DBMS Implementation : InnoDB Initialize

woonhak.kang

woonagi319@skku.edu

Contents

Message Log for development

- ib_logf(log_level, “message”)
  - or you can also do fprintf(stderr, “”)
- log levels
  - in file : include/ha_prototypes.h:449

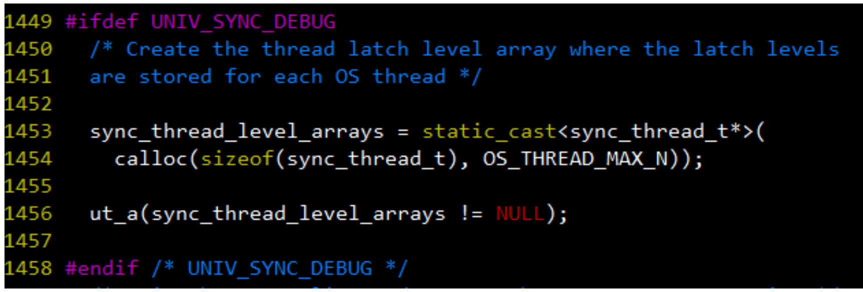
```
enum ib_log_level_t {
  IB_LOG_LEVEL_INFO,
  IB_LOG_LEVEL_WARN,
  IB_LOG_LEVEL_ERROR,
  IB_LOG_LEVEL_FATAL
};
```

InnoDB entry

- `MYSQL_SRC`
  - `e.g.) /home/woonhak/workspace/mysql-5.6.21`
- `INNODB_SRC`
  - `$MYSQL_SRC/storage/innobase/`
- InnoDB entry point
  - in file: srv/srv0start.cc:1502 innobase_start_or_create_for_mysql(void)
  - Starts InnoDB and creates a new database if database files are not found and the user wants.

Coding conventions

- srv_XXX : global system variables
- end of #if~#else~#endif directive
- module/function name
  - use abbr.
    - for e.g.)
    - dictionary -> dict, server->srv, utility->ut, memory->mem, buffer->buf
- Assertion
  - ut_a(), ut_ad()



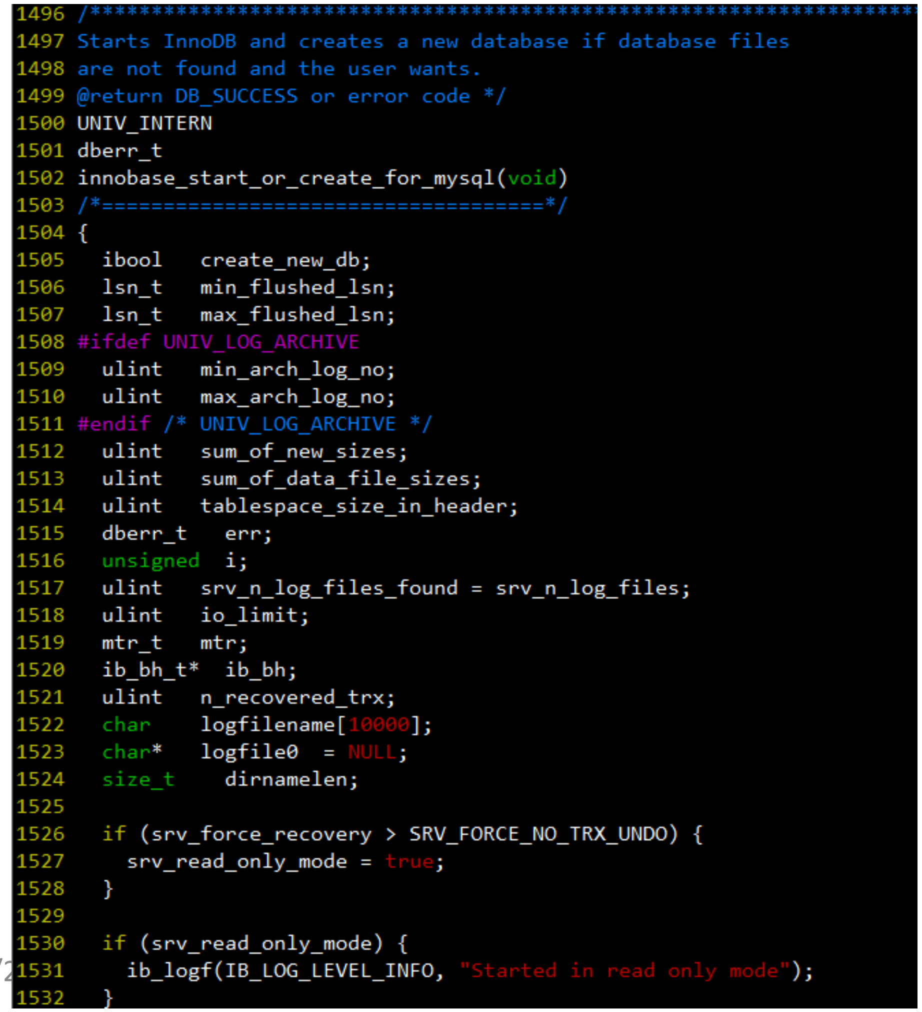
```
#ifdef UNIV_SYNC_DEBUG
  /* Create the thread latch level array where the latch levels
  are stored for each OS thread */

  sync_thread_level_arrays = static_cast<sync_thread_t*>(
    calloc(sizeof(sync_thread_t), OS_THREAD_MAX_N));

  ut_a(sync_thread_level_arrays != NULL);

#endif /* UNIV_SYNC_DEBUG */
```

InnoDB Initialize

- We are under
  - Linux
  - native aio (libaio)
  - use O_DIRECT
  - does not use raw_disk (files on file system)
  - default database files are already created
- Initialize
  - aio system
  - file system
  - buffer pool
  - log system

InnoDB Initialize

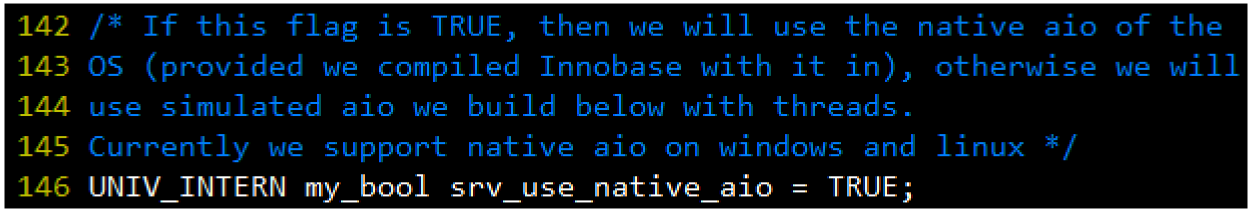
- srv/srv0start.cc:1502



```
1496 /*******************************************************************
1497 Starts InnoDB and creates a new database if database files
1498 are not found and the user wants.
1499 @return DB_SUCCESS or error code */
1500 UNIV_INTERN
1501 dberr_t
1502 innobase_start_or_create_for_mysql(void)
1503 /*====================================*/
1504 {
1505 	ibool	create_new_db;
1506 	lsn_t	min_flushed_lsn;
1507 	lsn_t	max_flushed_lsn;
1508 #ifdef UNIV_LOG_ARCHIVE
1509 	ulint	min_arch_log_no;
1510 	ulint	max_arch_log_no;
1511 #endif /* UNIV_LOG_ARCHIVE */
1512 	ulint	sum_of_new_sizes;
1513 	ulint	sum_of_data_file_sizes;
1514 	ulint	tablespace_size_in_header;
1515 	dberr_t	err;
1516 	unsigned	i;
1517 	ulint	srv_n_log_files_found = srv_n_log_files;
1518 	ulint	io_limit;
1519 	mtr_t	mtr;
1520 	ib_bh_t*	ib_bh;
1521 	ulint	n_recovered_trx;
1522 	char	logfilename[10000];
1523 	char*	logfile0	= NULL;
1524 	size_t	dirnamelen;
1525 
1526 	if (srv_force_recovery > SRV_FORCE_NO_TRX_UNDO) {
1527 		srv_read_only_mode = true;
1528 	}
1529 
1530 	if (srv_read_only_mode) {
1531 		ib_logf(IB_LOG_LEVEL_INFO, "Started in read only mode");
1532 	}
```

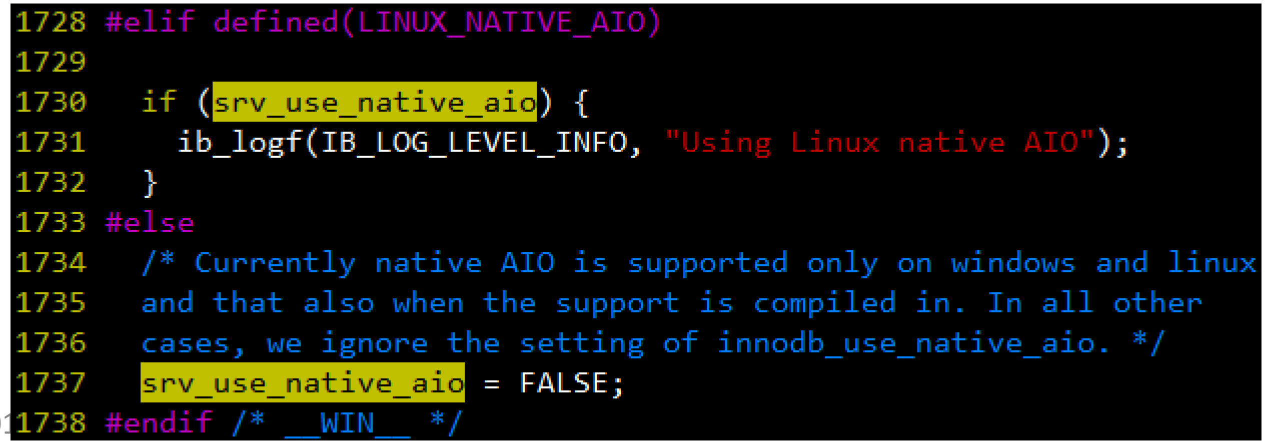
InnoDB Initialize

- `srv_use_native_aio`
  - `include/srv0srv.cc`



```
142 /* If this flag is TRUE, then we will use the native aio of the
143 OS (provided we compiled Innobase with it in), otherwise we will
144 use simulated aio we build below with threads.
145 Currently we support native aio on windows and linux */
146 UNIV_INTERN my_bool srv_use_native_aio = TRUE;
```

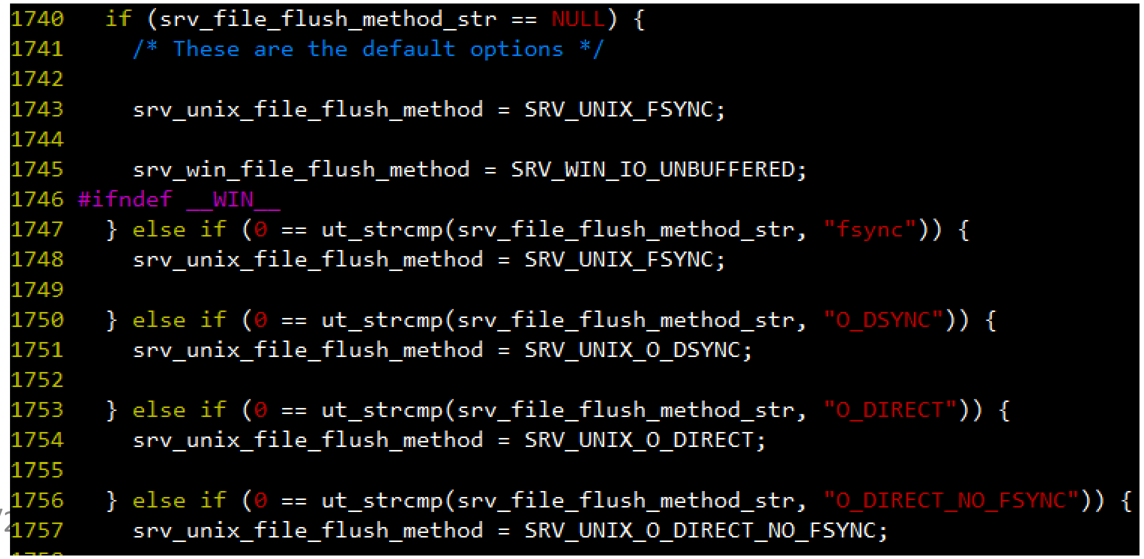
- `srv/srv0start.cc:17xx`
  - Check use native AIO



```
1728 #elif defined(LINUX_NATIVE_AIO)
1729
1730   if (srv_use_native_aio) {
1731     ib_logf(IB_LOG_LEVEL_INFO, "Using Linux native AIO");
1732   }
1733 #else
1734   /* Currently native AIO is supported only on windows and linux
1735   and that also when the support is compiled in. In all other
1736   cases, we ignore the setting of innodb_use_native_aio. */
1737   srv_use_native_aio = FALSE;
1738 #endif /* __WIN__ */
```

InnoDB Initialize

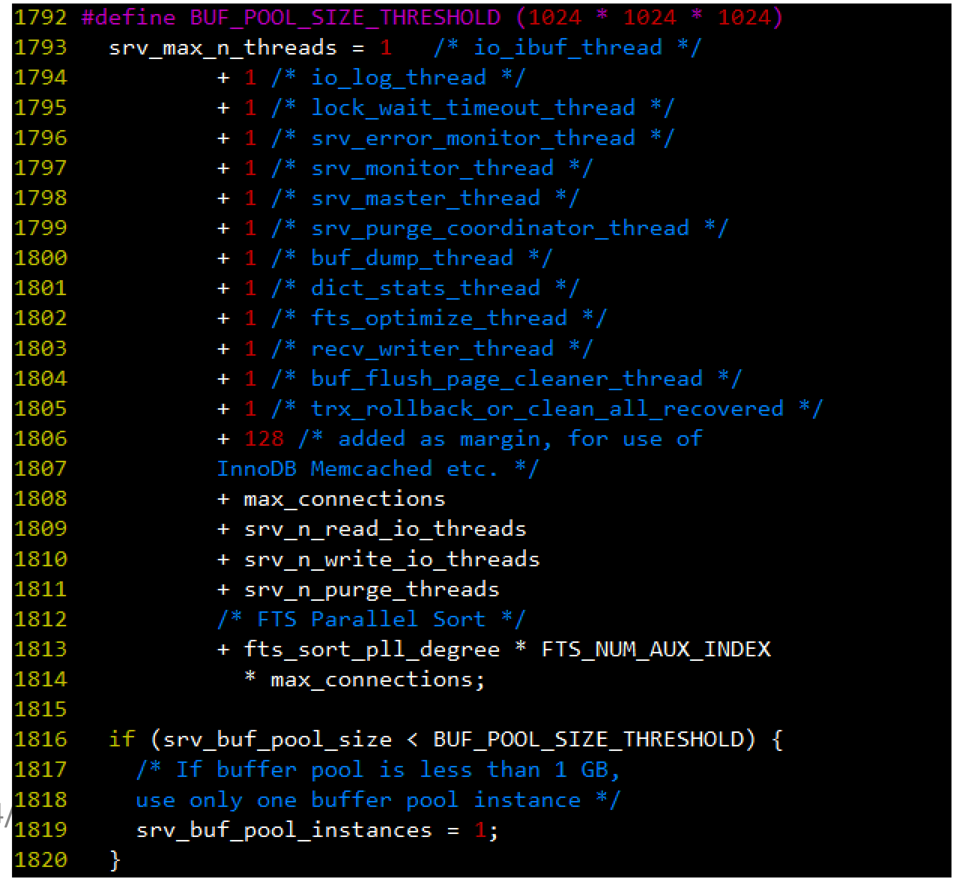
- `srv/srv0start.cc:17XX`
- file flush method : `srv_unix_file_flush_method`
  - `fsync` : use fsync() to flush both log and data - default
  - `o_dsync` : use O_SYNC for log, fsync() for data
  - `o_direct` : use O_DIRECT for data, log is buffered, use fsync() to flush log and data
  - `o_direct_no_fsync` : use O_DIRECT but skip fsync for data



```
  if (srv_file_flush_method_str == NULL) {
    /* These are the default options */

    srv_unix_file_flush_method = SRV_UNIX_FSYNC;

    srv_win_file_flush_method = SRV_WIN_IO_UNBUFFERED;
#ifndef __WIN__
  } else if (0 == ut_strcmp(srv_file_flush_method_str, "fsync")) {
    srv_unix_file_flush_method = SRV_UNIX_FSYNC;

  } else if (0 == ut_strcmp(srv_file_flush_method_str, "O_DSYNC")) {
    srv_unix_file_flush_method = SRV_UNIX_O_DSYNC;

  } else if (0 == ut_strcmp(srv_file_flush_method_str, "O_DIRECT")) {
    srv_unix_file_flush_method = SRV_UNIX_O_DIRECT;

  } else if (0 == ut_strcmp(srv_file_flush_method_str, "O_DIRECT_NO_FSYNC")) {
    srv_unix_file_flush_method = SRV_UNIX_O_DIRECT_NO_FSYNC;
```

buffer pool size threshold

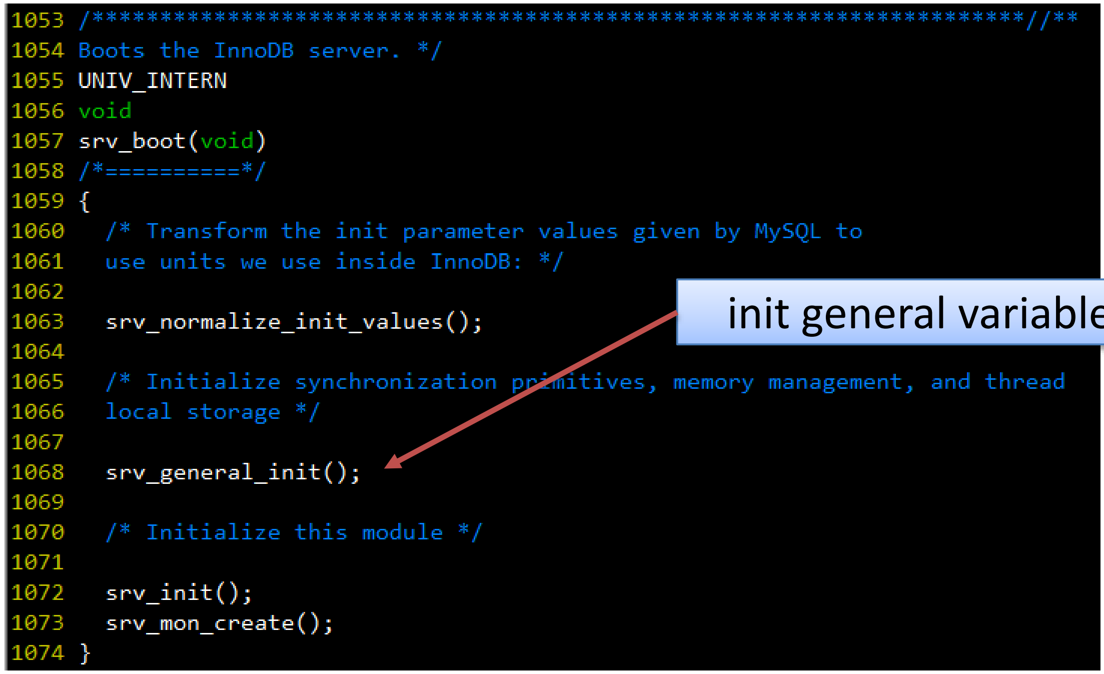
- srv/srv0start.cc
  - if buffer pool size less than threshold than buf pool instances = 1
  - #define BUF_POOL_SIZE_THRESHOLD (1024 * 1024 * 1024)



```
#define BUF_POOL_SIZE_THRESHOLD (1024 * 1024 * 1024)
  srv_max_n_threads = 1   /* io_ibuf_thread */
          + 1 /* io_log_thread */
          + 1 /* lock_wait_timeout_thread */
          + 1 /* srv_error_monitor_thread */
          + 1 /* srv_monitor_thread */
          + 1 /* srv_master_thread */
          + 1 /* srv_purge_coordinator_thread */
          + 1 /* buf_dump_thread */
          + 1 /* dict_stats_thread */
          + 1 /* fts_optimize_thread */
          + 1 /* recv_writer_thread */
          + 1 /* buf_flush_page_cleaner_thread */
          + 1 /* trx_rollback_or_clean_all_recovered */
          + 128 /* added as margin, for use of
          InnoDB Memcached etc. */
          + max_connections
          + srv_n_read_io_threads
          + srv_n_write_io_threads
          + srv_n_purge_threads
          /* FTS Parallel Sort */
          + fts_sort_pll_degree * FTS_NUM_AUX_INDEX
            * max_connections;

  if (srv_buf_pool_size < BUF_POOL_SIZE_THRESHOLD) {
    /* If buffer pool is less than 1 GB,
    use only one buffer pool instance */
    srv_buf_pool_instances = 1;
  }
```

server boot

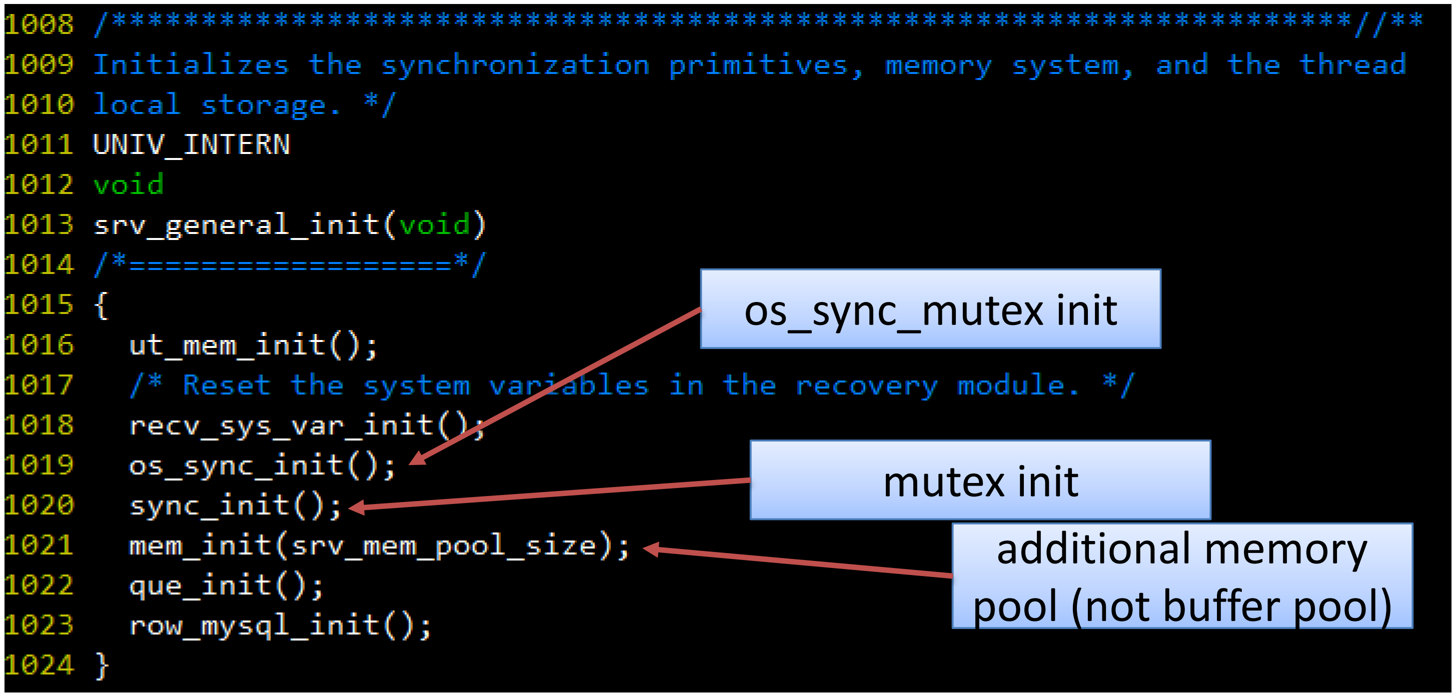
- Boots the InnoDB server
  - invokes `srv_boot()` at `srv0start.cc:1822`
- `srv/srv0srv.cc:1057,srv_boot()`



```
1053 /********************************************************************//**
1054 Boots the InnoDB server. */
1055 UNIV_INTERN
1056 void
1057 srv_boot(void)
1058 /*==========*/
1059 {
1060   /* Transform the init parameter values given by MySQL to
1061   use units we use inside InnoDB: */
1062 
1063   srv_normalize_init_values();
1064 
1065   /* Initialize synchronization primitives, memory management, and thread
1066   local storage */
1067 
1068   srv_general_init();
1069 
1070   /* Initialize this module */
1071 
1072   srv_init();
1073   srv_mon_create();
1074 }
```

