.gowda@gmail.com
- Sourav Poddar
  - sourav.poddar@ti.com