The following options are available in this test mode:

Option	*Description*	*Default value*
--mutex-num	Number of mutexes. The actual mutex to lock is chosen randomly before each lock	4096
--mutex-locks	Number of mutex locks to acquire per each request	50000
--mutex-loops	Number of iterations for an empty loop to perform before acquiring the lock	10000

4.4. memory

This test mode can be used to benchmark sequential memory reads or writes. Depending on command line options each thread can access either a global or a local block for all memory operations.

The following options are available in this test mode:

Option	*Description*	*Default value*
--memory-block-size	Size of memory block to use	1K
--memory-scope	Possible values: global, local. Specifies whether each thread will use a globally allocated memory block, or a local one.	global
--memory-total-size	Total size of data to transfer	100G
--memory-oper	Type of memory operations. Possible values: read, write.	100G

4.5. fileio

This test mode can be used to produce various kinds of file I/O workloads. At the prepare stage SysBench creates a specified number of files with a specified total size, then at the run stage, each thread performs specified I/O operations on this set of files.

When the global `--validate` option is used with the `fileio` test mode,

SysBench performs checksums validation on all data read from the disk. On each write operation the block is filled with random values, then the checksum is calculated and stored in the block along with the offset of this block within a file. On each read operation the block is validated by comparing the stored offset with the real offset, and the stored checksum with the real calculated checksum.

The following I/O operations are supported:

seqwr
: sequential write

seqrewr
: sequential rewrite

seqrd
: sequential read

rndrd
: random read

rndwr
: random write

rndrw
: combined random read/write

Also, the following file access modes can be specified, if the underlying platform supports them:

Asynchronous I/O mode
: At the moment only Linux AIO implementation is supported. When running in asynchronous mode, SysBench queues a specified number of I/O requests using Linux AIO API, then waits for at least one of submitted requests to complete. After that a new series of I/O requests is submitted.

Slow `mmap()` mode

: In this mode SysBench will use `mmap`'ed I/O. However, a

: separate `mmap` will be used for each I/O request due to the limitation of

: 32-bit architectures (we cannot`mmap()` the whole file, as its size migth

: possibly exceed the maximum of 2 GB of the process address space).

Fast `mmap()` mode

On 64-bit architectures it is possible to mmap() the whole file into the process address space, avoiding the limitation of 2 GB on 32-bit platforms.

Using fdatasync() instead of fsync()

Additional flags to open(2)

SysBench can use additional flags to open(2), such as O_SYNC, O_DSYNC and O_DIRECT.

Below is a list of test-specific option for the **fileio** mode:

Option	*Description*	*Default value*
--file-num	Number of files to create	128
--file-block-size	Block size to use in all I/O operations	16K
--file-total-size	Total size of files	2G
--file-test-mode	Type of workload to produce. Possible values: seqwr, seqrewr, seqrd, rndrd, rndwr, rndwr (see above)	*required*
--file-io-mode	I/O mode. Possible values: sync, async, fastmmap, slowmmap (only if supported by the platform, see above).	sync
--file-async-backlog	Number of asynchronous operations to queue per thread (only for --file-io-mode=async, see above)	128
--file-extra-flags	Additional flags to use with open(2)	
--file-fsync-freq	Do fsync() after this number of requests (0 - don't use fsync())	100
--file-fsync-all	Do fsync() after each write operation	no
--file-fsync-end	Do fsync() at the end of the test	yes
--file-fsync-mode	Which method to use for synchronization. Possible values: fsync, fdatasync (see above)	fsync
--file-merge	Merge at most this number of I/O requests if possible	0

d-requests	(0 - don't merge)	
--file-rw-ratio	reads/writes ration for combined random read/write test	1.5

Usage example:

```
$ sysbench --num-threads=16 --test=fileio --file-total-size=3G
--file-test-mode=rndrw prepare
$ sysbench --num-threads=16 --test=fileio --file-total-size=3G
--file-test-mode=rndrw run
$ sysbench --num-threads=16 --test=fileio --file-total-size=3G
--file-test-mode=rndrw cleanup
```

In the above example the first command creates 128 files with the total size of 3 GB in the current directory, the second command runs the actual benchmark and displays the results upon completion, and the third one removes the files used for the test.

4.6. oltp

This test mode was written to benchmark a real database performance. At the **prepare** stage the following table is created in the specified database (`sbtest` by default):

```
CREATE TABLE `sbtest` (
`id` int(10) unsigned NOT NULL auto_increment,
`k` int(10) unsigned NOT NULL default '0',
`c` char(120) NOT NULL default '',
`pad` char(60) NOT NULL default '',
PRIMARY KEY  (`id`),
KEY `k` (`k`));
```

Then this table is filled with a specified number of rows.

The following execution modes are available at the **run** stage:

Simple

In this mode each thread runs simple queries of the following form:

```
SELECT c FROM sbtest WHERE id=N
```