server boot

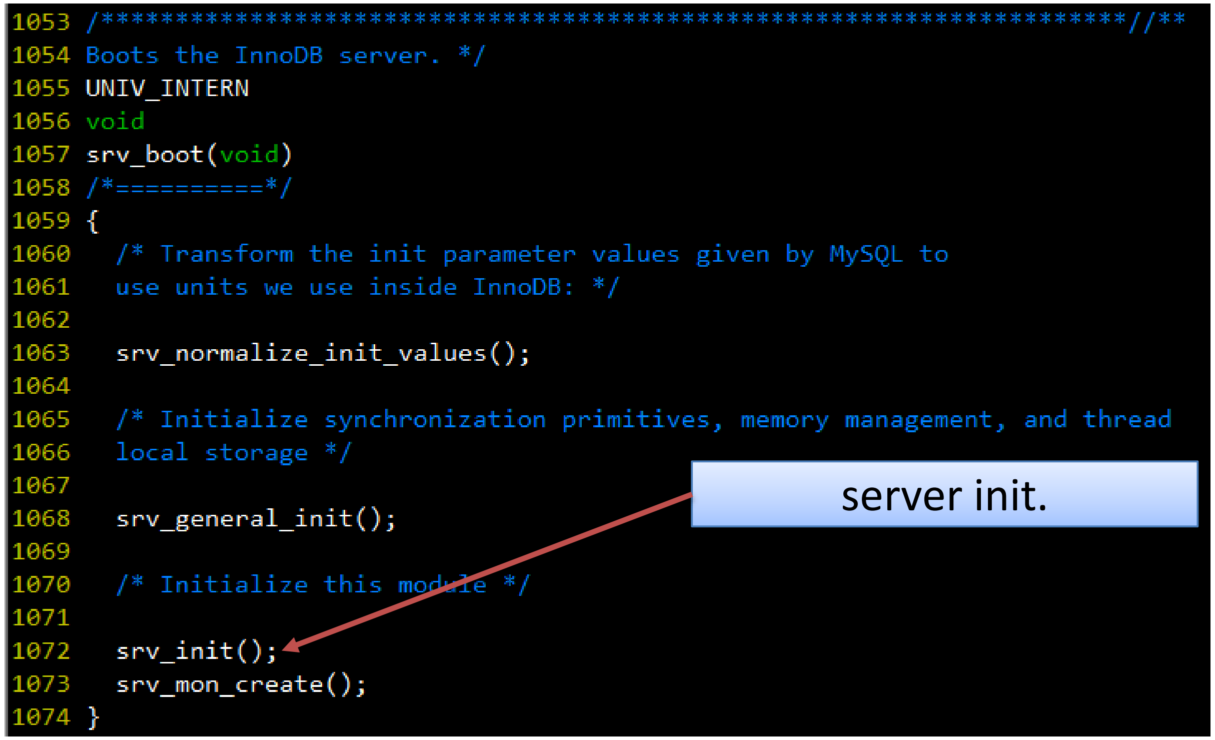
- `srv/srv0srv.cc:1013, srv_general_init()`



```
1008 /*********************************************************************//**
1009 Initializes the synchronization primitives, memory system, and the thread
1010 local storage. */
1011 UNIV_INTERN
1012 void
1013 srv_general_init(void)
1014 /*==================*/
1015 {
1016   ut_mem_init();
1017   /* Reset the system variables in the recovery module. */
1018   recv_sys_var_init();
1019   os_sync_init();
1020   sync_init();
1021   mem_init(srv_mem_pool_size);
1022   que_init();
1023   row_mysql_init();
1024 }
```

server boot

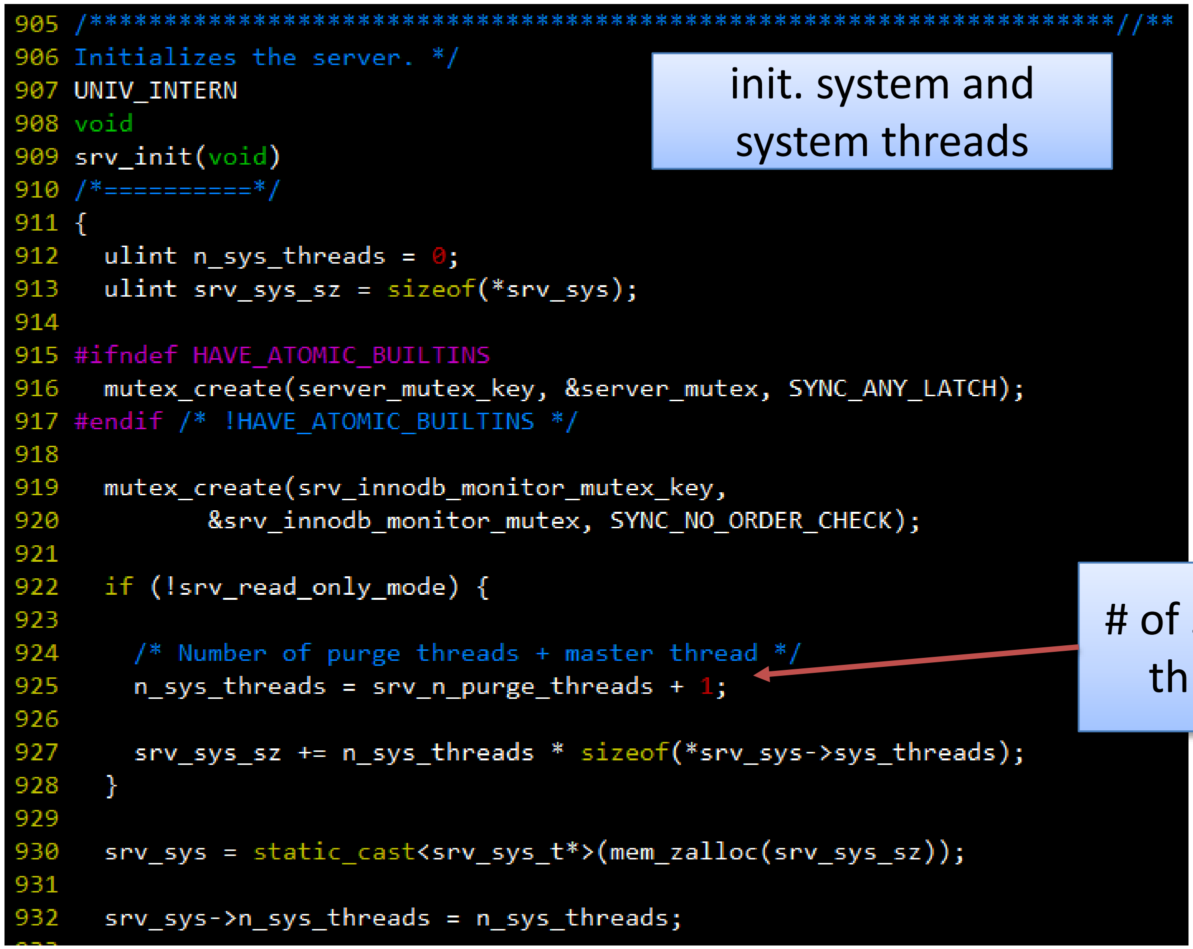
- `srv/srv0srv.cc:1057, srv_boot()`



```
1053 /********************************************************************//**
1054 Boots the InnoDB server. */
1055 UNIV_INTERN
1056 void
1057 srv_boot(void)
1058 /*==========*/
1059 {
1060   /* Transform the init parameter values given by MySQL to
1061   use units we use inside InnoDB: */
1062
1063   srv_normalize_init_values();
1064
1065   /* Initialize synchronization primitives, memory management, and thread
1066   local storage */
1067
1068   srv_general_init();
1069
1070   /* Initialize this module */
1071
1072   srv_init();
1073   srv_mon_create();
1074 }
```

server boot

- srv/srv0srv.cc:909, srv_init()



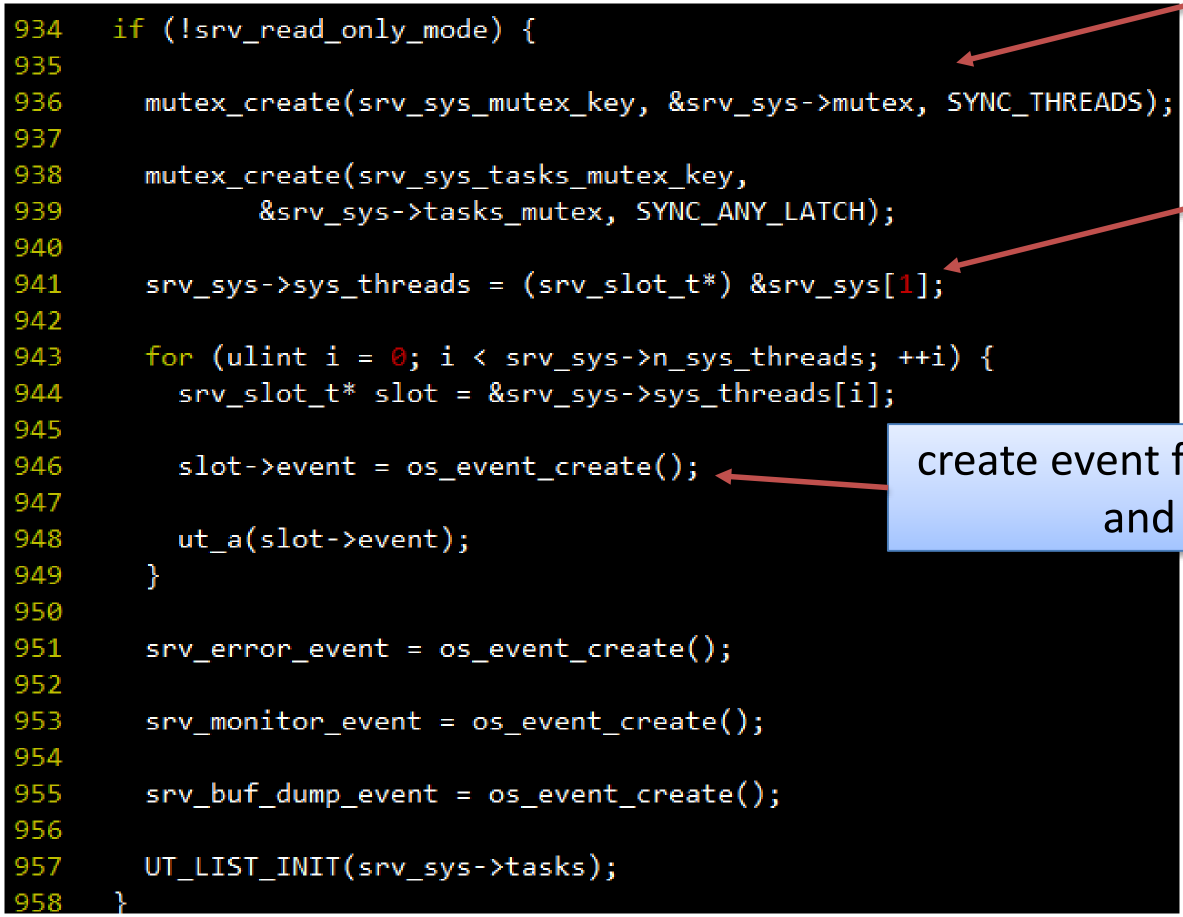
```
905 /*********************************************************************//**
906 Initializes the server. */
907 UNIV_INTERN
908 void
909 srv_init(void)
910 /*==========*/
911 {
912   ulint n_sys_threads = 0;
913   ulint srv_sys_sz = sizeof(*srv_sys);
914 
915 #ifndef HAVE_ATOMIC_BUILTINS
916   mutex_create(server_mutex_key, &server_mutex, SYNC_ANY_LATCH);
917 #endif /* !HAVE_ATOMIC_BUILTINS */
918 
919   mutex_create(srv_innodb_monitor_mutex_key,
920          &srv_innodb_monitor_mutex, SYNC_NO_ORDER_CHECK);
921 
922   if (!srv_read_only_mode) {
923 
924     /* Number of purge threads + master thread */
925     n_sys_threads = srv_n_purge_threads + 1;
926 
927     srv_sys_sz += n_sys_threads * sizeof(*srv_sys->sys_threads);
928   }
929 
930   srv_sys = static_cast<srv_sys_t*>(mem_zalloc(srv_sys_sz));
931 
932   srv_sys->n_sys_threads = n_sys_threads;
```

init. system and system threads

# of system threads = purge thread + master thread

server boot

- srv/srv0srv.cc:909, srv_init()



```
  if (!srv_read_only_mode) {

    mutex_create(srv_sys_mutex_key, &srv_sys->mutex, SYNC_THREADS);

    mutex_create(srv_sys_tasks_mutex_key,
           &srv_sys->tasks_mutex, SYNC_ANY_LATCH);

    srv_sys->sys_threads = (srv_slot_t*) &srv_sys[1];

    for (ulint i = 0; i < srv_sys->n_sys_threads; ++i) {
      srv_slot_t* slot = &srv_sys->sys_threads[i];

      slot->event = os_event_create();

      ut_a(slot->event);
    }

    srv_error_event = os_event_create();

    srv_monitor_event = os_event_create();

    srv_buf_dump_event = os_event_create();

    UT_LIST_INIT(srv_sys->tasks);
  }
```

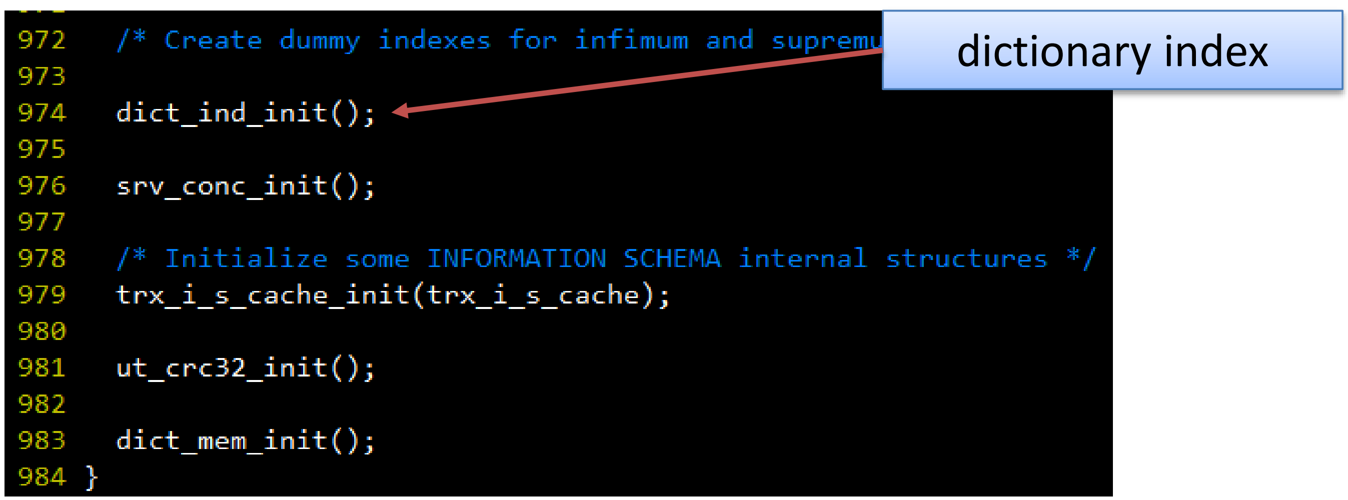
mutex for srv_sys

system threads

create event for synchronization and signaling

server boot

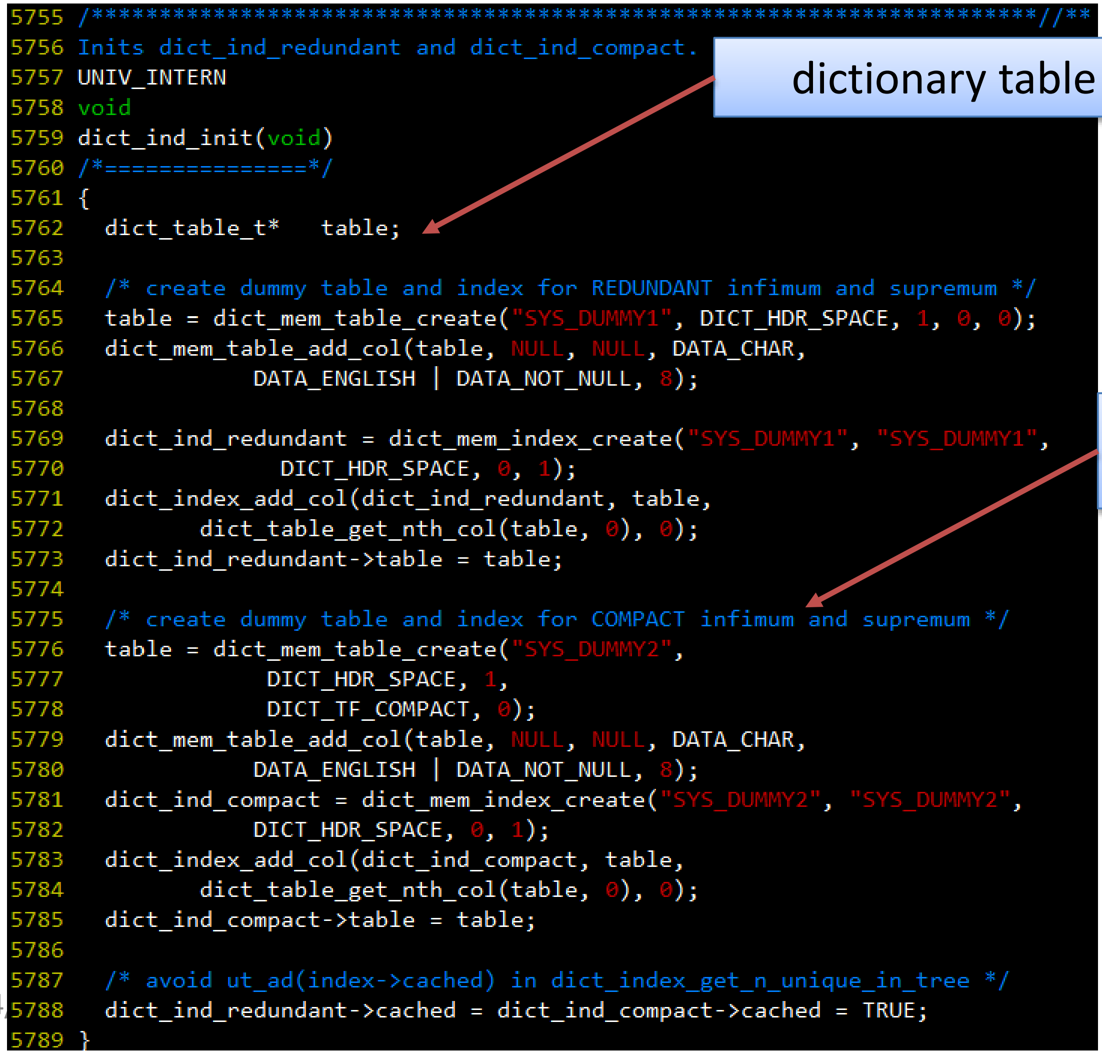
- srv/srv0srv.cc:909, srv_init()



```
  /* Create dummy indexes for infimum and supremu

  dict_ind_init();

  srv_conc_init();

  /* Initialize some INFORMATION SCHEMA internal structures */
  trx_i_s_cache_init(trx_i_s_cache);

  ut_crc32_init();

  dict_mem_init();
}
```

server boot

- dict/dict0dict.cc:5759, dict_ind_init()



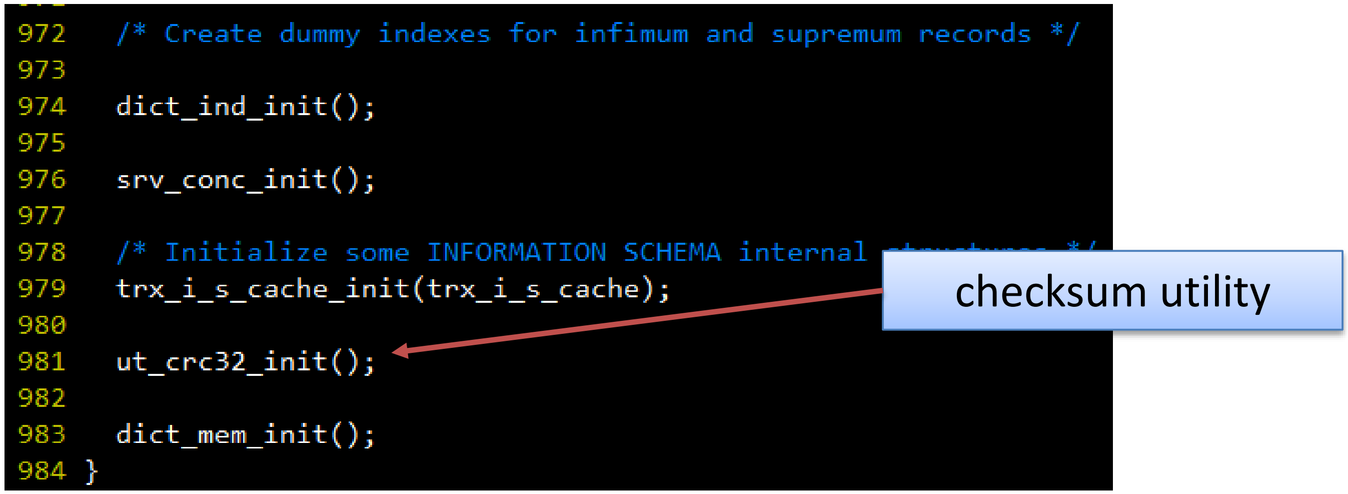
```
5755 /*********************************************************************//**
5756 Inits dict_ind_redundant and dict_ind_compact.
5757 UNIV_INTERN
5758 void
5759 dict_ind_init(void)
5760 /*===============*/
5761 {
5762   dict_table_t*    table;
5763 
5764   /* create dummy table and index for REDUNDANT infimum and supremum */
5765   table = dict_mem_table_create("SYS_DUMMY1", DICT_HDR_SPACE, 1, 0, 0);
5766   dict_mem_table_add_col(table, NULL, NULL, DATA_CHAR,
5767              DATA_ENGLISH | DATA_NOT_NULL, 8);
5768 
5769   dict_ind_redundant = dict_mem_index_create("SYS_DUMMY1", "SYS_DUMMY1",
5770               DICT_HDR_SPACE, 0, 1);
5771   dict_index_add_col(dict_ind_redundant, table,
5772          dict_table_get_nth_col(table, 0), 0);
5773   dict_ind_redundant->table = table;
5774 
5775   /* create dummy table and index for COMPACT infimum and supremum */
5776   table = dict_mem_table_create("SYS_DUMMY2",
5777               DICT_HDR_SPACE, 1,
5778               DICT_TF_COMPACT, 0);
5779   dict_mem_table_add_col(table, NULL, NULL, DATA_CHAR,
5780              DATA_ENGLISH | DATA_NOT_NULL, 8);
5781   dict_ind_compact = dict_mem_index_create("SYS_DUMMY2", "SYS_DUMMY2",
5782              DICT_HDR_SPACE, 0, 1);
5783   dict_index_add_col(dict_ind_compact, table,
5784          dict_table_get_nth_col(table, 0), 0);
5785   dict_ind_compact->table = table;
5786 
5787   /* avoid ut_ad(index->cached) in dict_index_get_n_unique_in_tree */
5788   dict_ind_redundant->cached = dict_ind_compact->cached = TRUE;
5789 }
```

dictionary table

by default : we use compact type

server boot

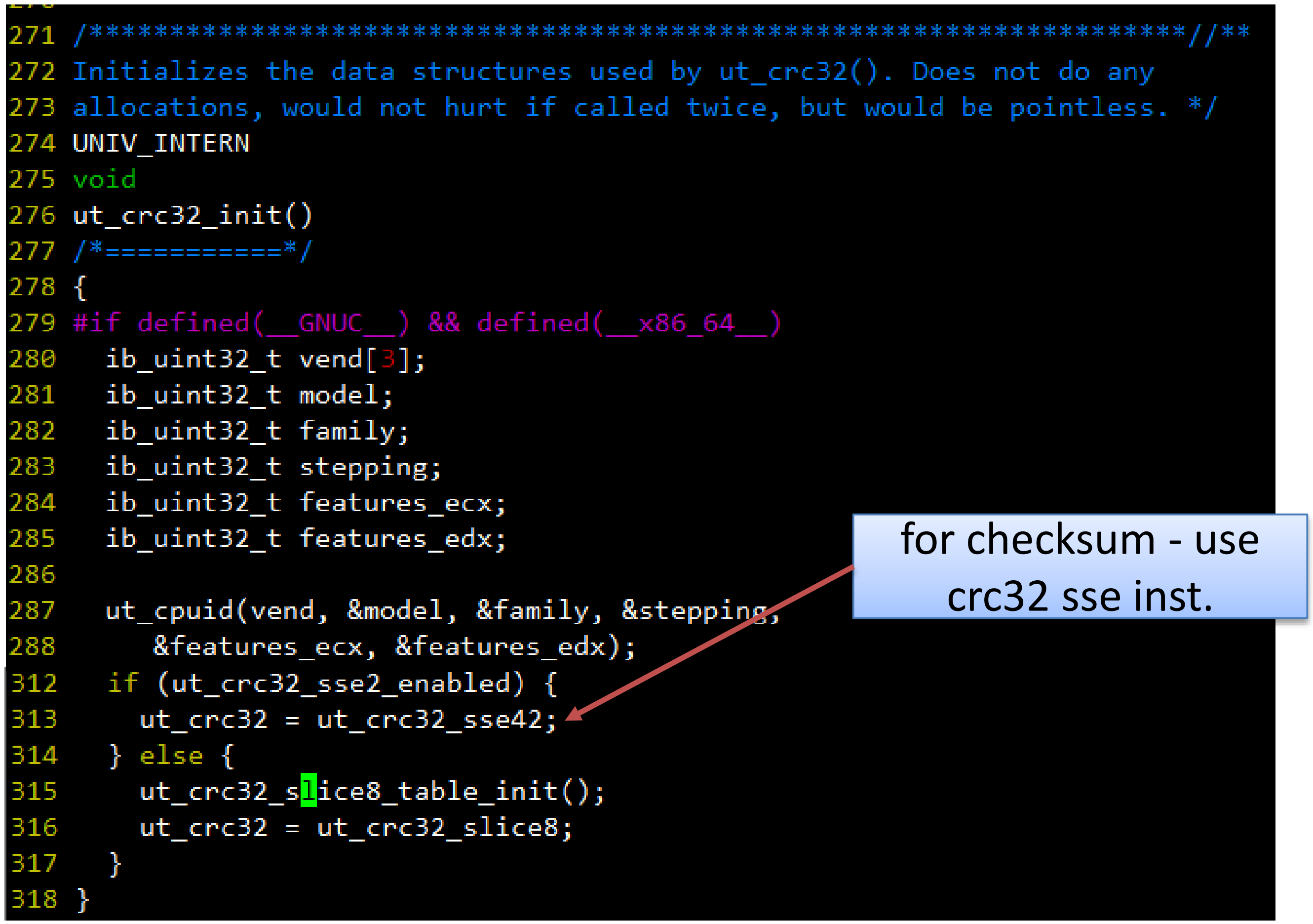
- srv/srv0srv.cc:909, srv_init()



```
	/* Create dummy indexes for infimum and supremum records */

	dict_ind_init();

	srv_conc_init();

	/* Initialize some INFORMATION SCHEMA internal structures */
	trx_i_s_cache_init(trx_i_s_cache);

	ut_crc32_init();

	dict_mem_init();
}
```

server boot

- ut/ut0crc32.cc:276, ut_crc32_init()

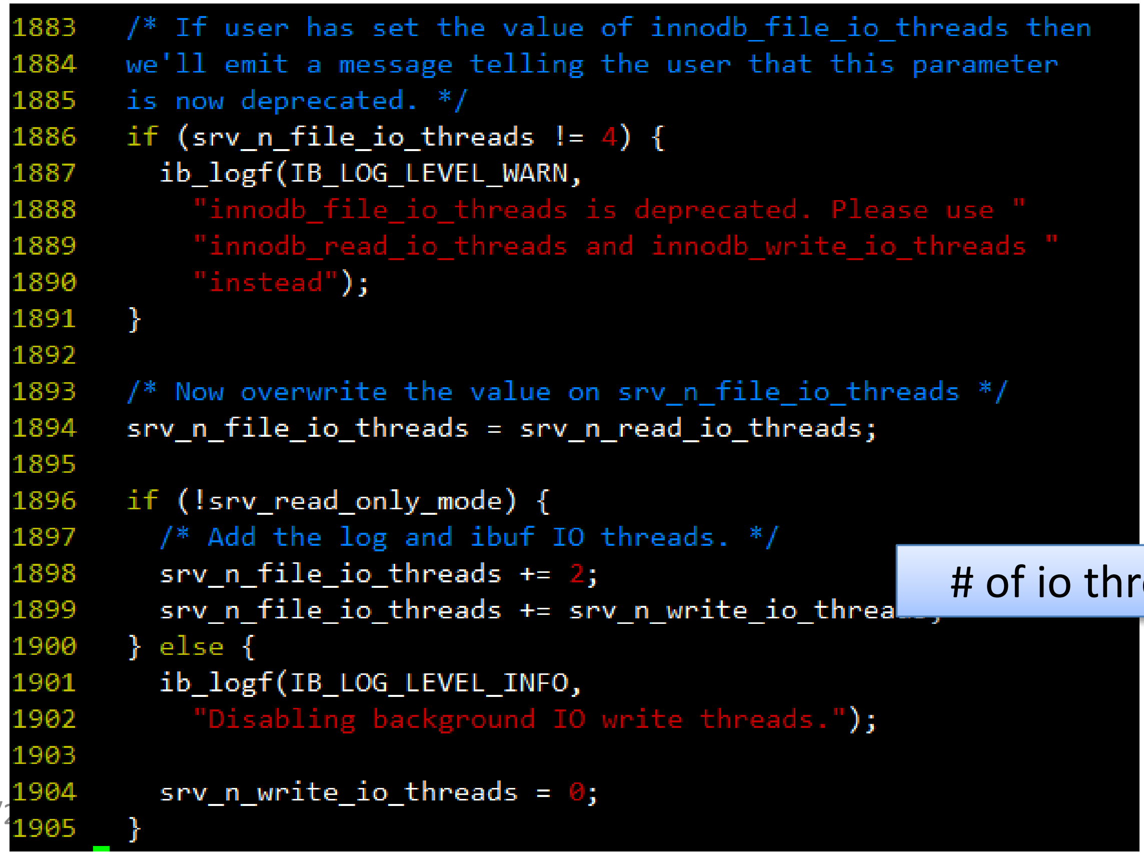


```
271 /********************************************************************//**
272 Initializes the data structures used by ut_crc32(). Does not do any
273 allocations, would not hurt if called twice, but would be pointless. */
274 UNIV_INTERN
275 void
276 ut_crc32_init()
277 /*===========*/
278 {
279 #if defined(__GNUC__) && defined(__x86_64__)
280   ib_uint32_t vend[3];
281   ib_uint32_t model;
282   ib_uint32_t family;
283   ib_uint32_t stepping;
284   ib_uint32_t features_ecx;
285   ib_uint32_t features_edx;
286
287   ut_cpuid(vend, &model, &family, &stepping,
288      &features_ecx, &features_edx);
312   if (ut_crc32_sse2_enabled) {
313     ut_crc32 = ut_crc32_sse42;
314   } else {
315     ut_crc32_slice8_table_init();
316     ut_crc32 = ut_crc32_slice8;
317   }
318 }
```

for checksum - use crc32 sse inst.

InnoDB Initialize

- `srv/srv0start.cc`
- total io threads

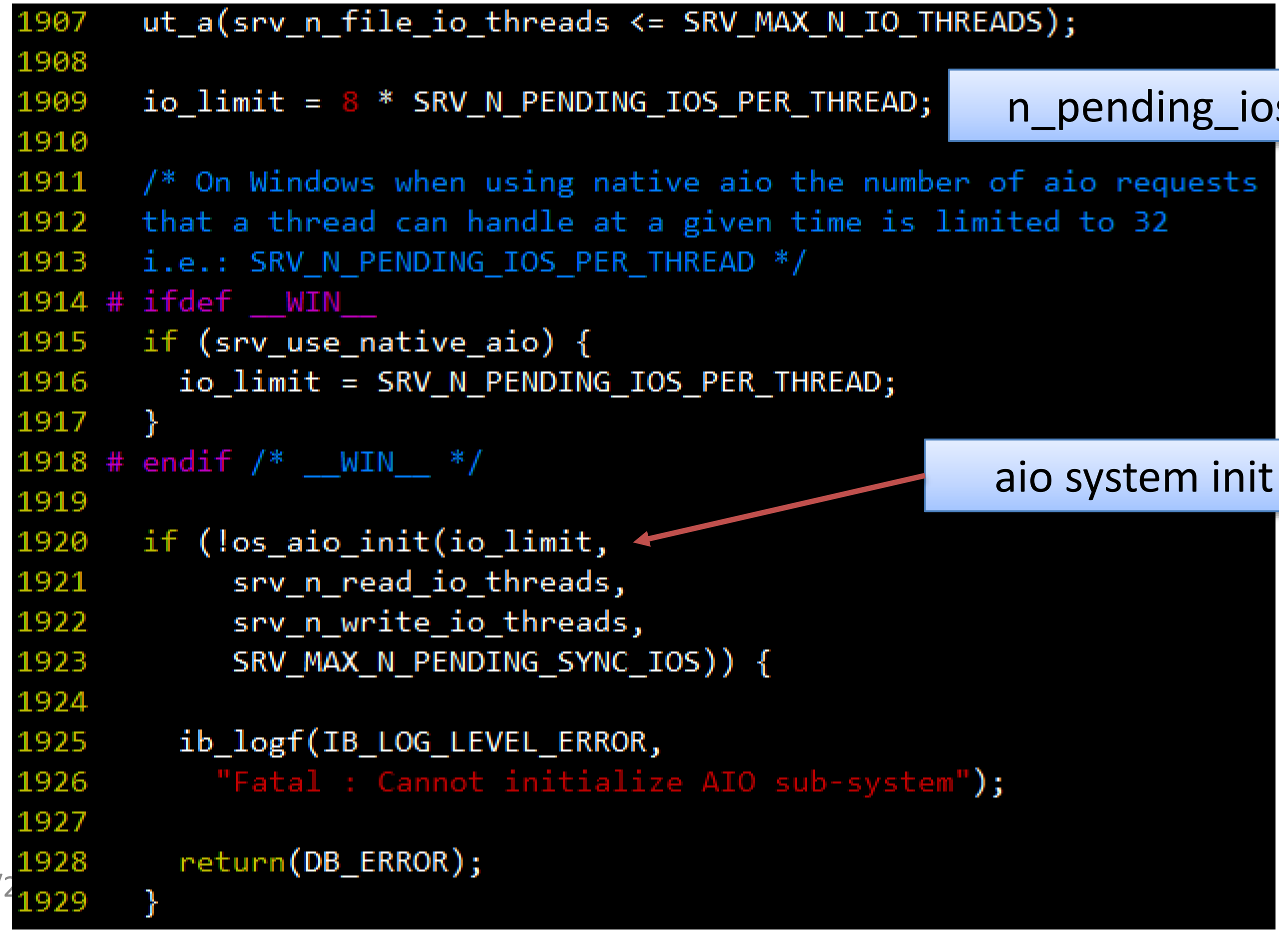


```
/* If user has set the value of innodb_file_io_threads then
we'll emit a message telling the user that this parameter
is now deprecated. */
if (srv_n_file_io_threads != 4) {
  ib_logf(IB_LOG_LEVEL_WARN,
    "innodb_file_io_threads is deprecated. Please use "
    "innodb_read_io_threads and innodb_write_io_threads "
    "instead");
}

/* Now overwrite the value on srv_n_file_io_threads */
srv_n_file_io_threads = srv_n_read_io_threads;

if (!srv_read_only_mode) {
  /* Add the log and ibuf IO threads. */
  srv_n_file_io_threads += 2;
  srv_n_file_io_threads += srv_n_write_io_threa
} else {
  ib_logf(IB_LOG_LEVEL_INFO,
    "Disabling background IO write threads.");

  srv_n_write_io_threads = 0;
}
```

# of io threads = read + write + 2

InnoDB Initialize

- `srv/srv0start.cc`
- aio system init : os_aio_init()



```
ut_a(srv_n_file_io_threads <= SRV_MAX_N_IO_THREADS);

io_limit = 8 * SRV_N_PENDING_IOS_PER_THREAD;

/* On Windows when using native aio the number of aio requests
that a thread can handle at a given time is limited to 32
i.e.: SRV_N_PENDING_IOS_PER_THREAD */
# ifdef __WIN__
if (srv_use_native_aio) {
  io_limit = SRV_N_PENDING_IOS_PER_THREAD;
}
# endif /* __WIN__ */

if (!os_aio_init(io_limit,
     srv_n_read_io_threads,
     srv_n_write_io_threads,
     SRV_MAX_N_PENDING_SYNC_IOS)) {

  ib_logf(IB_LOG_LEVEL_ERROR,
    "Fatal : Cannot initialize AIO sub-system");

  return(DB_ERROR);
}
```

n_pending_ios_per_thread = 32

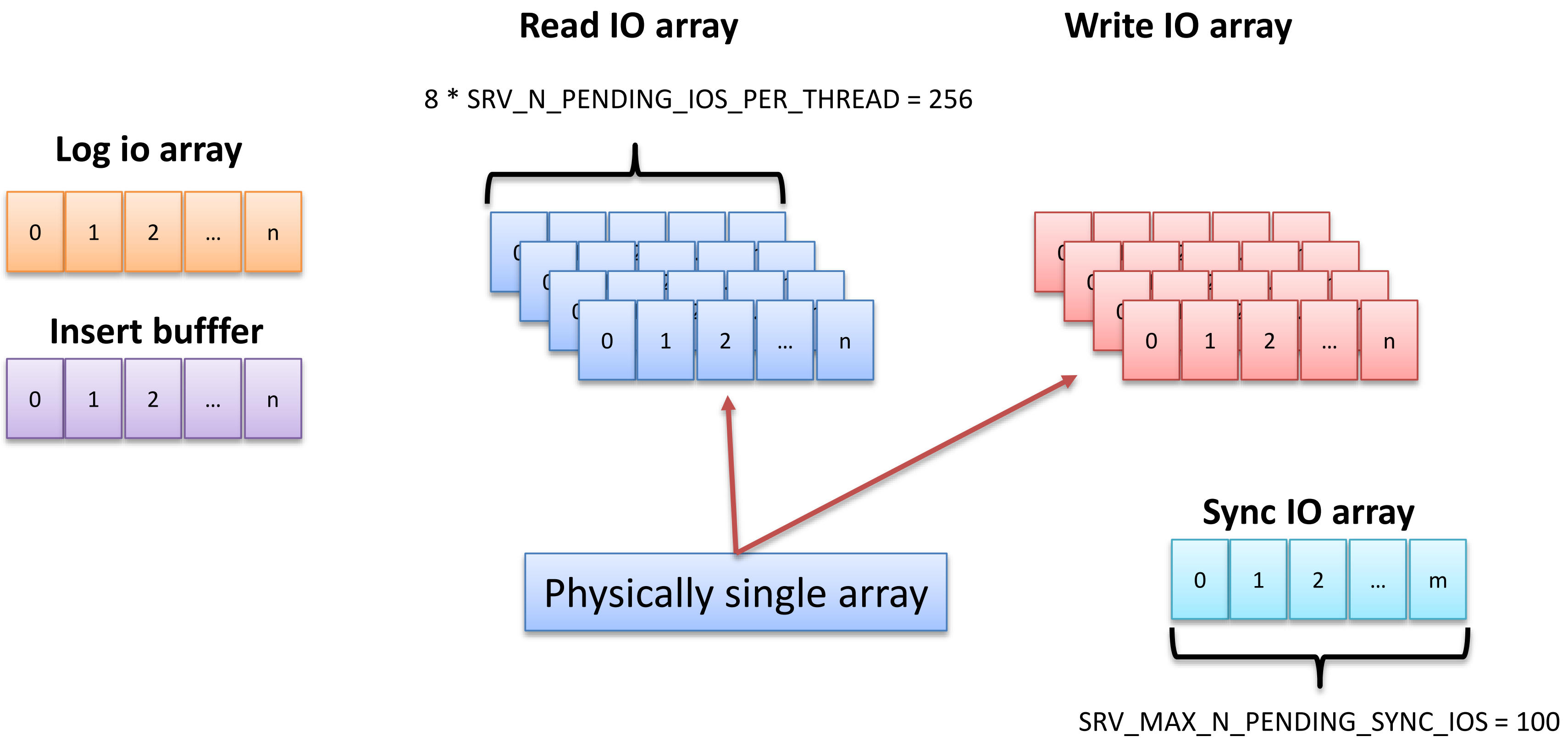
aio system init

aio system init

- Initializes the asynchronous io system.
- Creates one array each for ibuf and log i/o.
- Also creates one array each for read and write where each array is divided logically into n_read_segs and n_write_segs respectively.
- The caller must create an i/o handler thread for each segment in these arrays.
- This function also creates the sync array.
- No i/o handler thread needs to be created for that

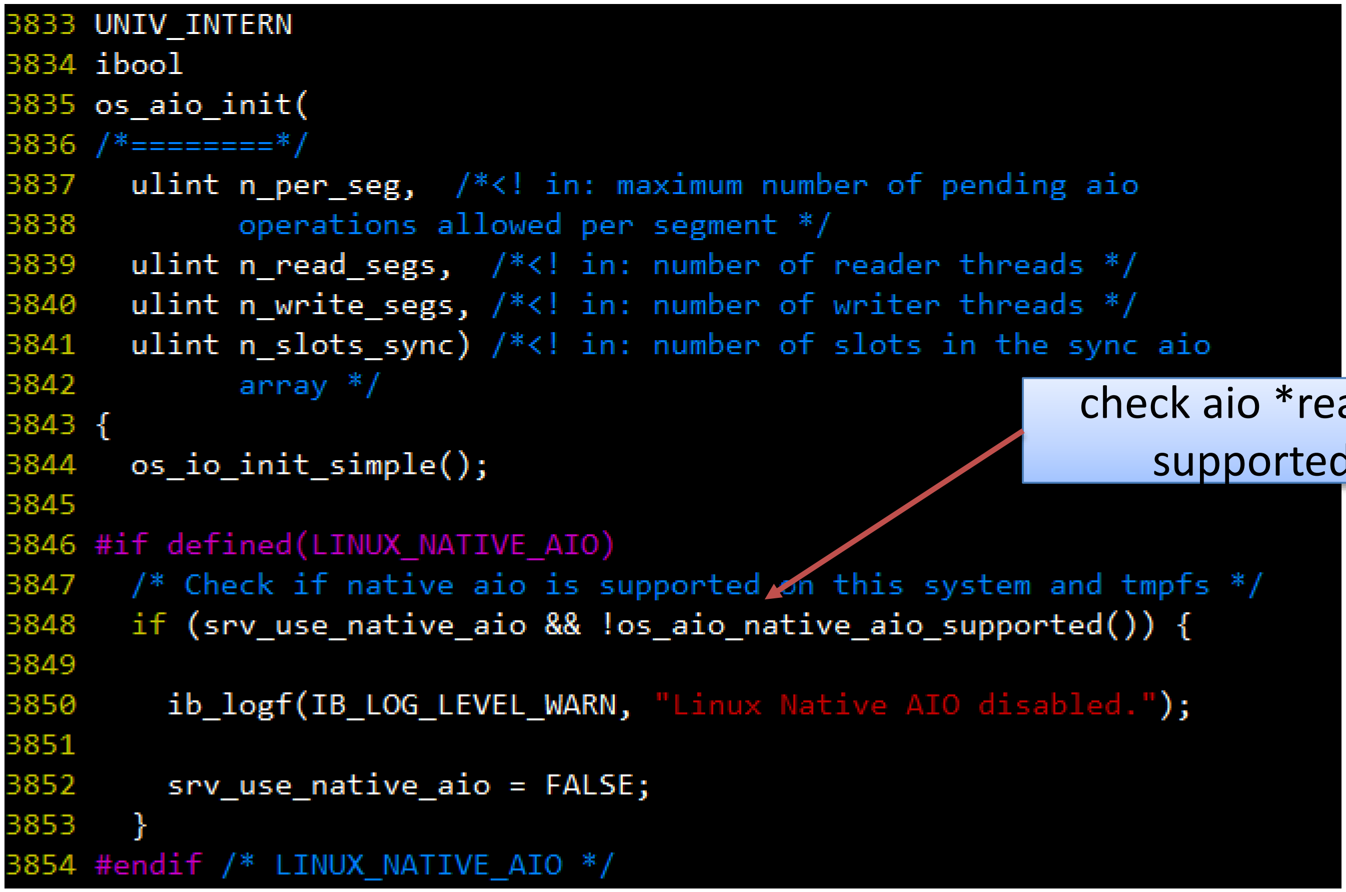
aio system init

- aio thread and array



aio system init

- os/os0file.cc:3835, os_aio_init()

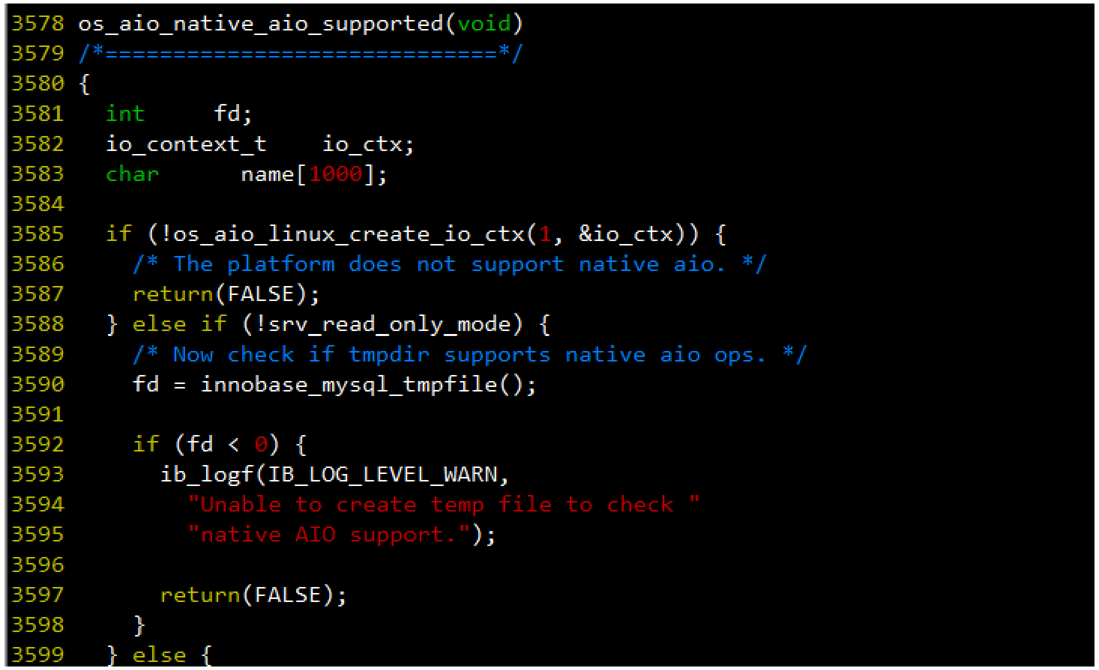


```
3833 UNIV_INTERN
3834 ibool
3835 os_aio_init(
3836 /*=========*/
3837   ulint n_per_seg,  /*<! in: maximum number of pending aio
3838         operations allowed per segment */
3839   ulint n_read_segs,  /*<! in: number of reader threads */
3840   ulint n_write_segs, /*<! in: number of writer threads */
3841   ulint n_slots_sync) /*<! in: number of slots in the sync aio
3842         array */
3843 {
3844   os_io_init_simple();
3845
3846 #if defined(LINUX_NATIVE_AIO)
3847   /* Check if native aio is supported on this system and tmpfs */
3848   if (srv_use_native_aio && !os_aio_native_aio_supported()) {
3849
3850     ib_logf(IB_LOG_LEVEL_WARN, "Linux Native AIO disabled.");
3851
3852     srv_use_native_aio = FALSE;
3853   }
3854 #endif /* LINUX_NATIVE_AIO */
```

check aio *really* supported

aio system init

- `os/os0file.cc:3578`, os_aio_native_aio_supported()
  - make a temp io context
  - and do test read or write

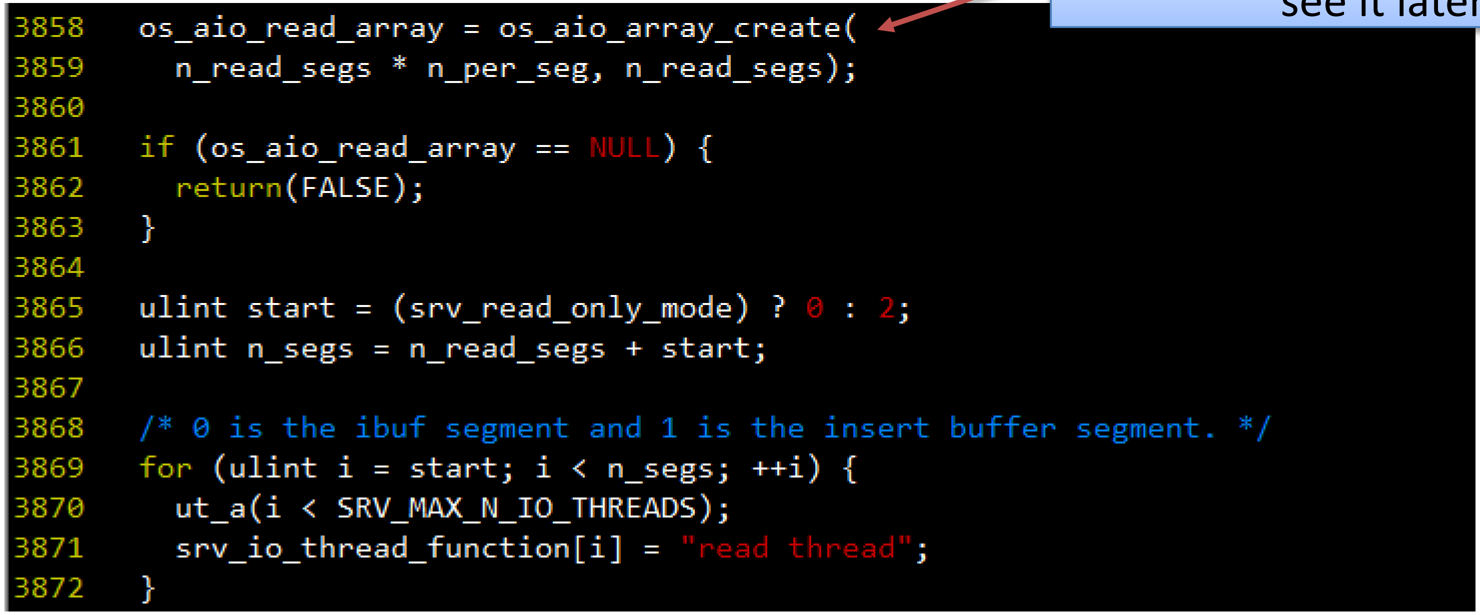


```
3578 os_aio_native_aio_supported(void)
3579 /*==============================*/
3580 {
3581   int      fd;
3582   io_context_t    io_ctx;
3583   char        name[1000];
3584
3585   if (!os_aio_linux_create_io_ctx(1, &io_ctx)) {
3586     /* The platform does not support native aio. */
3587     return(FALSE);
3588   } else if (!srv_read_only_mode) {
3589     /* Now check if tmpdir supports native aio ops. */
3590     fd = innobase_mysql_tmpfile();
3591
3592     if (fd < 0) {
3593       ib_logf(IB_LOG_LEVEL_WARN,
3594         "Unable to create temp file to check "
3595         "native AIO support.");
3596
3597       return(FALSE);
3598     }
3599   } else {
```

aio system init

- `os/os0file.cc:3835, os_aio_init()`
- prepare read io threads

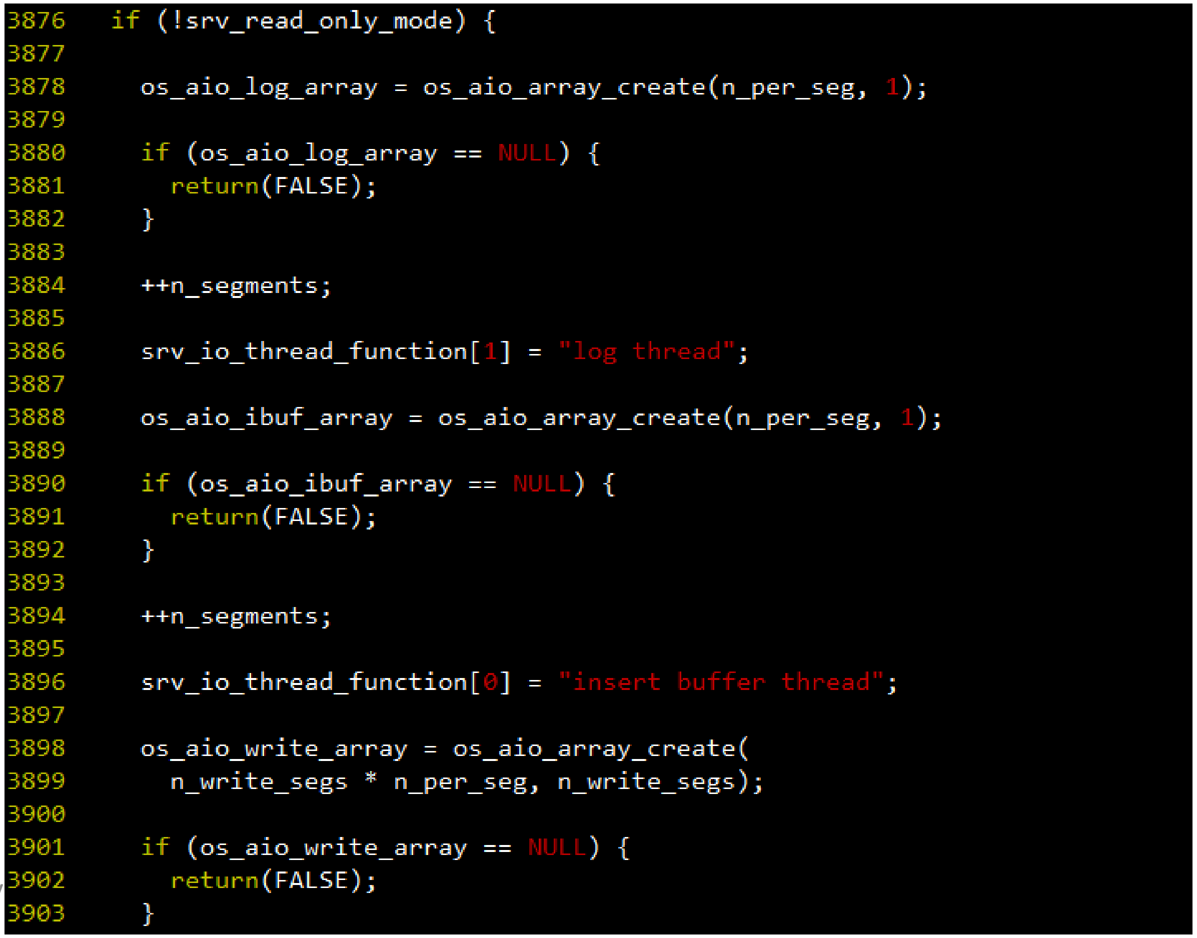
make aio array (kind of io queue)
see it later



```
os_aio_read_array = os_aio_array_create(
  n_read_segs * n_per_seg, n_read_segs);

if (os_aio_read_array == NULL) {
  return(FALSE);
}

ulint start = (srv_read_only_mode) ? 0 : 2;
ulint n_segs = n_read_segs + start;

/* 0 is the ibuf segment and 1 is the insert buffer segment. */
for (ulint i = start; i < n_segs; ++i) {
  ut_a(i < SRV_MAX_N_IO_THREADS);
  srv_io_thread_function[i] = "read thread";
}
```

aio system init

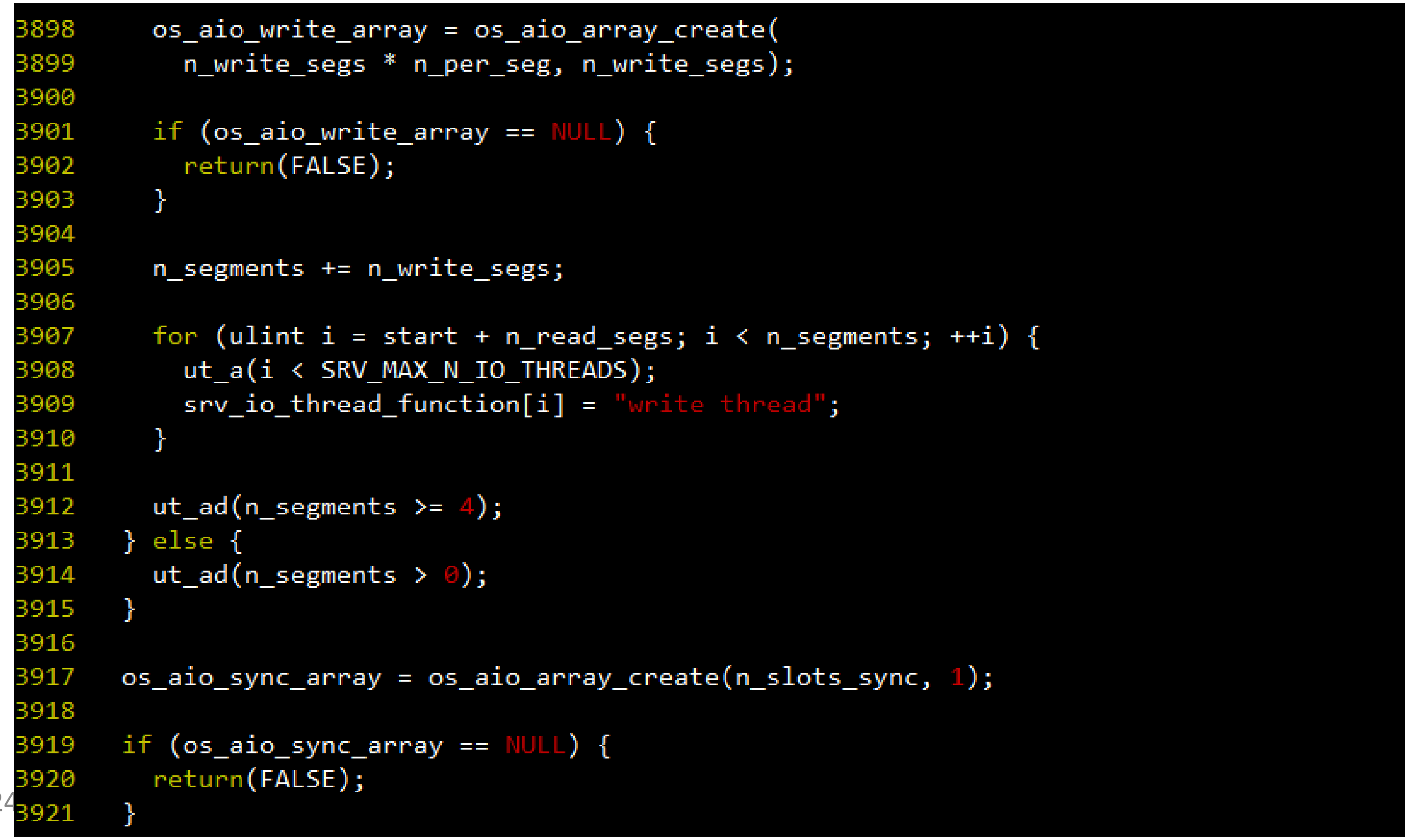
- `os/os0file.cc:3835, os_aio_init()`
- prepare log and ibuf thread



```
if (!srv_read_only_mode) {

  os_aio_log_array = os_aio_array_create(n_per_seg, 1);

  if (os_aio_log_array == NULL) {
    return(FALSE);
  }

  ++n_segments;

  srv_io_thread_function[1] = "log thread";

  os_aio_ibuf_array = os_aio_array_create(n_per_seg, 1);

  if (os_aio_ibuf_array == NULL) {
    return(FALSE);
  }

  ++n_segments;

  srv_io_thread_function[0] = "insert buffer thread";

  os_aio_write_array = os_aio_array_create(
    n_write_segs * n_per_seg, n_write_segs);

  if (os_aio_write_array == NULL) {
    return(FALSE);
  }
```

aio system init

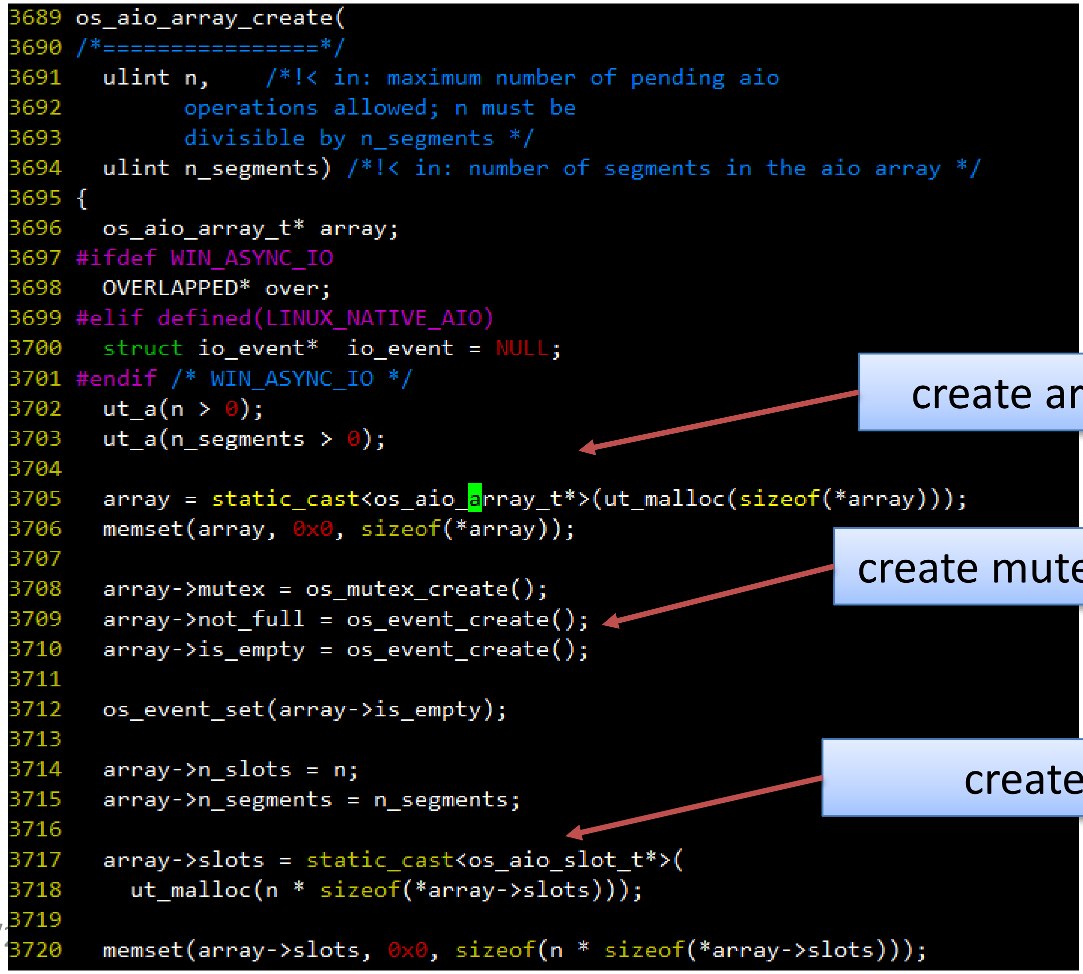
- `os/os0file.cc:3835, os_aio_init()`
- prepare write io thread and sync array



```
    os_aio_write_array = os_aio_array_create(
      n_write_segs * n_per_seg, n_write_segs);

    if (os_aio_write_array == NULL) {
      return(FALSE);
    }

    n_segments += n_write_segs;

    for (ulint i = start + n_read_segs; i < n_segments; ++i) {
      ut_a(i < SRV_MAX_N_IO_THREADS);
      srv_io_thread_function[i] = "write thread";
    }

    ut_ad(n_segments >= 4);
  } else {
    ut_ad(n_segments > 0);
  }

  os_aio_sync_array = os_aio_array_create(n_slots_sync, 1);

  if (os_aio_sync_array == NULL) {
    return(FALSE);
  }
```

aio system init

- os/os0file.cc:3689, create aio array



```
3689 os_aio_array_create(
3690 /*=================*/
3691   ulint n,    /*!< in: maximum number of pending aio
3692         operations allowed; n must be
3693         divisible by n_segments */
3694   ulint n_segments) /*!< in: number of segments in the aio array */
3695 {
3696   os_aio_array_t* array;
3697 #ifdef WIN_ASYNC_IO
3698   OVERLAPPED* over;
3699 #elif defined(LINUX_NATIVE_AIO)
3700   struct io_event*  io_event = NULL;
3701 #endif /* WIN_ASYNC_IO */
3702   ut_a(n > 0);
3703   ut_a(n_segments > 0);
3704
3705   array = static_cast<os_aio_array_t*>(ut_malloc(sizeof(*array)));
3706   memset(array, 0x0, sizeof(*array));
3707
3708   array->mutex = os_mutex_create();
3709   array->not_full = os_event_create();
3710   array->is_empty = os_event_create();
3711
3712   os_event_set(array->is_empty);
3713
3714   array->n_slots = n;
3715   array->n_segments = n_segments;
3716
3717   array->slots = static_cast<os_aio_slot_t*>(
3718     ut_malloc(n * sizeof(*array->slots)));
3719
3720   memset(array->slots, 0x0, sizeof(n * sizeof(*array->slots)));
```

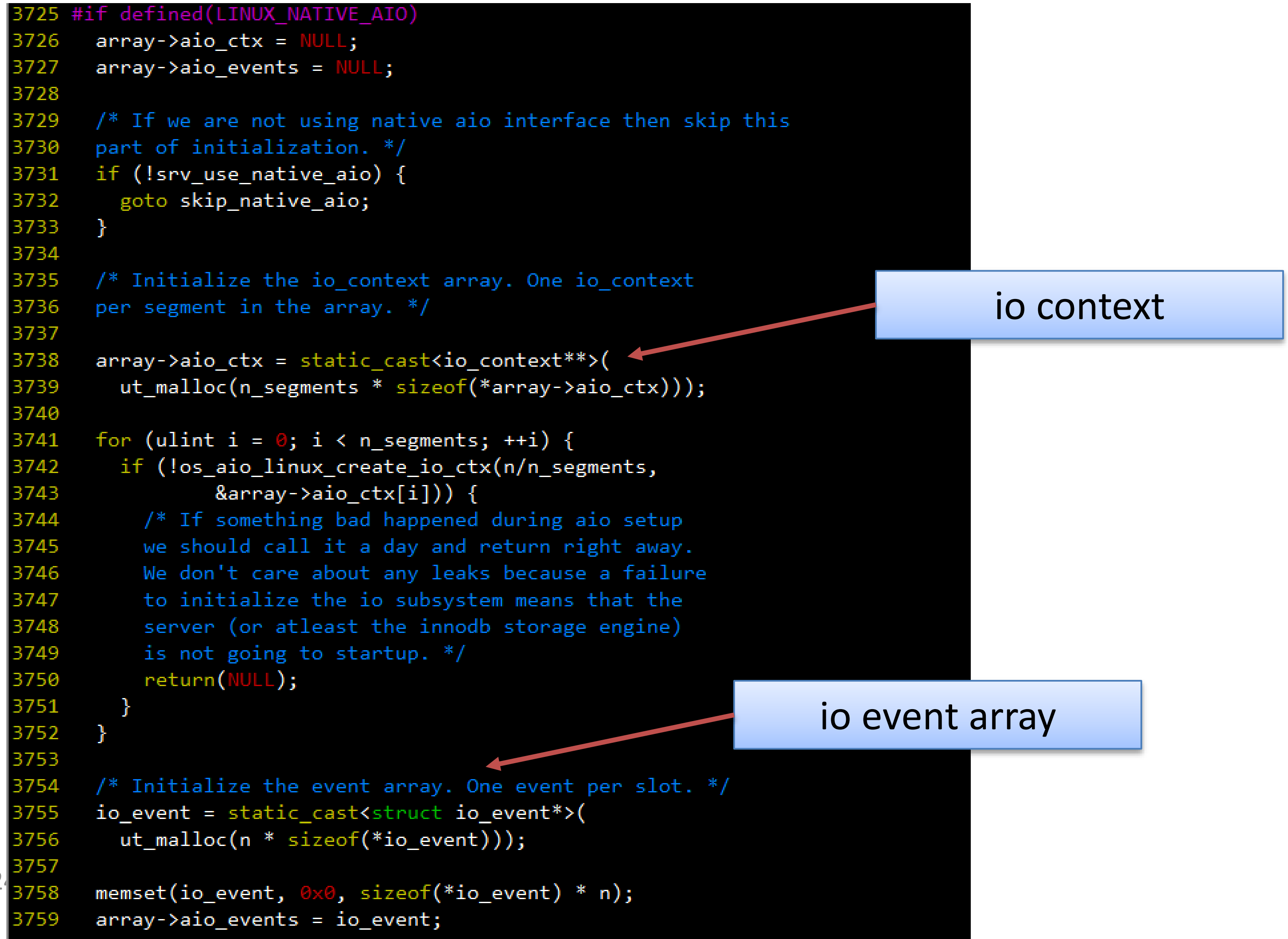
create array struct

create mutex for the array

create slots

aio system init

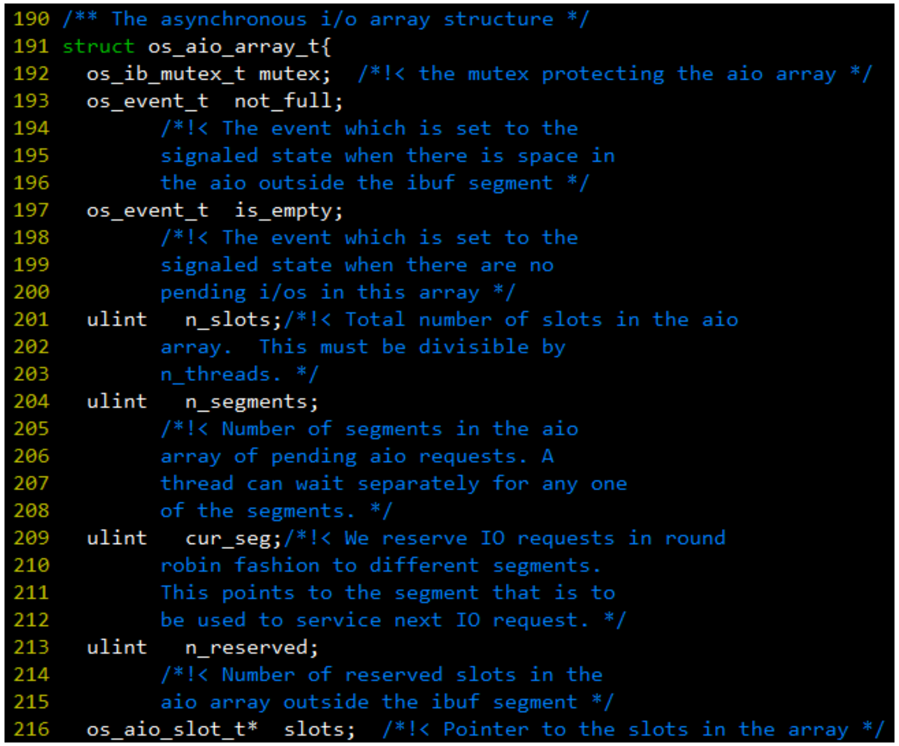
- `os/os0file.cc:3689, create aio array`



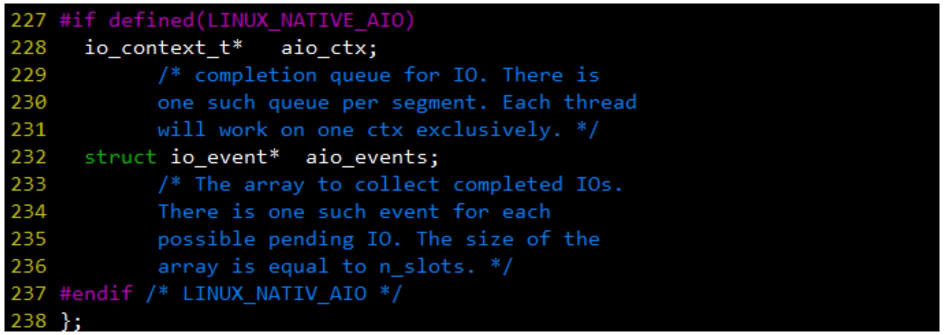
```
#if defined(LINUX_NATIVE_AIO)
  array->aio_ctx = NULL;
  array->aio_events = NULL;

  /* If we are not using native aio interface then skip this
  part of initialization. */
  if (!srv_use_native_aio) {
    goto skip_native_aio;
  }

  /* Initialize the io_context array. One io_context
  per segment in the array. */

  array->aio_ctx = static_cast<io_context**>(
    ut_malloc(n_segments * sizeof(*array->aio_ctx)));

  for (ulint i = 0; i < n_segments; ++i) {
    if (!os_aio_linux_create_io_ctx(n/n_segments,
            &array->aio_ctx[i])) {
      /* If something bad happened during aio setup
      we should call it a day and return right away.
      We don't care about any leaks because a failure
      to initialize the io subsystem means that the
      server (or atleast the innodb storage engine)
      is not going to startup. */
      return(NULL);
    }
  }

  /* Initialize the event array. One event per slot. */
  io_event = static_cast<struct io_event*>(
    ut_malloc(n * sizeof(*io_event)));

  memset(io_event, 0x0, sizeof(*io_event) * n);
  array->aio_events = io_event;
```

aio system init

- os_aio_array_t



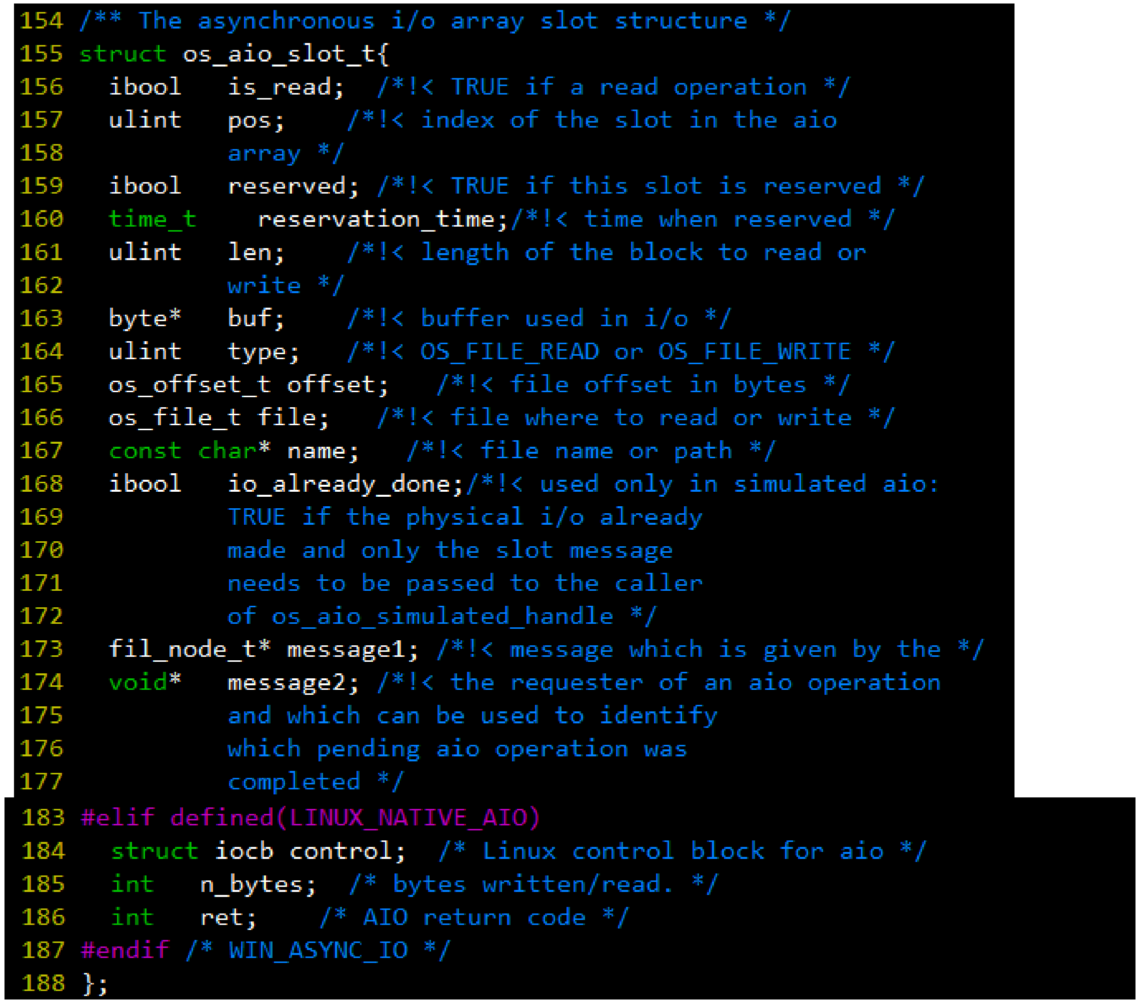
```
/** The asynchronous i/o array structure */
struct os_aio_array_t{
  os_ib_mutex_t mutex;  /*!< the mutex protecting the aio array */
  os_event_t  not_full;
        /*!< The event which is set to the
        signaled state when there is space in
        the aio outside the ibuf segment */
  os_event_t  is_empty;
        /*!< The event which is set to the
        signaled state when there are no
        pending i/os in this array */
  ulint   n_slots;/*!< Total number of slots in the aio
        array.  This must be divisible by
        n_threads. */
  ulint   n_segments;
        /*!< Number of segments in the aio
        array of pending aio requests. A
        thread can wait separately for any one
        of the segments. */
  ulint   cur_seg;/*!< We reserve IO requests in round
        robin fashion to different segments.
        This points to the segment that is to
        be used to service next IO request. */
  ulint   n_reserved;
        /*!< Number of reserved slots in the
        aio array outside the ibuf segment */
  os_aio_slot_t*  slots;  /*!< Pointer to the slots in the array */
```



```
#if defined(LINUX_NATIVE_AIO)
  io_context_t*   aio_ctx;
        /* completion queue for IO. There is
        one such queue per segment. Each thread
        will work on one ctx exclusively. */
  struct io_event*  aio_events;
        /* The array to collect completed IOs.
        There is one such event for each
        possible pending IO. The size of the
        array is equal to n_slots. */
#endif /* LINUX_NATIV_AIO */
};
```

aio system init

- os_aio_slot_t

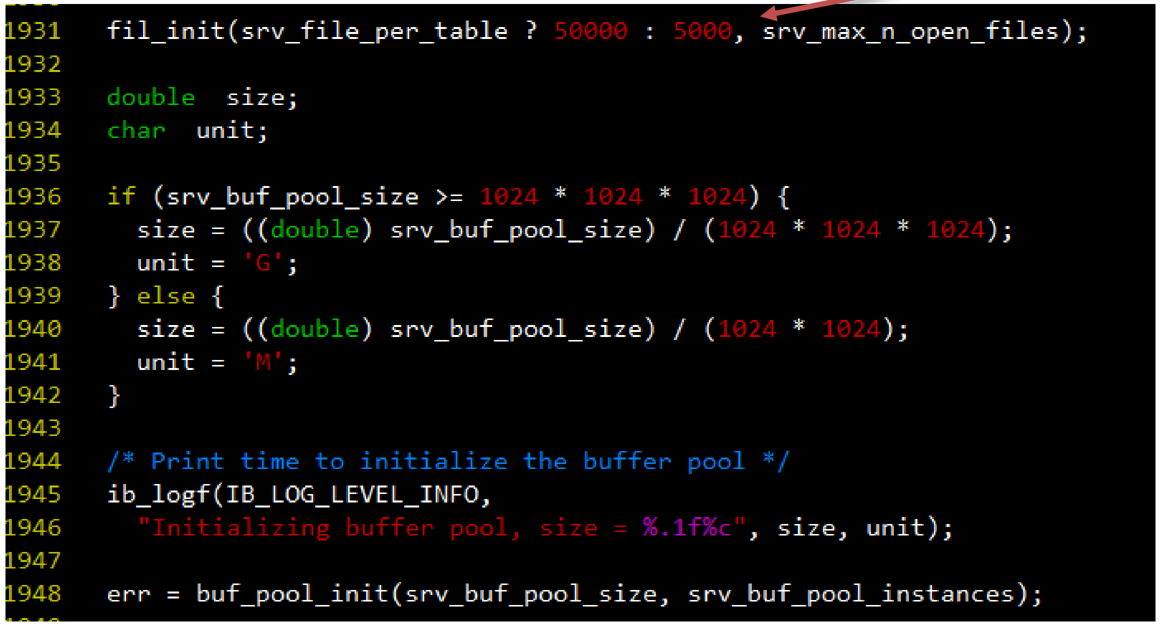


```
/** The asynchronous i/o array slot structure */
struct os_aio_slot_t{
  ibool   is_read;  /*!< TRUE if a read operation */
  ulint   pos;    /*!< index of the slot in the aio
          array */
  ibool   reserved; /*!< TRUE if this slot is reserved */
  time_t    reservation_time;/*!< time when reserved */
  ulint   len;    /*!< length of the block to read or
          write */
  byte*   buf;    /*!< buffer used in i/o */
  ulint   type;   /*!< OS_FILE_READ or OS_FILE_WRITE */
  os_offset_t offset;   /*!< file offset in bytes */
  os_file_t file;   /*!< file where to read or write */
  const char* name;   /*!< file name or path */
  ibool   io_already_done;/*!< used only in simulated aio:
          TRUE if the physical i/o already
          made and only the slot message
          needs to be passed to the caller
          of os_aio_simulated_handle */
  fil_node_t* message1; /*!< message which is given by the */
  void*   message2; /*!< the requester of an aio operation
          and which can be used to identify
          which pending aio operation was
          completed */
#elif defined(LINUX_NATIVE_AIO)
  struct iocb control;  /* Linux control block for aio */
  int   n_bytes;  /* bytes written/read. */
  int   ret;    /* AIO return code */
#endif /* WIN_ASYNC_IO */
};
```

InnoDB Initialize

- file system init
- `srv0start.cc:1931, fil_init()`

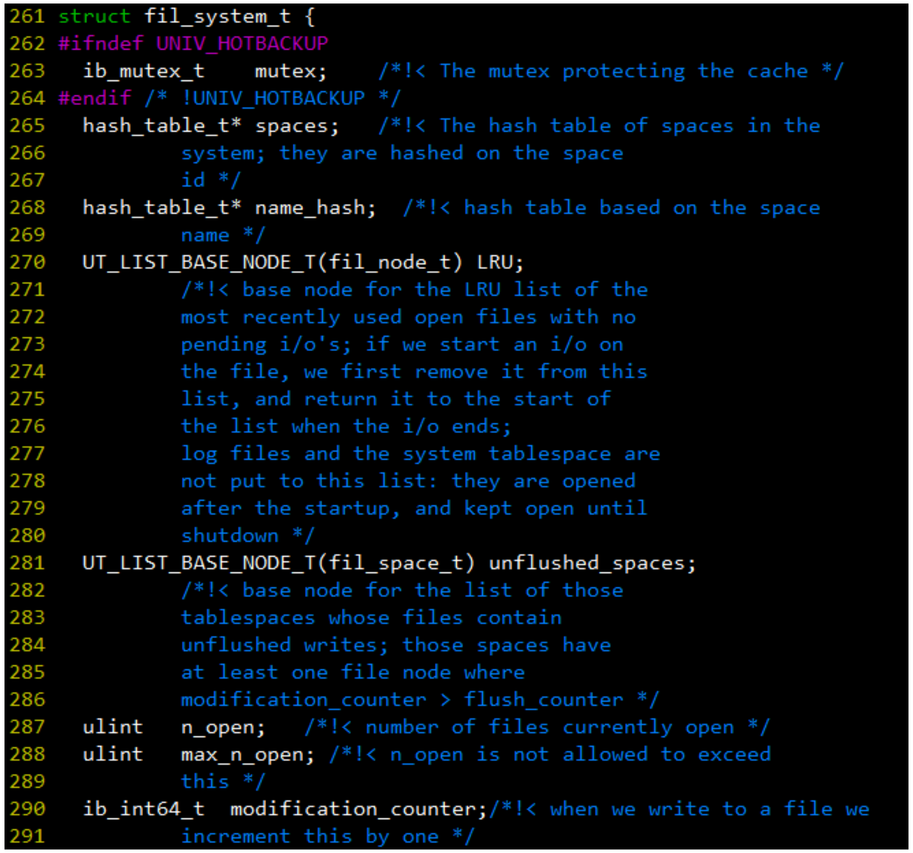
invoke file system init



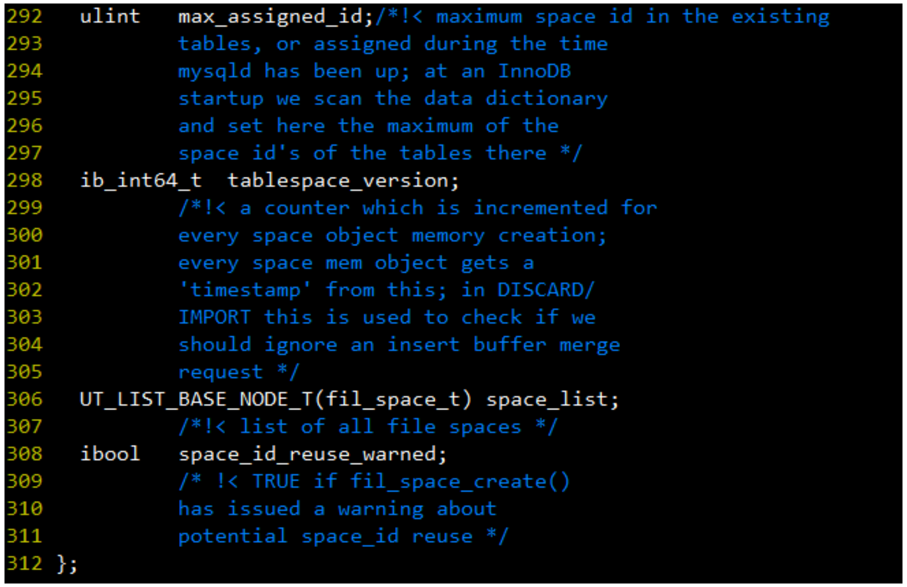
```
1931    fil_init(srv_file_per_table ? 50000 : 5000, srv_max_n_open_files);
1932
1933    double  size;
1934    char  unit;
1935
1936    if (srv_buf_pool_size >= 1024 * 1024 * 1024) {
1937      size = ((double) srv_buf_pool_size) / (1024 * 1024 * 1024);
1938      unit = 'G';
1939    } else {
1940      size = ((double) srv_buf_pool_size) / (1024 * 1024);
1941      unit = 'M';
1942    }
1943
1944    /* Print time to initialize the buffer pool */
1945    ib_logf(IB_LOG_LEVEL_INFO,
1946      "Initializing buffer pool, size = %.1f%c", size, unit);
1947
1948    err = buf_pool_init(srv_buf_pool_size, srv_buf_pool_instances);
```

file system init

- fil_system_t



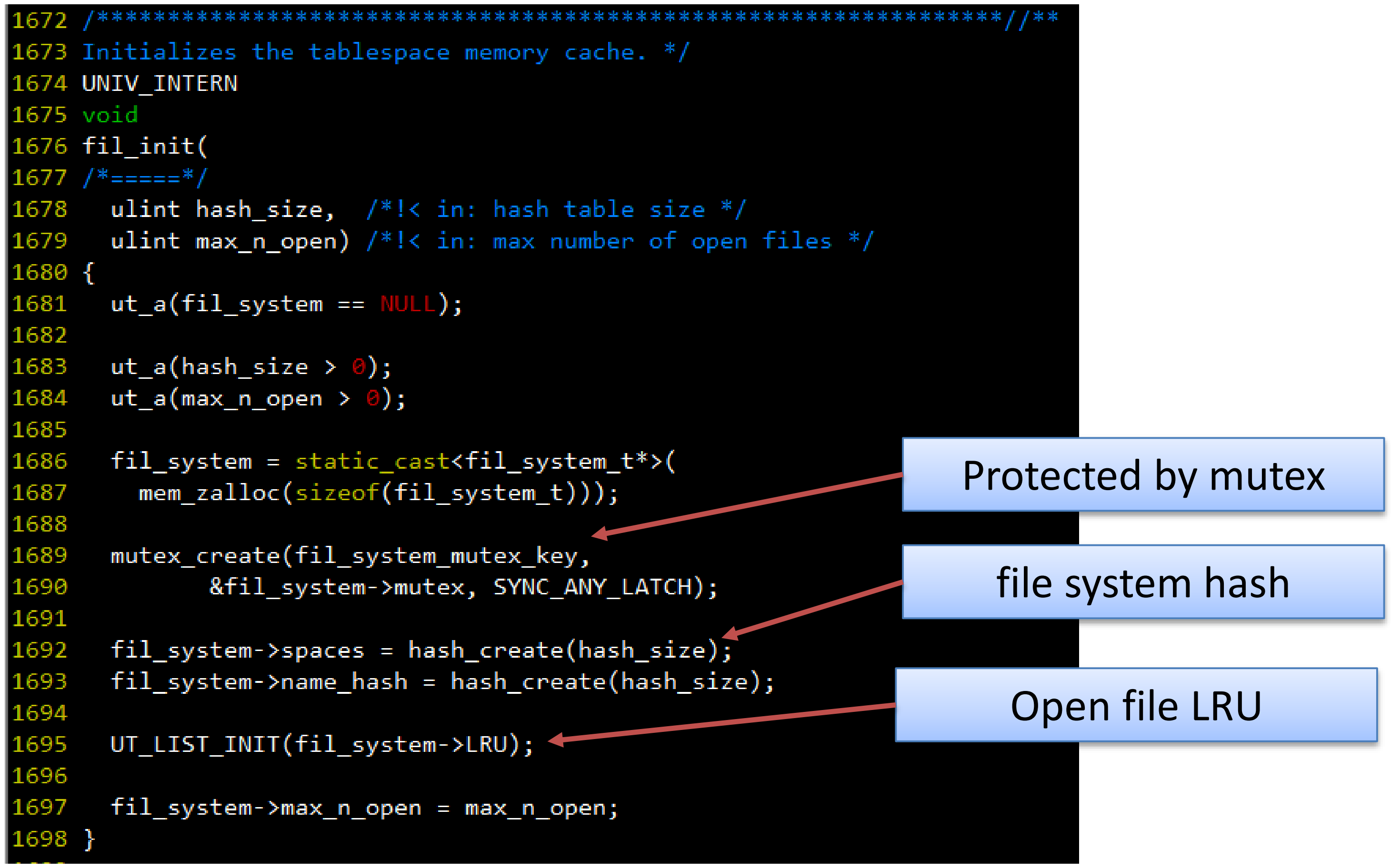
```
struct fil_system_t {
#ifndef UNIV_HOTBACKUP
	ib_mutex_t	mutex;		/*!< The mutex protecting the cache */
#endif /* !UNIV_HOTBACKUP */
	hash_table_t*	spaces;		/*!< The hash table of spaces in the
					system; they are hashed on the space
					id */
	hash_table_t*	name_hash;	/*!< hash table based on the space
					name */
	UT_LIST_BASE_NODE_T(fil_node_t) LRU;
					/*!< base node for the LRU list of the
					most recently used open files with no
					pending i/o's; if we start an i/o on
					the file, we first remove it from this
					list, and return it to the start of
					the list when the i/o ends;
					log files and the system tablespace are
					not put to this list: they are opened
					after the startup, and kept open until
					shutdown */
	UT_LIST_BASE_NODE_T(fil_space_t) unflushed_spaces;
					/*!< base node for the list of those
					tablespaces whose files contain
					unflushed writes; those spaces have
					at least one file node where
					modification_counter > flush_counter */
	ulint		n_open;		/*!< number of files currently open */
	ulint		max_n_open;	/*!< n_open is not allowed to exceed
					this */
	ib_int64_t	modification_counter;/*!< when we write to a file we
					increment this by one */
```



```
	ulint		max_assigned_id;/*!< maximum space id in the existing
					tables, or assigned during the time
					mysqld has been up; at an InnoDB
					startup we scan the data dictionary
					and set here the maximum of the
					space id's of the tables there */
	ib_int64_t	tablespace_version;
					/*!< a counter which is incremented for
					every space object memory creation;
					every space mem object gets a
					'timestamp' from this; in DISCARD/
					IMPORT this is used to check if we
					should ignore an insert buffer merge
					request */
	UT_LIST_BASE_NODE_T(fil_space_t) space_list;
					/*!< list of all file spaces */
	ibool		space_id_reuse_warned;
					/* !< TRUE if fil_space_create()
					has issued a warning about
					potential space_id reuse */
};
```

file system init

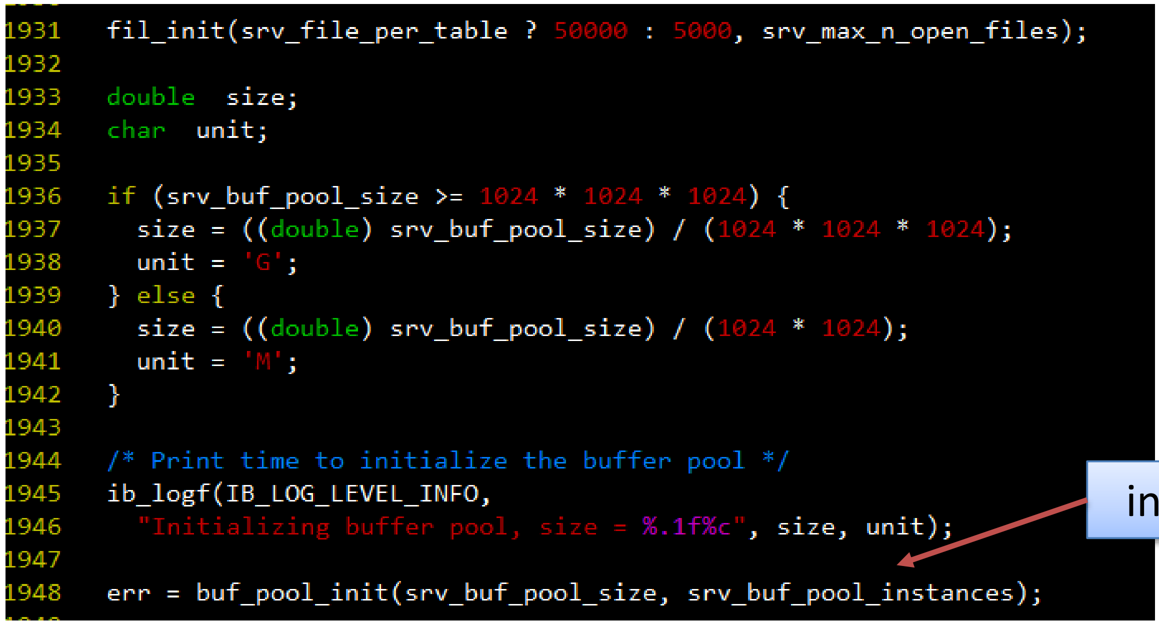
- `fil/fil0fil.cc:1676, fil_init()`



```
1672 /********************************************************************//**
1673 Initializes the tablespace memory cache. */
1674 UNIV_INTERN
1675 void
1676 fil_init(
1677 /*=====*/
1678   ulint hash_size,  /*!< in: hash table size */
1679   ulint max_n_open) /*!< in: max number of open files */
1680 {
1681   ut_a(fil_system == NULL);
1682 
1683   ut_a(hash_size > 0);
1684   ut_a(max_n_open > 0);
1685 
1686   fil_system = static_cast<fil_system_t*>(
1687     mem_zalloc(sizeof(fil_system_t)));
1688 
1689   mutex_create(fil_system_mutex_key,
1690          &fil_system->mutex, SYNC_ANY_LATCH);
1691 
1692   fil_system->spaces = hash_create(hash_size);
1693   fil_system->name_hash = hash_create(hash_size);
1694 
1695   UT_LIST_INIT(fil_system->LRU);
1696 
1697   fil_system->max_n_open = max_n_open;
1698 }
```

InnoDB Initialize

- Buffer Pool Init
- `srv0start.cc:1948, buf_pool_init()`

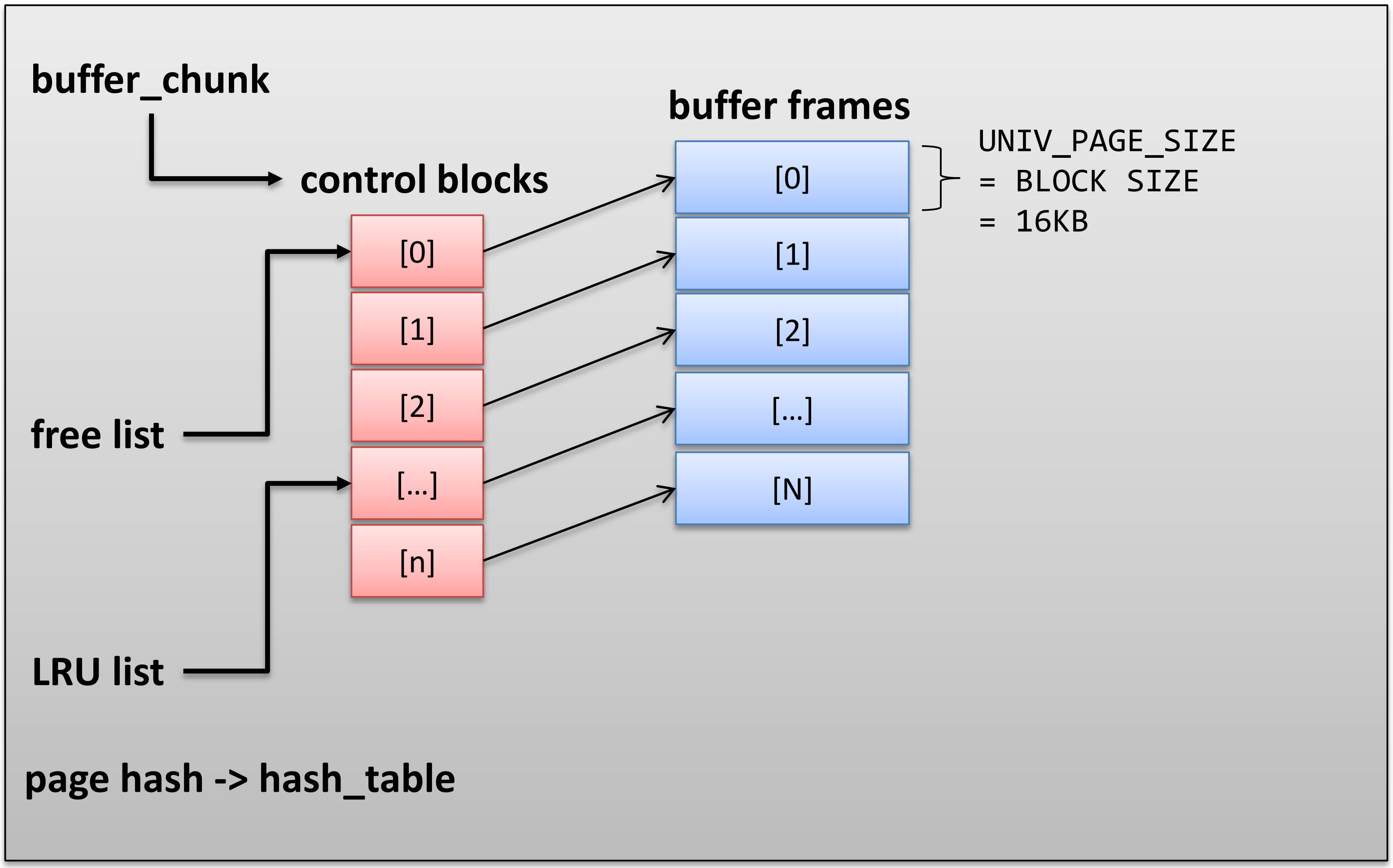


```
1931   fil_init(srv_file_per_table ? 50000 : 5000, srv_max_n_open_files);
1932
1933   double  size;
1934   char  unit;
1935
1936   if (srv_buf_pool_size >= 1024 * 1024 * 1024) {
1937     size = ((double) srv_buf_pool_size) / (1024 * 1024 * 1024);
1938     unit = 'G';
1939   } else {
1940     size = ((double) srv_buf_pool_size) / (1024 * 1024);
1941     unit = 'M';
1942   }
1943
1944   /* Print time to initialize the buffer pool */
1945   ib_logf(IB_LOG_LEVEL_INFO,
1946     "Initializing buffer pool, size = %.1f%c", size, unit);
1947
1948   err = buf_pool_init(srv_buf_pool_size, srv_buf_pool_instances);
```

invoke buffer pool init

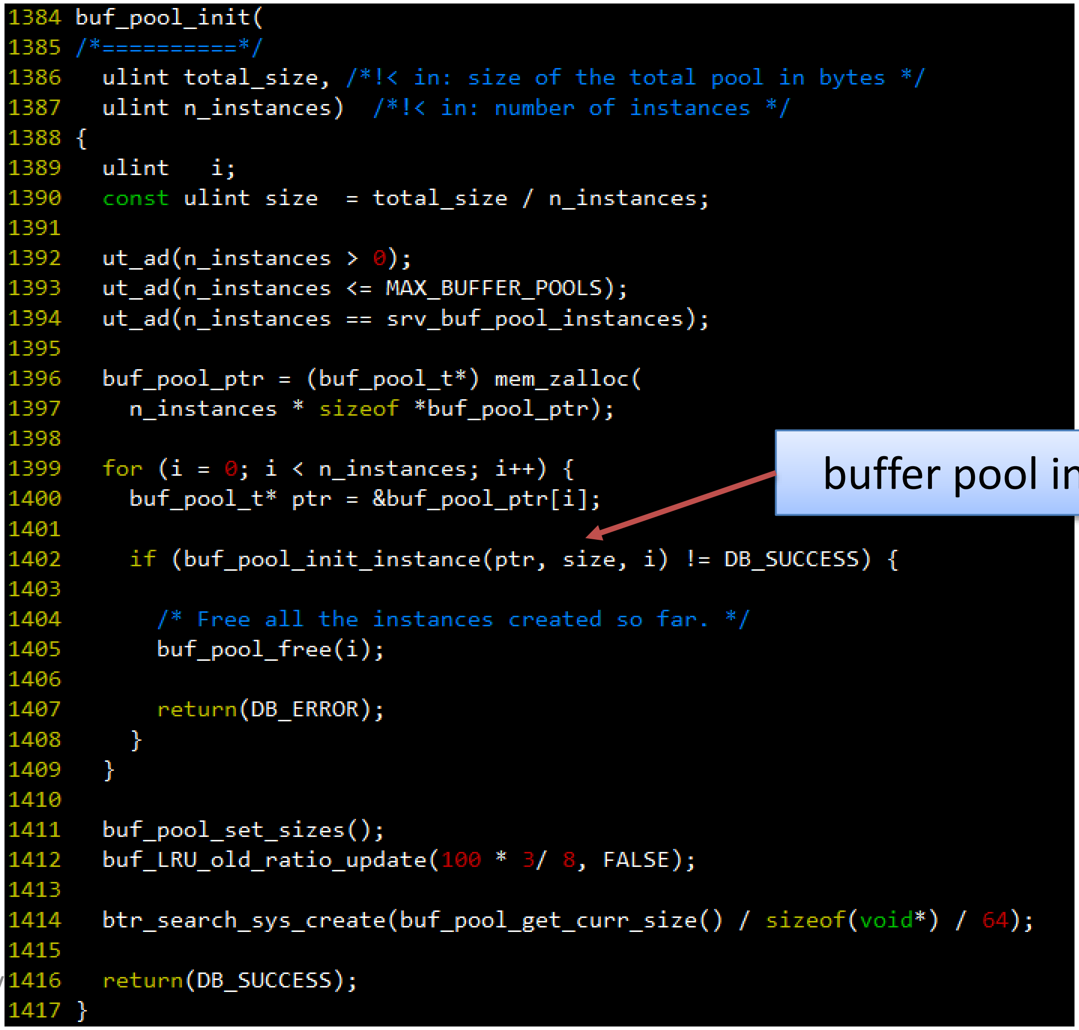
Buffer pool init

buffer_pool_ptr[]



Buffer pool init

- buf/buf0buf.cc:1384, buf_pool_init()

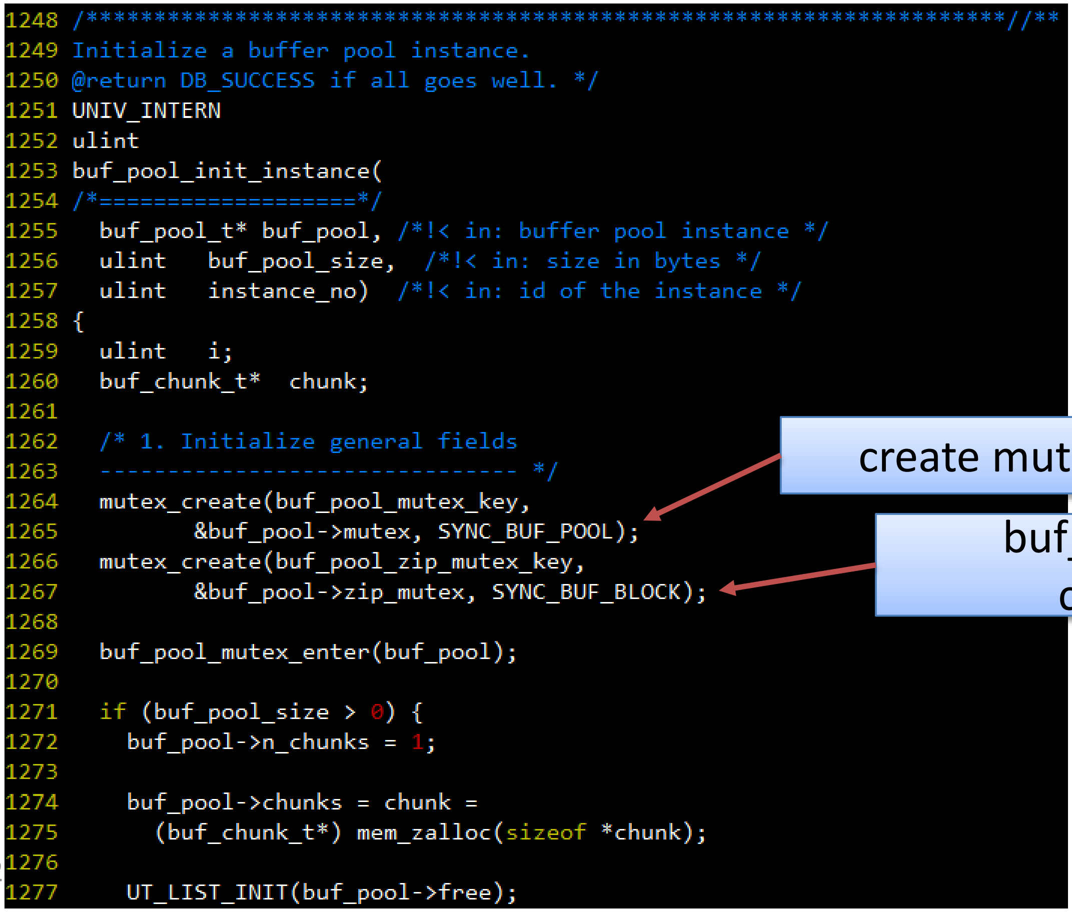


```
1384 buf_pool_init(
1385 /*==========*/
1386   ulint total_size, /*!< in: size of the total pool in bytes */
1387   ulint n_instances)  /*!< in: number of instances */
1388 {
1389   ulint   i;
1390   const ulint size  = total_size / n_instances;
1391
1392   ut_ad(n_instances > 0);
1393   ut_ad(n_instances <= MAX_BUFFER_POOLS);
1394   ut_ad(n_instances == srv_buf_pool_instances);
1395
1396   buf_pool_ptr = (buf_pool_t*) mem_zalloc(
1397     n_instances * sizeof *buf_pool_ptr);
1398
1399   for (i = 0; i < n_instances; i++) {
1400     buf_pool_t* ptr = &buf_pool_ptr[i];
1401
1402     if (buf_pool_init_instance(ptr, size, i) != DB_SUCCESS) {
1403
1404       /* Free all the instances created so far. */
1405       buf_pool_free(i);
1406
1407       return(DB_ERROR);
1408     }
1409   }
1410
1411   buf_pool_set_sizes();
1412   buf_LRU_old_ratio_update(100 * 3/ 8, FALSE);
1413
1414   btr_search_sys_create(buf_pool_get_curr_size() / sizeof(void*) / 64);
1415
1416   return(DB_SUCCESS);
1417 }
```

buffer pool init per instance

Buffer pool init

- `buf/buf0buf.cc:1253, buf_pool_instance()`



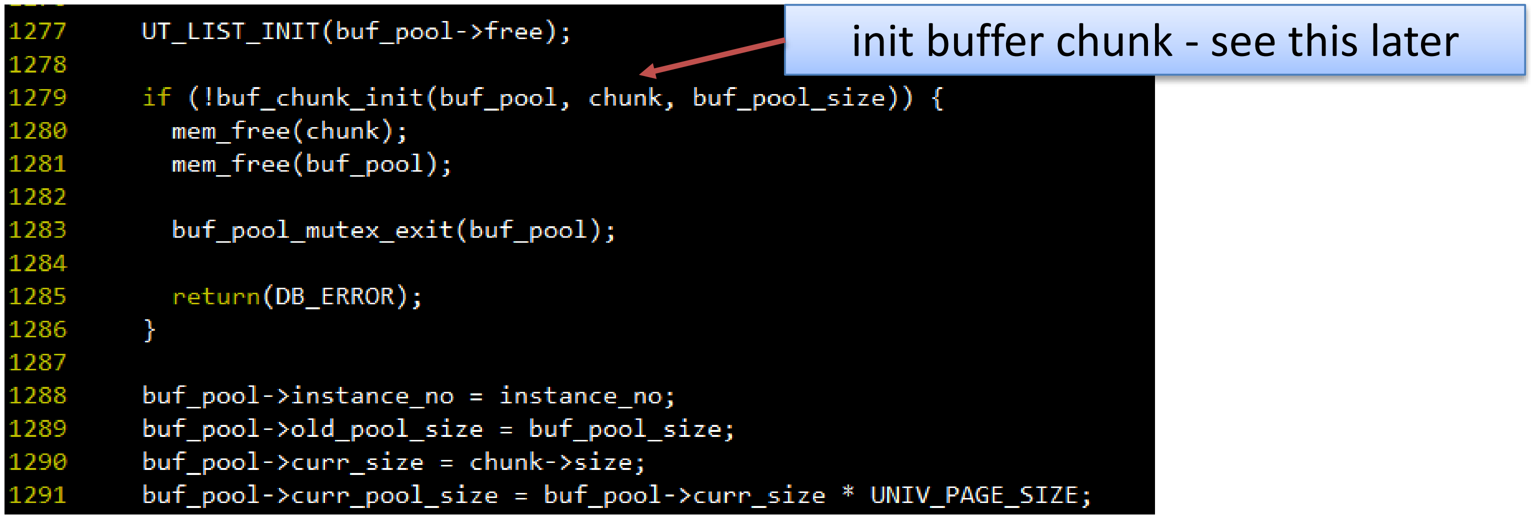
```
1248 /********************************************************************//**
1249 Initialize a buffer pool instance.
1250 @return DB_SUCCESS if all goes well. */
1251 UNIV_INTERN
1252 ulint
1253 buf_pool_init_instance(
1254 /*===================*/
1255   buf_pool_t* buf_pool, /*!< in: buffer pool instance */
1256   ulint   buf_pool_size,  /*!< in: size in bytes */
1257   ulint   instance_no)  /*!< in: id of the instance */
1258 {
1259   ulint   i;
1260   buf_chunk_t*  chunk;
1261 
1262   /* 1. Initialize general fields
1263   ------------------------------- */
1264   mutex_create(buf_pool_mutex_key,
1265          &buf_pool->mutex, SYNC_BUF_POOL);
1266   mutex_create(buf_pool_zip_mutex_key,
1267          &buf_pool->zip_mutex, SYNC_BUF_BLOCK);
1268 
1269   buf_pool_mutex_enter(buf_pool);
1270 
1271   if (buf_pool_size > 0) {
1272     buf_pool->n_chunks = 1;
1273 
1274     buf_pool->chunks = chunk =
1275       (buf_chunk_t*) mem_zalloc(sizeof *chunk);
1276 
1277     UT_LIST_INIT(buf_pool->free);
```

create mutex and enter

buf_pool_zip : use compression

Buffer pool init

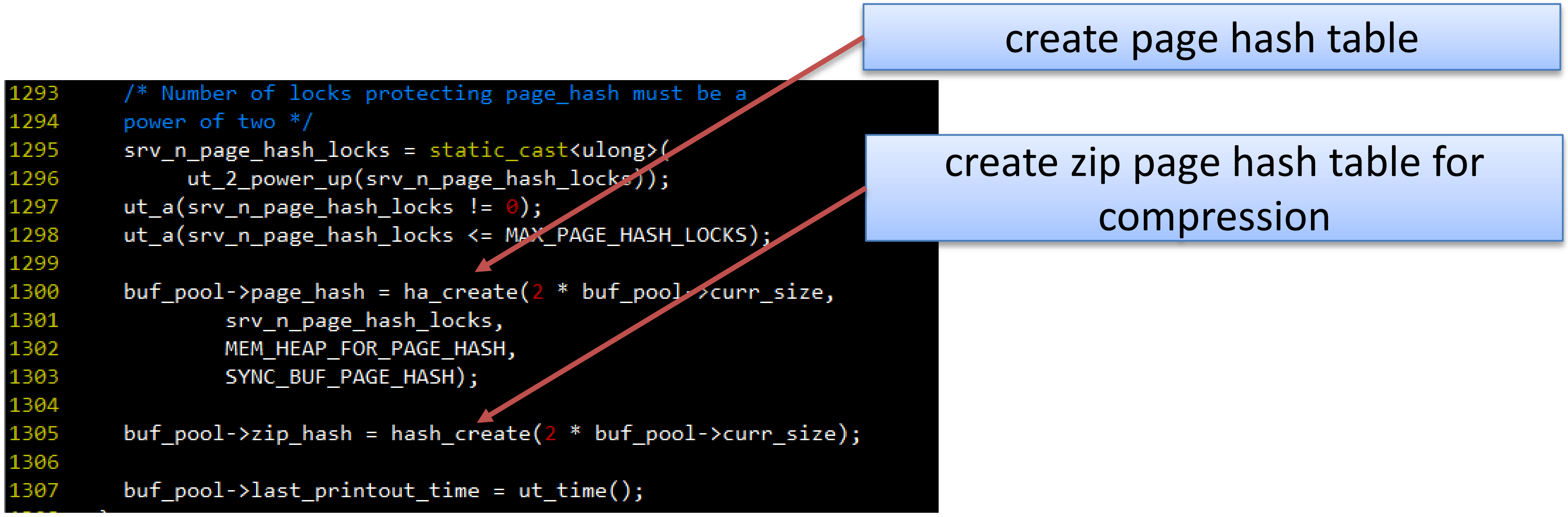
- `buf/buf0buf.cc:1253, buf_pool_instance()`



```
UT_LIST_INIT(buf_pool->free);

if (!buf_chunk_init(buf_pool, chunk, buf_pool_size)) {
  mem_free(chunk);
  mem_free(buf_pool);

  buf_pool_mutex_exit(buf_pool);

  return(DB_ERROR);
}

buf_pool->instance_no = instance_no;
buf_pool->old_pool_size = buf_pool_size;
buf_pool->curr_size = chunk->size;
buf_pool->curr_pool_size = buf_pool->curr_size * UNIV_PAGE_SIZE;
```

Buffer pool init

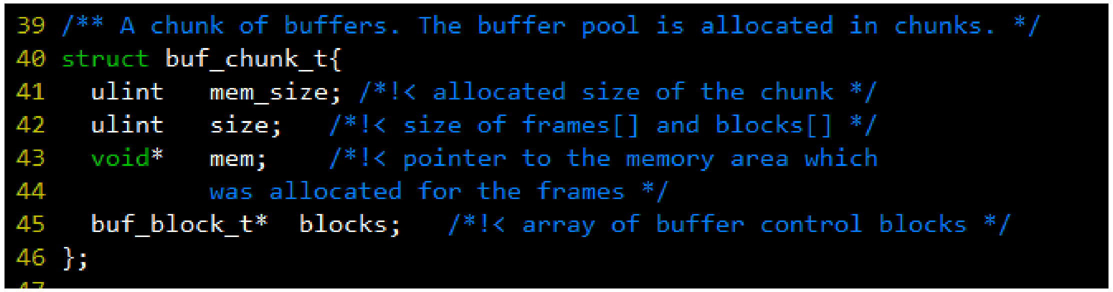
- `buf/buf0buf.cc:1253, buf_pool_instance()`



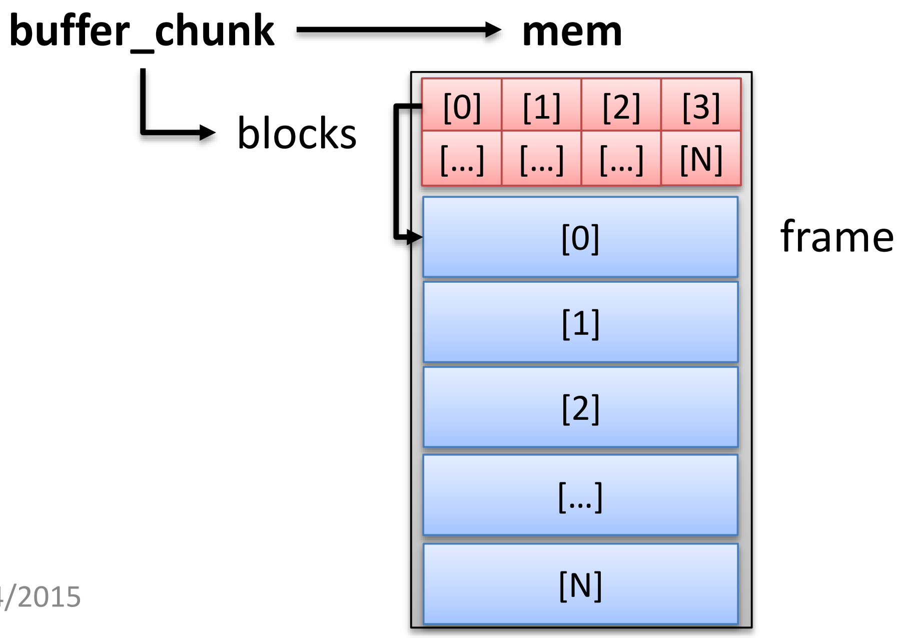
```
	/* Number of locks protecting page_hash must be a
	power of two */
	srv_n_page_hash_locks = static_cast<ulong>(
		ut_2_power_up(srv_n_page_hash_locks));
	ut_a(srv_n_page_hash_locks != 0);
	ut_a(srv_n_page_hash_locks <= MAX_PAGE_HASH_LOCKS);

	buf_pool->page_hash = ha_create(2 * buf_pool->curr_size,
			srv_n_page_hash_locks,
			MEM_HEAP_FOR_PAGE_HASH,
			SYNC_BUF_PAGE_HASH);

	buf_pool->zip_hash = hash_create(2 * buf_pool->curr_size);

	buf_pool->last_printout_time = ut_time();
```

Buffer pool init

- buffer chunk

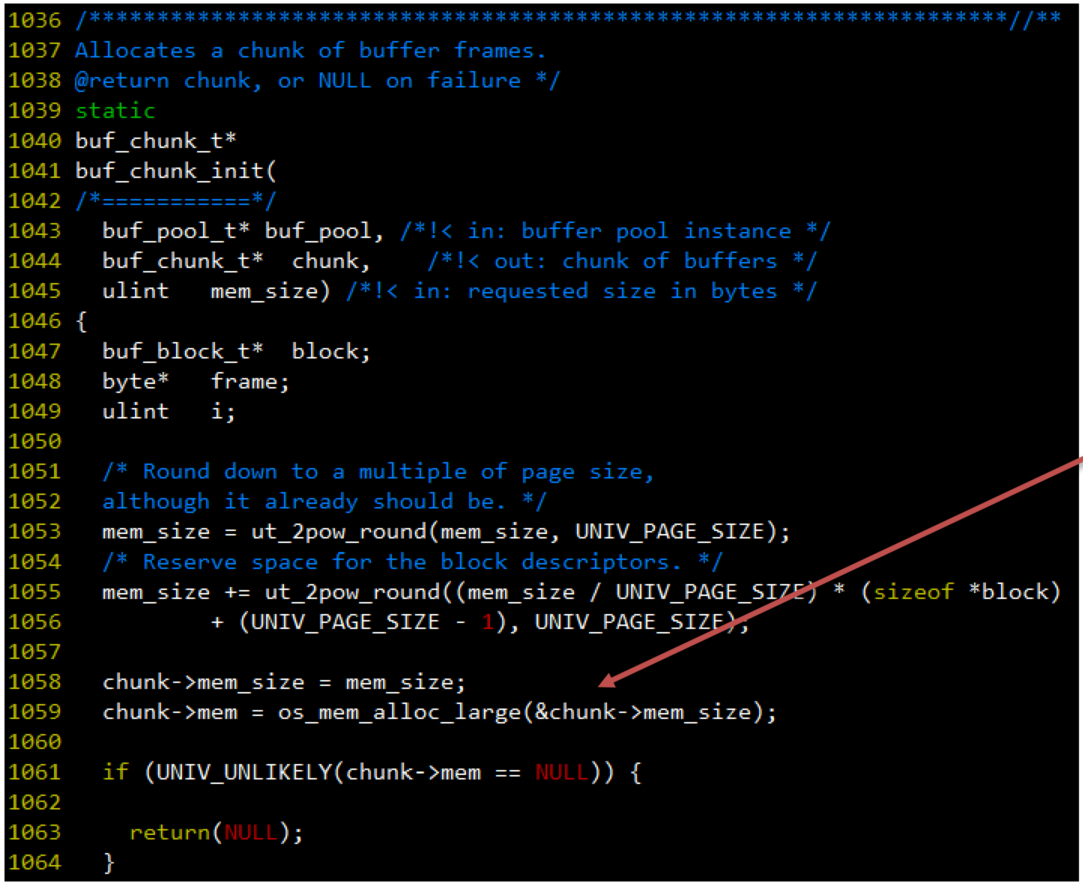


```
/** A chunk of buffers. The buffer pool is allocated in chunks. */
struct buf_chunk_t{
  ulint    mem_size; /*!< allocated size of the chunk */
  ulint    size;    /*!< size of frames[] and blocks[] */
  void*    mem;     /*!< pointer to the memory area which
           was allocated for the frames */
  buf_block_t*  blocks;    /*!< array of buffer control blocks */
};
```



Buffer pool init

- buf/buf0buf.cc:1041, chunk_init()

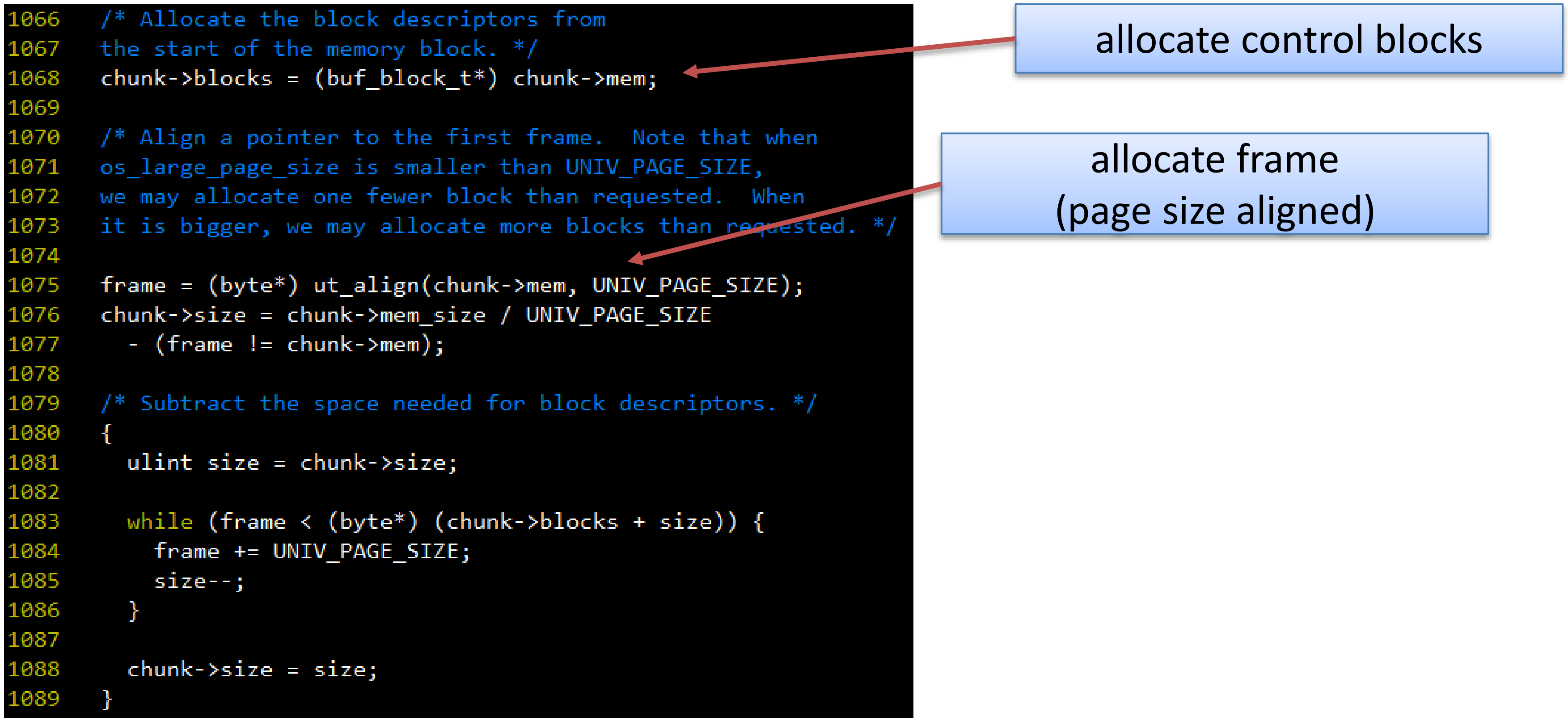


```
1036 /*******************************************************************//**
1037 Allocates a chunk of buffer frames.
1038 @return chunk, or NULL on failure */
1039 static
1040 buf_chunk_t*
1041 buf_chunk_init(
1042 /*===========*/
1043   buf_pool_t* buf_pool, /*!< in: buffer pool instance */
1044   buf_chunk_t*  chunk,    /*!< out: chunk of buffers */
1045   ulint   mem_size) /*!< in: requested size in bytes */
1046 {
1047   buf_block_t*  block;
1048   byte*   frame;
1049   ulint   i;
1050 
1051   /* Round down to a multiple of page size,
1052   although it already should be. */
1053   mem_size = ut_2pow_round(mem_size, UNIV_PAGE_SIZE);
1054   /* Reserve space for the block descriptors. */
1055   mem_size += ut_2pow_round((mem_size / UNIV_PAGE_SIZE) * (sizeof *block)
1056           + (UNIV_PAGE_SIZE - 1), UNIV_PAGE_SIZE);
1057 
1058   chunk->mem_size = mem_size;
1059   chunk->mem = os_mem_alloc_large(&chunk->mem_size);
1060 
1061   if (UNIV_UNLIKELY(chunk->mem == NULL)) {
1062 
1063     return(NULL);
1064   }
```

allocate chunk mem
(blocks + frames)

Buffer pool init

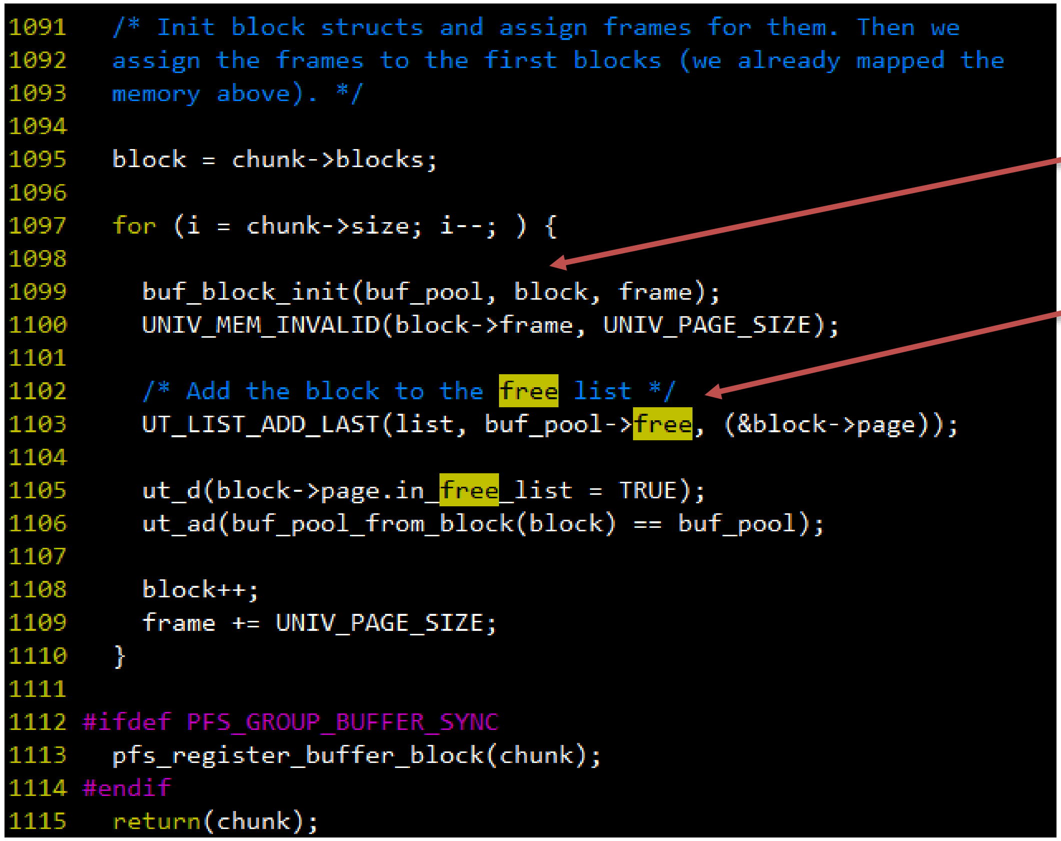
- `buf/buf0buf.cc:1041, chunk_init()`



```
	/* Allocate the block descriptors from
	the start of the memory block. */
	chunk->blocks = (buf_block_t*) chunk->mem;

	/* Align a pointer to the first frame.  Note that when
	os_large_page_size is smaller than UNIV_PAGE_SIZE,
	we may allocate one fewer block than requested.  When
	it is bigger, we may allocate more blocks than requested. */

	frame = (byte*) ut_align(chunk->mem, UNIV_PAGE_SIZE);
	chunk->size = chunk->mem_size / UNIV_PAGE_SIZE
		- (frame != chunk->mem);

	/* Subtract the space needed for block descriptors. */
	{
		ulint	size = chunk->size;

		while (frame < (byte*) (chunk->blocks + size)) {
			frame += UNIV_PAGE_SIZE;
			size--;
		}

		chunk->size = size;
	}
```

Buffer pool init

- buf/buf0buf.cc:1041, chunk_init()



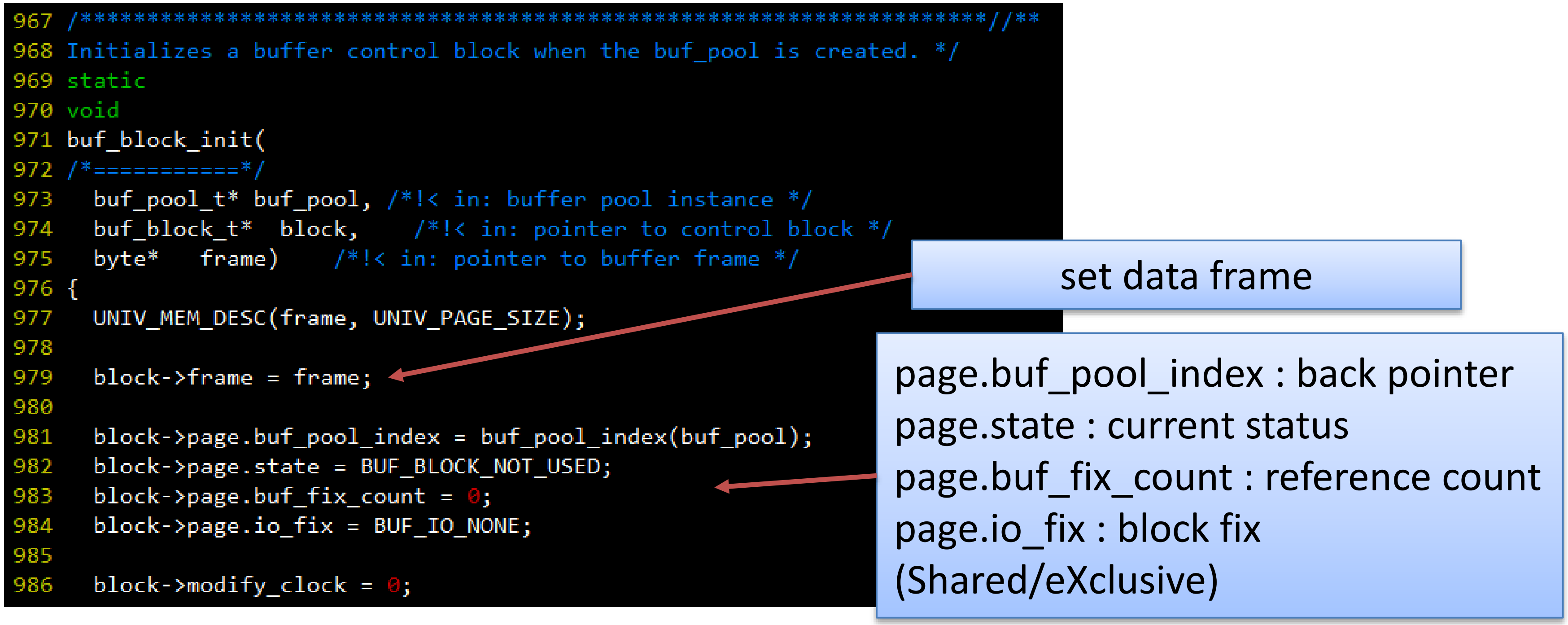
```
  /* Init block structs and assign frames for them. Then we
  assign the frames to the first blocks (we already mapped the
  memory above). */

  block = chunk->blocks;

  for (i = chunk->size; i--; ) {

    buf_block_init(buf_pool, block, frame);
    UNIV_MEM_INVALID(block->frame, UNIV_PAGE_SIZE);

    /* Add the block to the free list */
    UT_LIST_ADD_LAST(list, buf_pool->free, (&block->page));

    ut_d(block->page.in_free_list = TRUE);
    ut_ad(buf_pool_from_block(block) == buf_pool);

    block++;
    frame += UNIV_PAGE_SIZE;
  }

#ifdef PFS_GROUP_BUFFER_SYNC
  pfs_register_buffer_block(chunk);
#endif
  return(chunk);
```

init control block

add all blocks to free list

Buffer pool init

- `buf/buf0buf.cc:971, buf_block_init()`



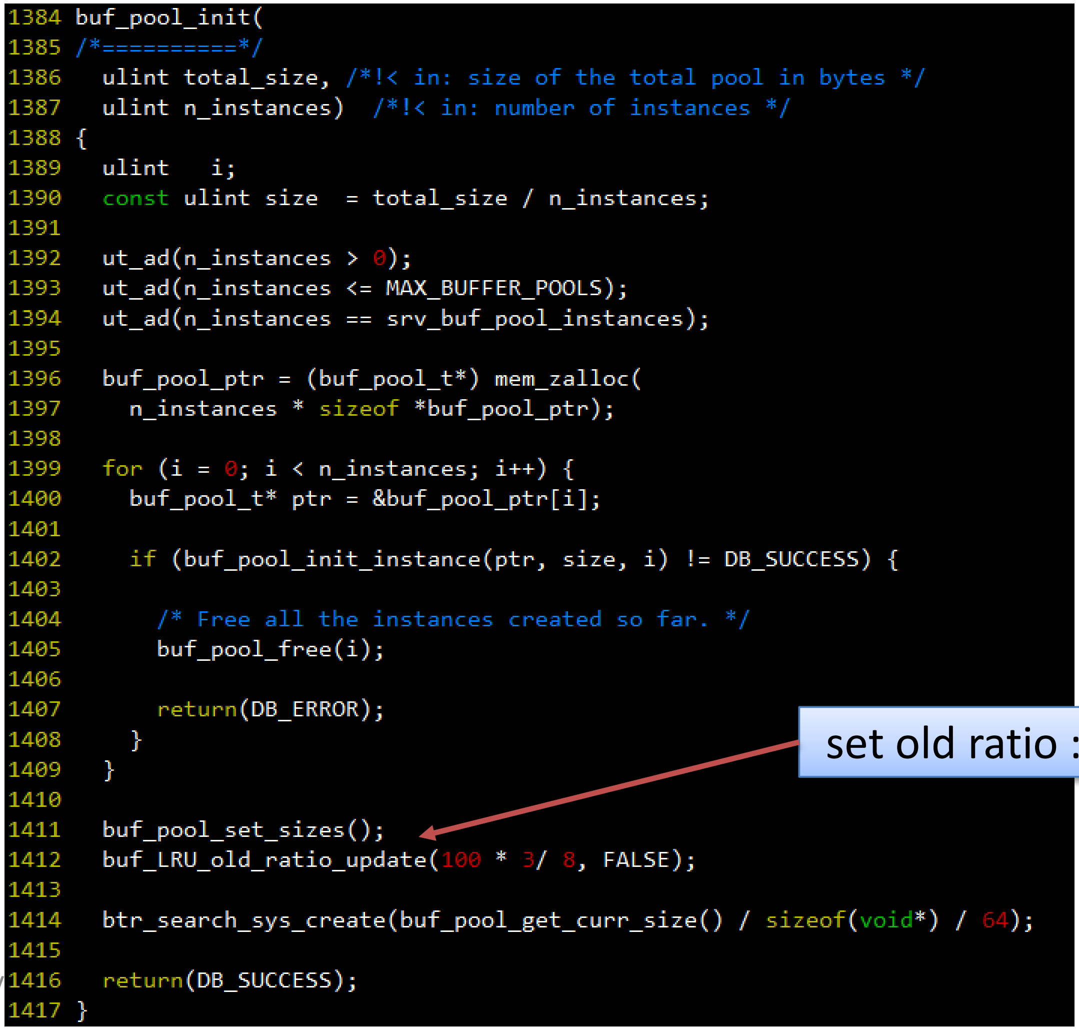
```
/*********************************************************************//**
Initializes a buffer control block when the buf_pool is created. */
static
void
buf_block_init(
/*===========*/
  buf_pool_t* buf_pool, /*!< in: buffer pool instance */
  buf_block_t*  block,     /*!< in: pointer to control block */
  byte*    frame)     /*!< in: pointer to buffer frame */
{
  UNIV_MEM_DESC(frame, UNIV_PAGE_SIZE);

  block->frame = frame;

  block->page.buf_pool_index = buf_pool_index(buf_pool);
  block->page.state = BUF_BLOCK_NOT_USED;
  block->page.buf_fix_count = 0;
  block->page.io_fix = BUF_IO_NONE;

  block->modify_clock = 0;
```

set data frame

page.buf_pool_index : back pointer
page.state : current status
page.buf_fix_count : reference count
page.io_fix : block fix
(Shared/eXclusive)

Buffer pool init

- buf/buf0buf.cc:1384, buf_pool_init()

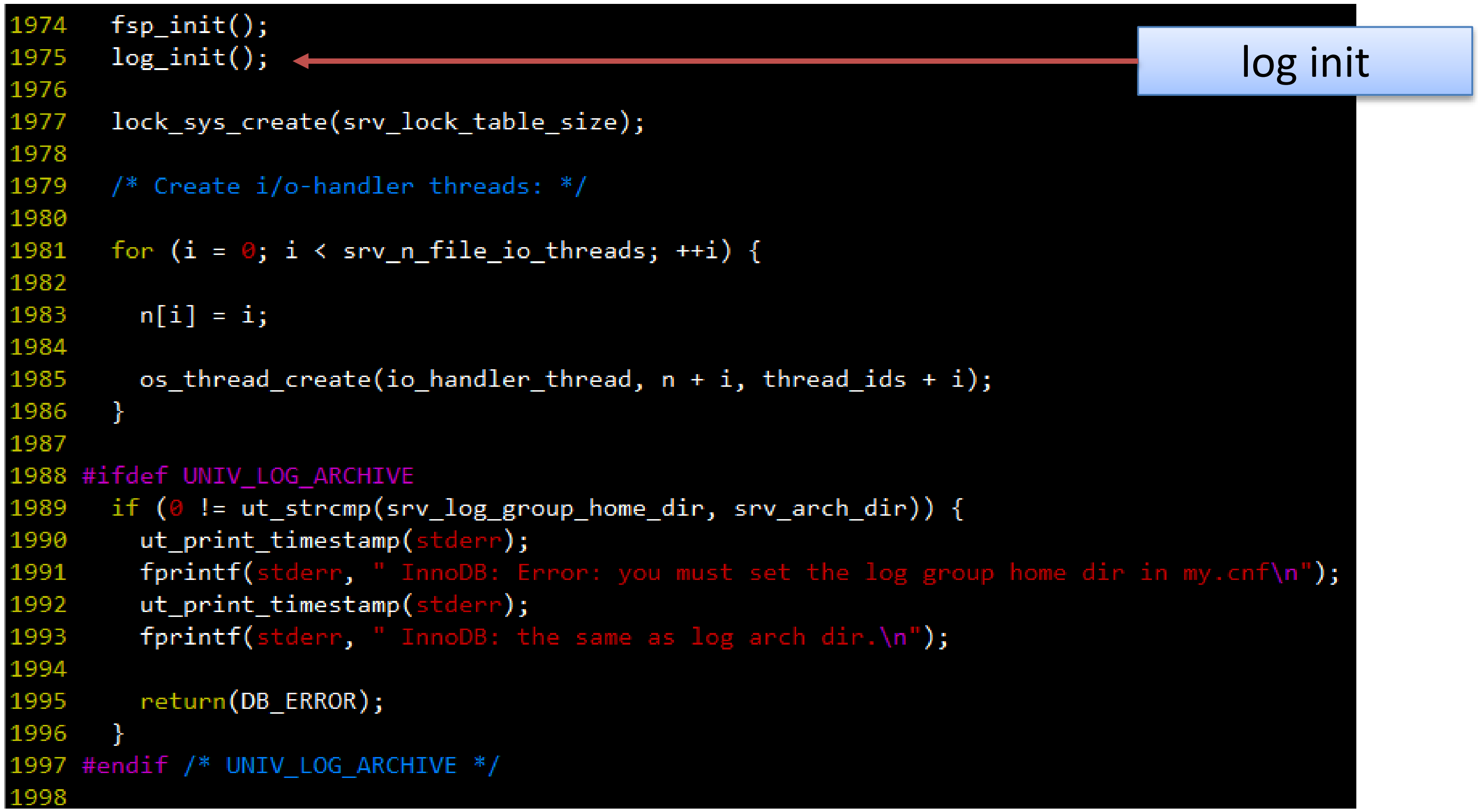


```
1384 buf_pool_init(
1385 /*==========*/
1386   ulint total_size, /*!< in: size of the total pool in bytes */
1387   ulint n_instances)  /*!< in: number of instances */
1388 {
1389   ulint   i;
1390   const ulint size  = total_size / n_instances;
1391 
1392   ut_ad(n_instances > 0);
1393   ut_ad(n_instances <= MAX_BUFFER_POOLS);
1394   ut_ad(n_instances == srv_buf_pool_instances);
1395 
1396   buf_pool_ptr = (buf_pool_t*) mem_zalloc(
1397     n_instances * sizeof *buf_pool_ptr);
1398 
1399   for (i = 0; i < n_instances; i++) {
1400     buf_pool_t* ptr = &buf_pool_ptr[i];
1401 
1402     if (buf_pool_init_instance(ptr, size, i) != DB_SUCCESS) {
1403 
1404       /* Free all the instances created so far. */
1405       buf_pool_free(i);
1406 
1407       return(DB_ERROR);
1408     }
1409   }
1410 
1411   buf_pool_set_sizes();
1412   buf_LRU_old_ratio_update(100 * 3/ 8, FALSE);
1413 
1414   btr_search_sys_create(buf_pool_get_curr_size() / sizeof(void*) / 64);
1415 
1416   return(DB_SUCCESS);
1417 }
```

set old ratio : 3/8 -> for mid-point insertion

log init

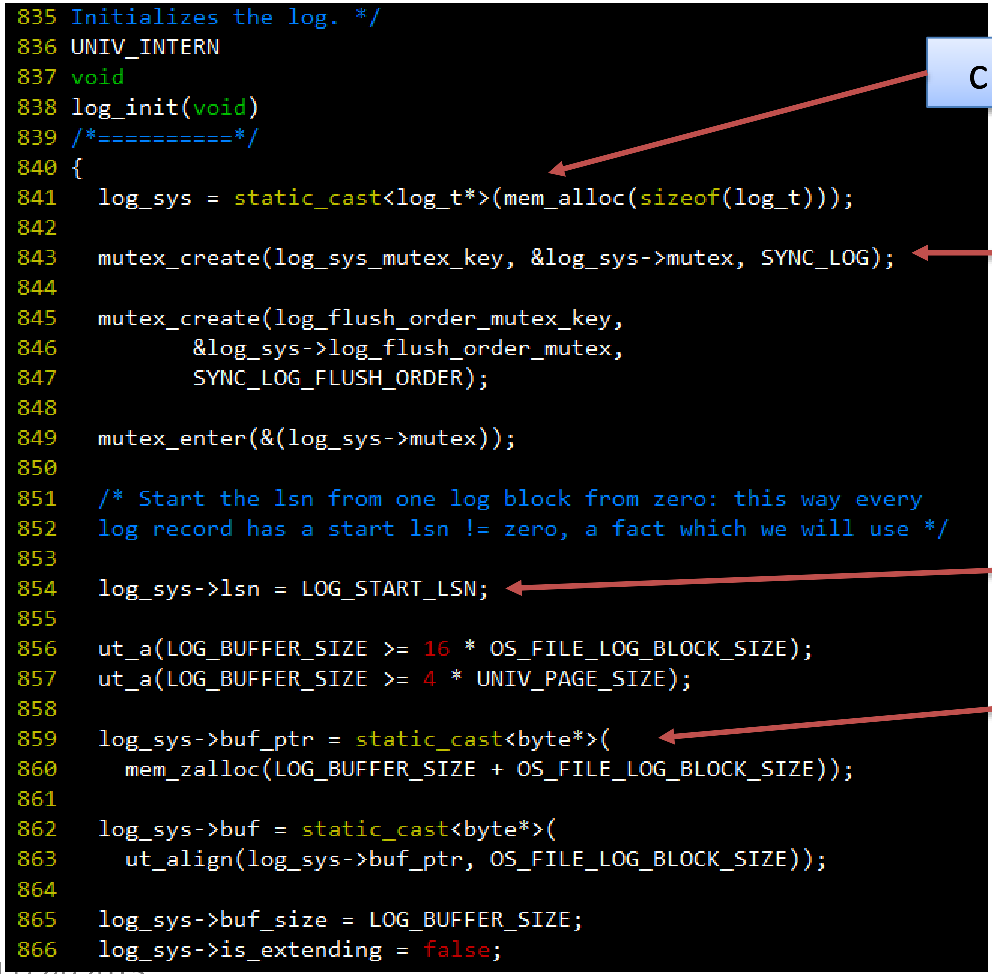
- srv/srv0start:1975, log_init()



```
1974   fsp_init();
1975   log_init();
1976
1977   lock_sys_create(srv_lock_table_size);
1978
1979   /* Create i/o-handler threads: */
1980
1981   for (i = 0; i < srv_n_file_io_threads; ++i) {
1982
1983     n[i] = i;
1984
1985     os_thread_create(io_handler_thread, n + i, thread_ids + i);
1986   }
1987
1988 #ifdef UNIV_LOG_ARCHIVE
1989   if (0 != ut_strcmp(srv_log_group_home_dir, srv_arch_dir)) {
1990     ut_print_timestamp(stderr);
1991     fprintf(stderr, " InnoDB: Error: you must set the log group home dir in my.cnf\n");
1992     ut_print_timestamp(stderr);
1993     fprintf(stderr, " InnoDB: the same as log arch dir.\n");
1994
1995     return(DB_ERROR);
1996   }
1997 #endif /* UNIV_LOG_ARCHIVE */
1998
```

log init

- `log/log0log.cc:838, log_init()`



```
835 Initializes the log. */
836 UNIV_INTERN
837 void
838 log_init(void)
839 /*==========*/
840 {
841   log_sys = static_cast<log_t*>(mem_alloc(sizeof(log_t)));
842
843   mutex_create(log_sys_mutex_key, &log_sys->mutex, SYNC_LOG);
844
845   mutex_create(log_flush_order_mutex_key,
846          &log_sys->log_flush_order_mutex,
847          SYNC_LOG_FLUSH_ORDER);
848
849   mutex_enter(&(log_sys->mutex));
850
851   /* Start the lsn from one log block from zero: this way every
852   log record has a start lsn != zero, a fact which we will use */
853
854   log_sys->lsn = LOG_START_LSN;
855
856   ut_a(LOG_BUFFER_SIZE >= 16 * OS_FILE_LOG_BLOCK_SIZE);
857   ut_a(LOG_BUFFER_SIZE >= 4 * UNIV_PAGE_SIZE);
858
859   log_sys->buf_ptr = static_cast<byte*>(
860     mem_zalloc(LOG_BUFFER_SIZE + OS_FILE_LOG_BLOCK_SIZE));
861
862   log_sys->buf = static_cast<byte*>(
863     ut_align(log_sys->buf_ptr, OS_FILE_LOG_BLOCK_SIZE));
864
865   log_sys->buf_size = LOG_BUFFER_SIZE;
866   log_sys->is_extending = false;
```

create log sys

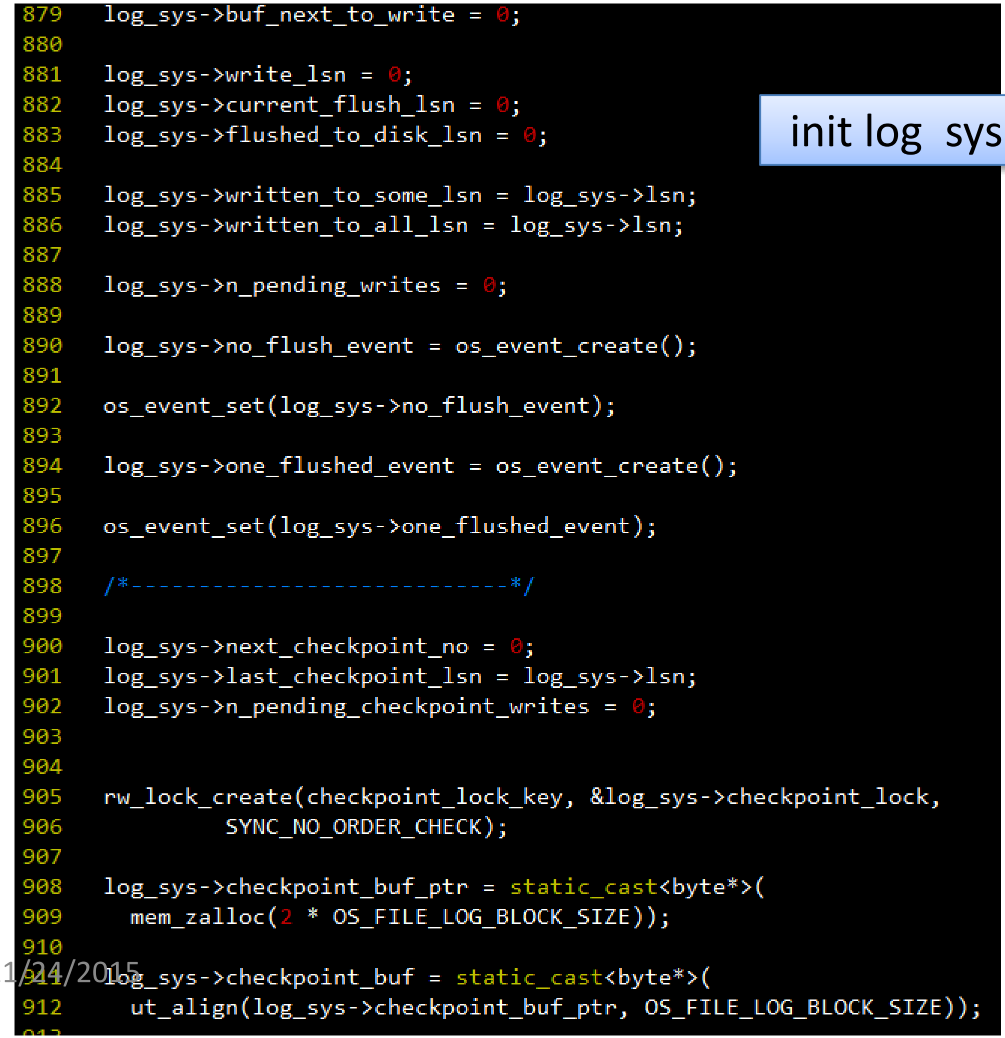
mutex for protection and synchronization

start_lsn != 0

buf_ptr : mem buffer
buf : page aligned buffer for log

log init

- log/log0log.cc:838, log_init()



```
log_sys->buf_next_to_write = 0;

log_sys->write_lsn = 0;
log_sys->current_flush_lsn = 0;
log_sys->flushed_to_disk_lsn = 0;

log_sys->written_to_some_lsn = log_sys->lsn;
log_sys->written_to_all_lsn = log_sys->lsn;

log_sys->n_pending_writes = 0;

log_sys->no_flush_event = os_event_create();

os_event_set(log_sys->no_flush_event);

log_sys->one_flushed_event = os_event_create();

os_event_set(log_sys->one_flushed_event);

/*----------------------------*/

log_sys->next_checkpoint_no = 0;
log_sys->last_checkpoint_lsn = log_sys->lsn;
log_sys->n_pending_checkpoint_writes = 0;


rw_lock_create(checkpoint_lock_key, &log_sys->checkpoint_lock,
         SYNC_NO_ORDER_CHECK);

log_sys->checkpoint_buf_ptr = static_cast<byte*>(
  mem_zalloc(2 * OS_FILE_LOG_BLOCK_SIZE));

log_sys->checkpoint_buf = static_cast<byte*>(
  ut_align(log_sys->checkpoint_buf_ptr, OS_FILE_LOG_BLOCK_SIZE));
```

init log sys variables

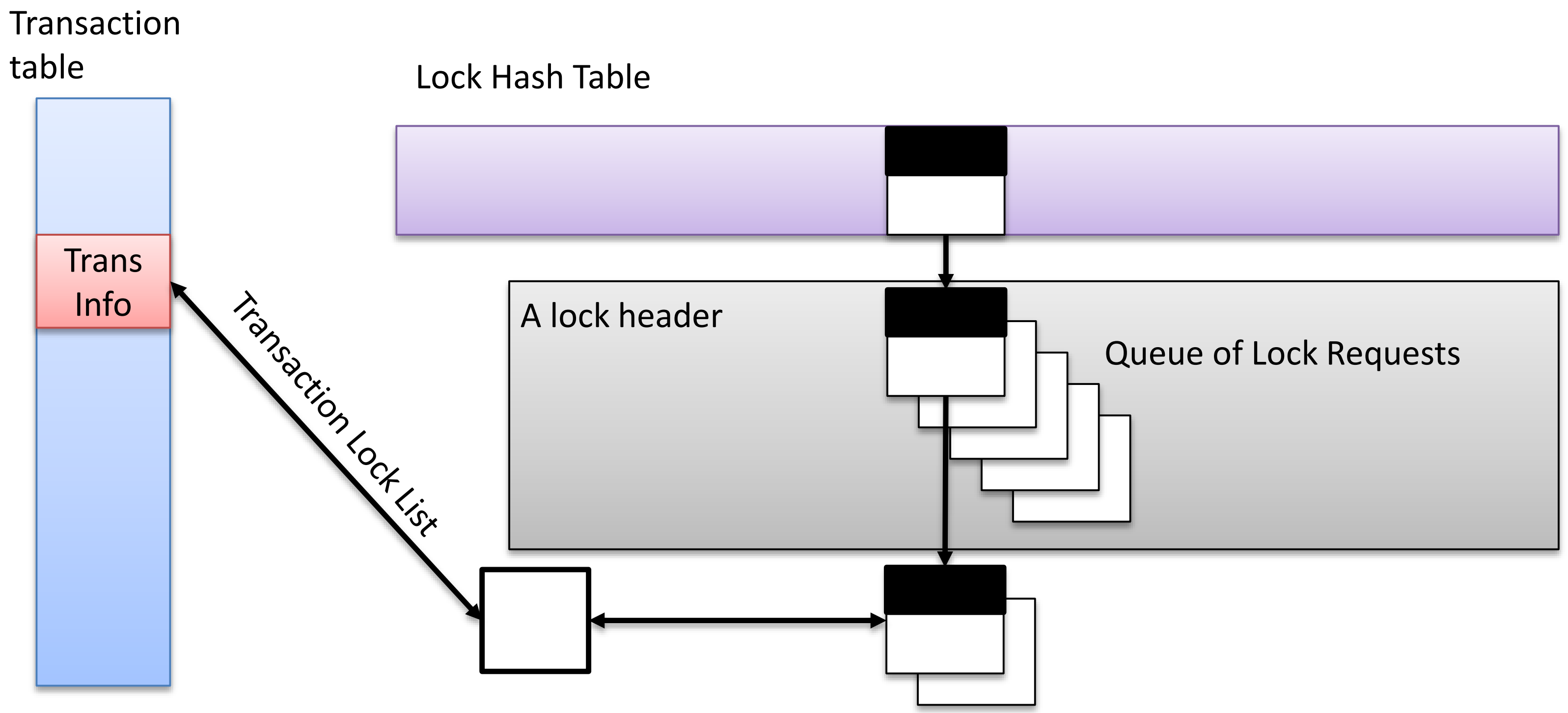
Contents

- Before we begin : tools
  - message logging and entry point
  - Coding conventions : the very first
- InnoDB Initialize
  - Init. in-memory objects and check configurations
  - aio system, file, buffer pool, log system
- Create objects
  - lock-system, io-handler, data files and log files
- Open
  - log and system table space
  - transaction system
- Recovery if necessary
- Create double write system if necessary

CREATE SERVER OBJECTS

Locking system

- Locking (and transaction model)
  - When a transaction updates a row in a table, or locks it with SELECT FOR UPDATE, InnoDB establishes a list or queue of locks on that row.
  - Similarly, InnoDB maintains a list of locks on a table for table-level locks transactions hold.



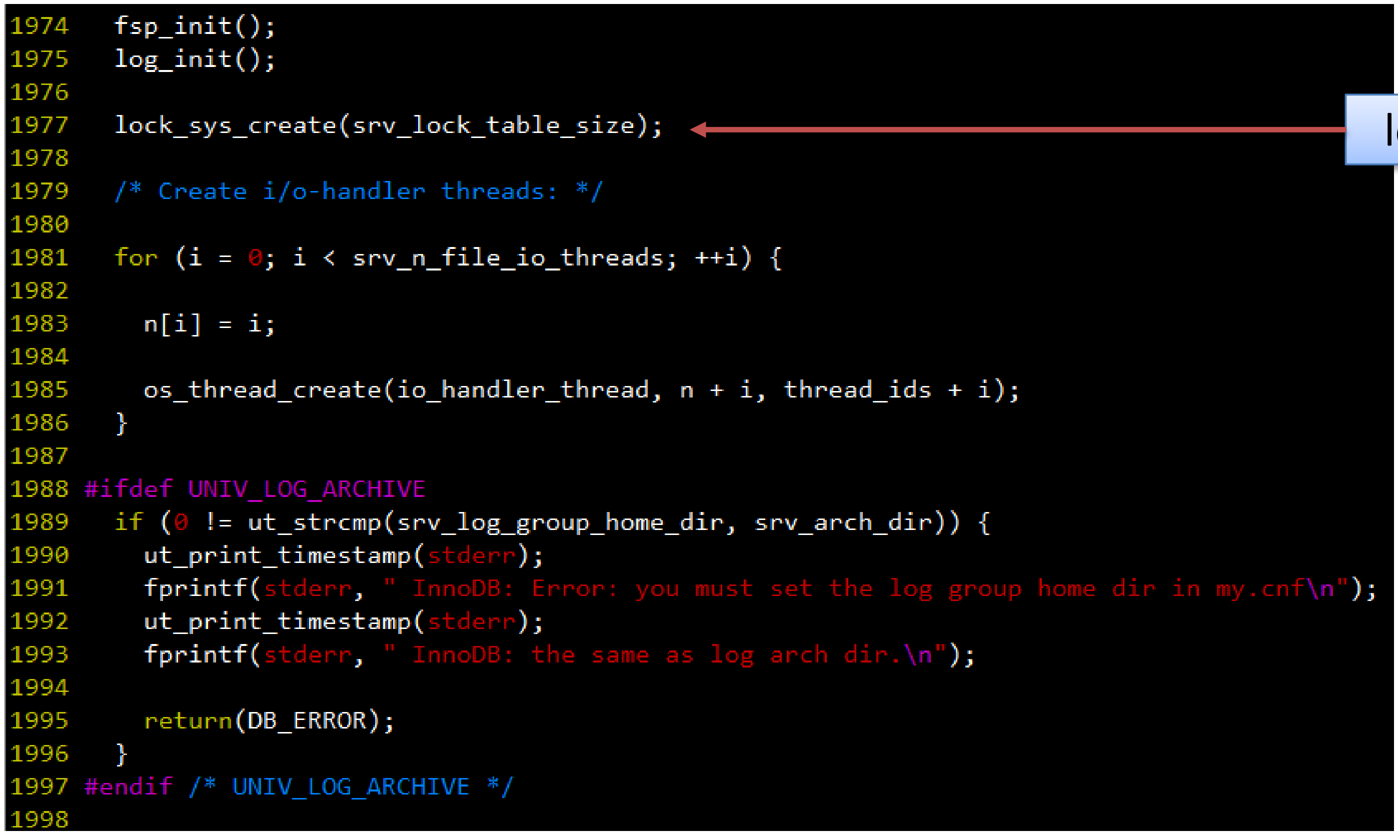
Locking system

- Lock Compatibility
  - S : shared, X: eXclusive

	Granted Mode						
Requested Mode	None	IS	IX	S	SIX	U	X
IS	+	+	+	+	+	-	-
IX	+	+	+	-	-	-	-
S	+	+	-	+	-	-	-
SIX	+	+	-	-	-	-	-
U	+	-	-	+	-	-	-
X	+	-	-	-	-	-	-

Locking system

- srv/srv0start:1977, lock_sys_create()

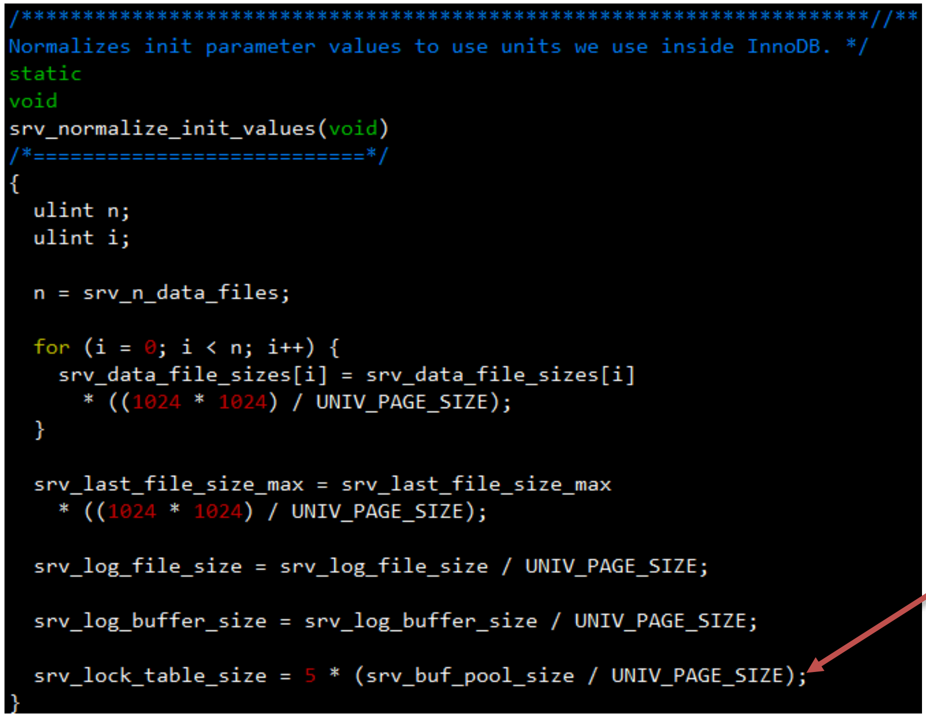


```
1974   fsp_init();
1975   log_init();
1976
1977   lock_sys_create(srv_lock_table_size);
1978
1979   /* Create i/o-handler threads: */
1980
1981   for (i = 0; i < srv_n_file_io_threads; ++i) {
1982
1983     n[i] = i;
1984
1985     os_thread_create(io_handler_thread, n + i, thread_ids + i);
1986   }
1987
1988 #ifdef UNIV_LOG_ARCHIVE
1989   if (0 != ut_strcmp(srv_log_group_home_dir, srv_arch_dir)) {
1990     ut_print_timestamp(stderr);
1991     fprintf(stderr, " InnoDB: Error: you must set the log group home dir in my.cnf\n");
1992     ut_print_timestamp(stderr);
1993     fprintf(stderr, " InnoDB: the same as log arch dir.\n");
1994
1995     return(DB_ERROR);
1996   }
1997 #endif /* UNIV_LOG_ARCHIVE */
1998
```

lock system create

Locking system

- lock table size
  - in `srv_normalize_init_values()`
  - 5 * # of pages in the buffer pool

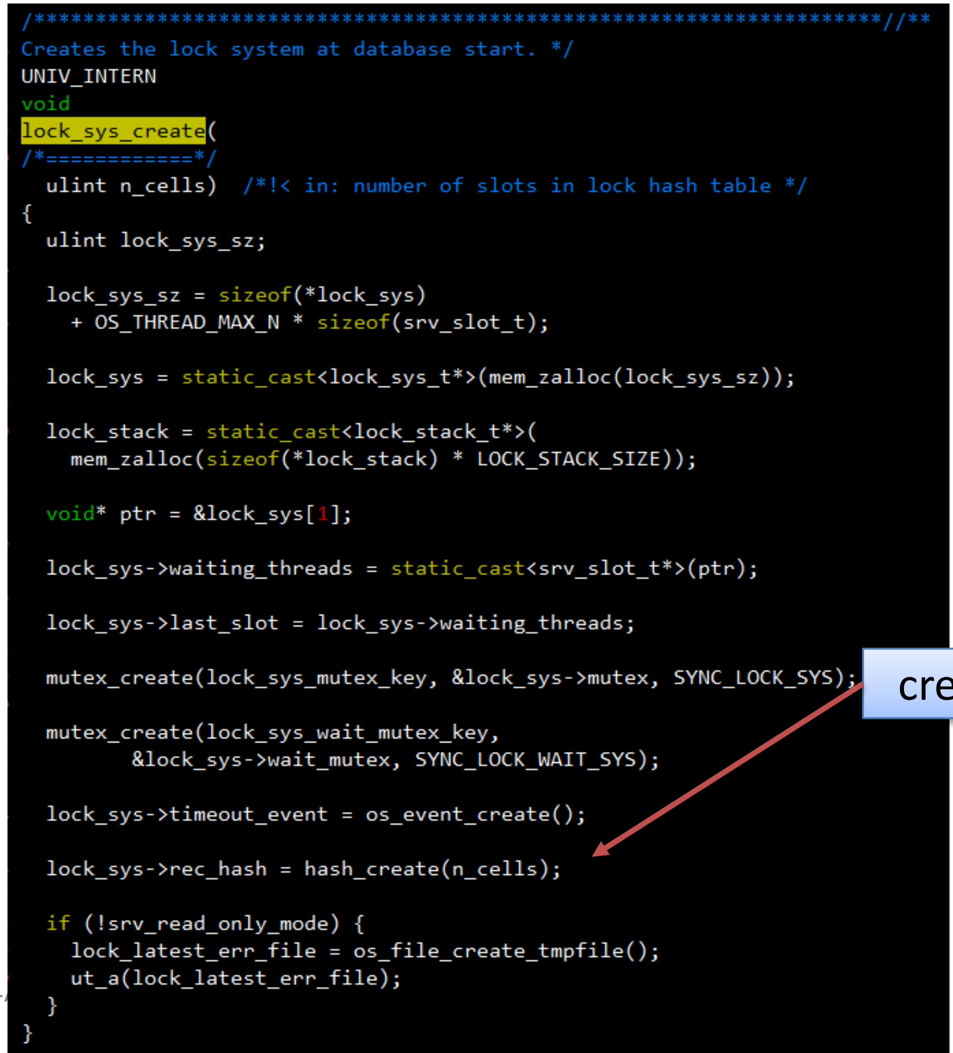


```
/*********************************************************************//**
Normalizes init parameter values to use units we use inside InnoDB. */
static
void
srv_normalize_init_values(void)
/*===========================*/
{
  ulint n;
  ulint i;

  n = srv_n_data_files;

  for (i = 0; i < n; i++) {
    srv_data_file_sizes[i] = srv_data_file_sizes[i]
      * ((1024 * 1024) / UNIV_PAGE_SIZE);
  }

  srv_last_file_size_max = srv_last_file_size_max
    * ((1024 * 1024) / UNIV_PAGE_SIZE);

  srv_log_file_size = srv_log_file_size / UNIV_PAGE_SIZE;

  srv_log_buffer_size = srv_log_buffer_size / UNIV_PAGE_SIZE;

  srv_lock_table_size = 5 * (srv_buf_pool_size / UNIV_PAGE_SIZE);
}
```

lock table size

Lock system

- lock/lock0lock:589, lock_sys_create()

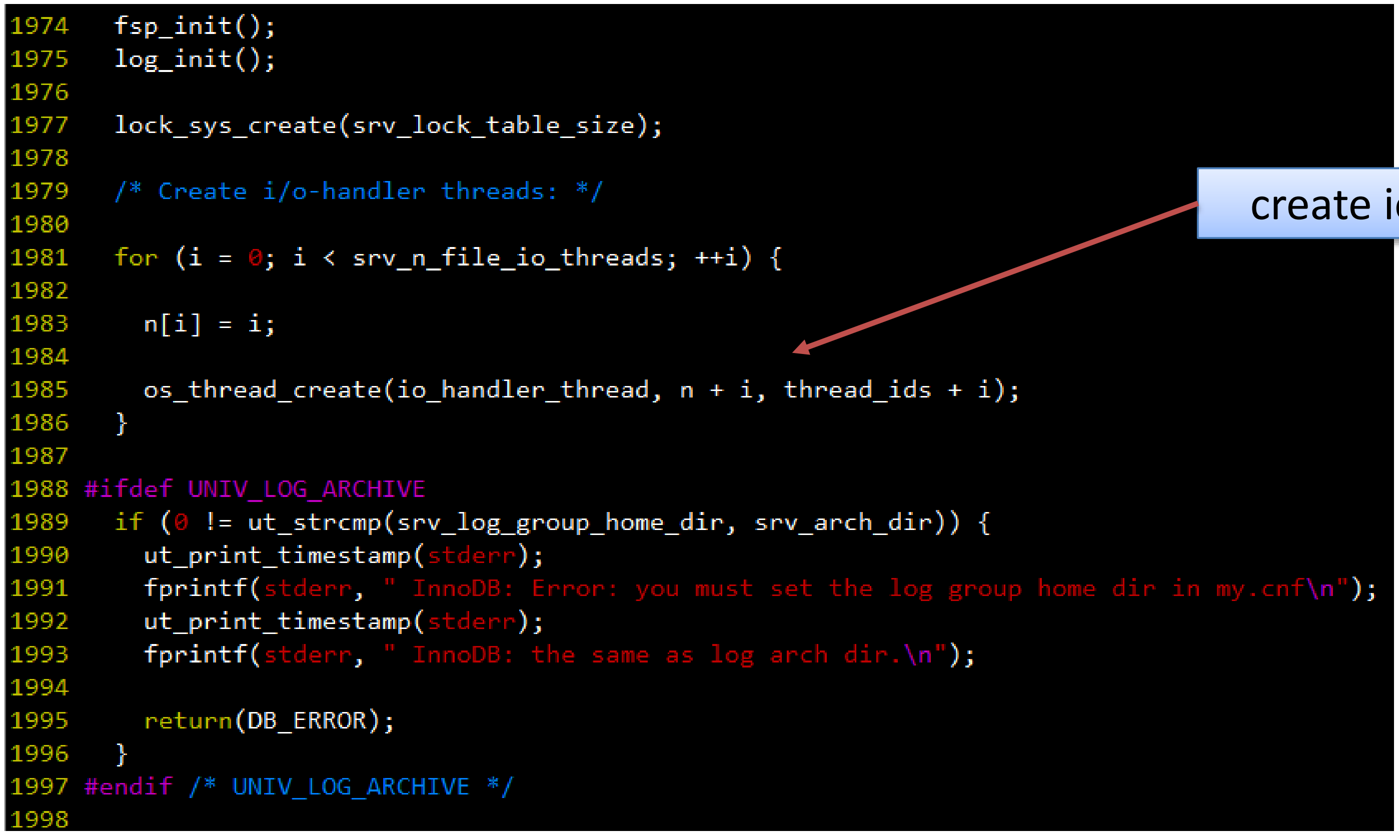


```
/*********************************************************************//**
Creates the lock system at database start. */
UNIV_INTERN
void
lock_sys_create(
/*============*/
  ulint n_cells)  /*!< in: number of slots in lock hash table */
{
  ulint lock_sys_sz;

  lock_sys_sz = sizeof(*lock_sys)
    + OS_THREAD_MAX_N * sizeof(srv_slot_t);

  lock_sys = static_cast<lock_sys_t*>(mem_zalloc(lock_sys_sz));

  lock_stack = static_cast<lock_stack_t*>(
    mem_zalloc(sizeof(*lock_stack) * LOCK_STACK_SIZE));

  void* ptr = &lock_sys[1];

  lock_sys->waiting_threads = static_cast<srv_slot_t*>(ptr);

  lock_sys->last_slot = lock_sys->waiting_threads;

  mutex_create(lock_sys_mutex_key, &lock_sys->mutex, SYNC_LOCK_SYS);

  mutex_create(lock_sys_wait_mutex_key,
         &lock_sys->wait_mutex, SYNC_LOCK_WAIT_SYS);

  lock_sys->timeout_event = os_event_create();

  lock_sys->rec_hash = hash_create(n_cells);

  if (!srv_read_only_mode) {
    lock_latest_err_file = os_file_create_tmpfile();
    ut_a(lock_latest_err_file);
  }
}
```

create record hash

Create io threads

- srv/srv0start:1985



```
1974  fsp_init();
1975  log_init();
1976
1977  lock_sys_create(srv_lock_table_size);
1978
1979  /* Create i/o-handler threads: */
1980
1981  for (i = 0; i < srv_n_file_io_threads; ++i) {
1982
1983    n[i] = i;
1984
1985    os_thread_create(io_handler_thread, n + i, thread_ids + i);
1986  }
1987
1988 #ifdef UNIV_LOG_ARCHIVE
1989  if (0 != ut_strcmp(srv_log_group_home_dir, srv_arch_dir)) {
1990    ut_print_timestamp(stderr);
1991    fprintf(stderr, " InnoDB: Error: you must set the log group home dir in my.cnf\n");
1992    ut_print_timestamp(stderr);
1993    fprintf(stderr, " InnoDB: the same as log arch dir.\n");
1994
1995    return(DB_ERROR);
1996  }
1997 #endif /* UNIV_LOG_ARCHIVE */
1998
```

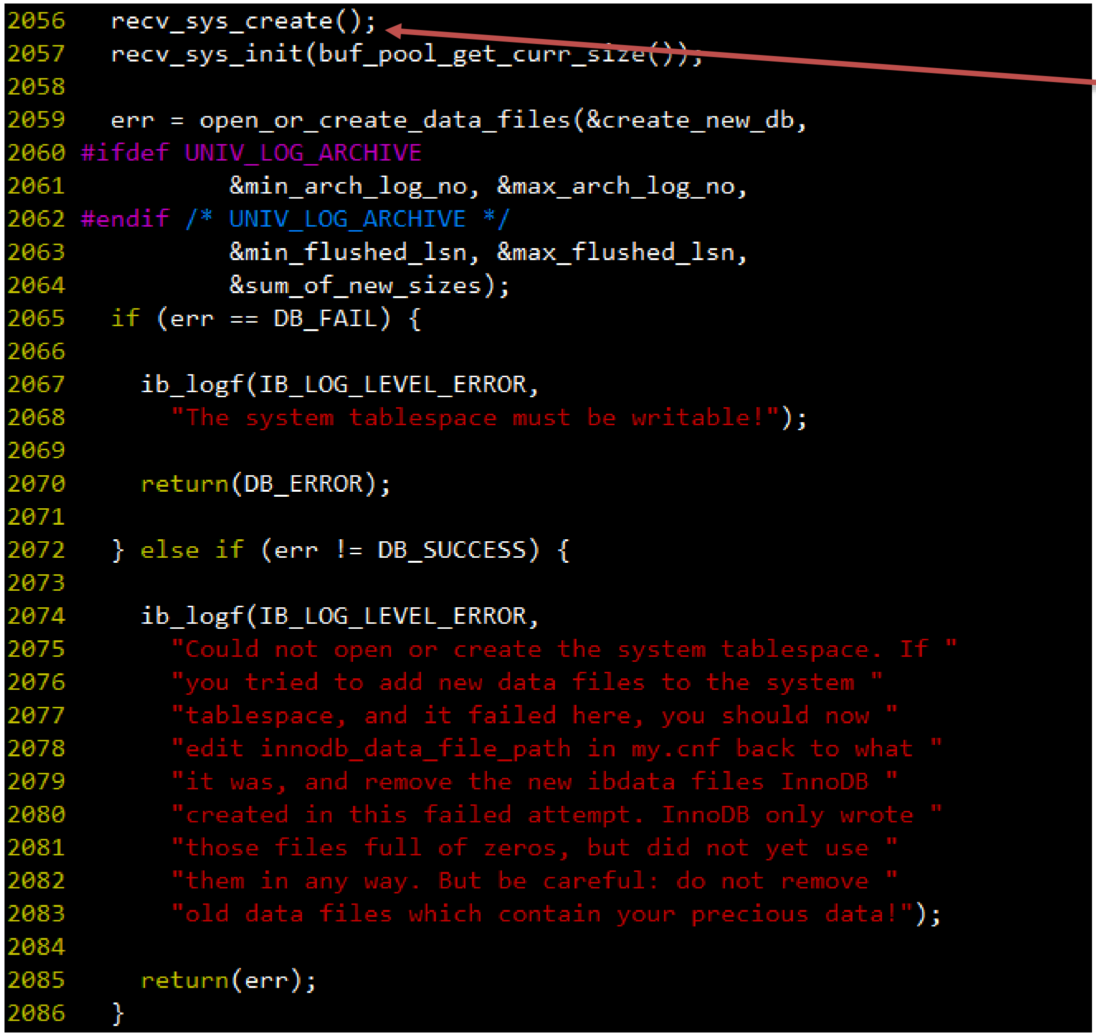
create io threads

Recovery system

- Recovery system
  - It mainly focuses on write-ahead-log and double write
  - If the system was not clean shutdown, then we have to do recovery

Recovery system

- srv/srv0start:2056, recv_sys_create()

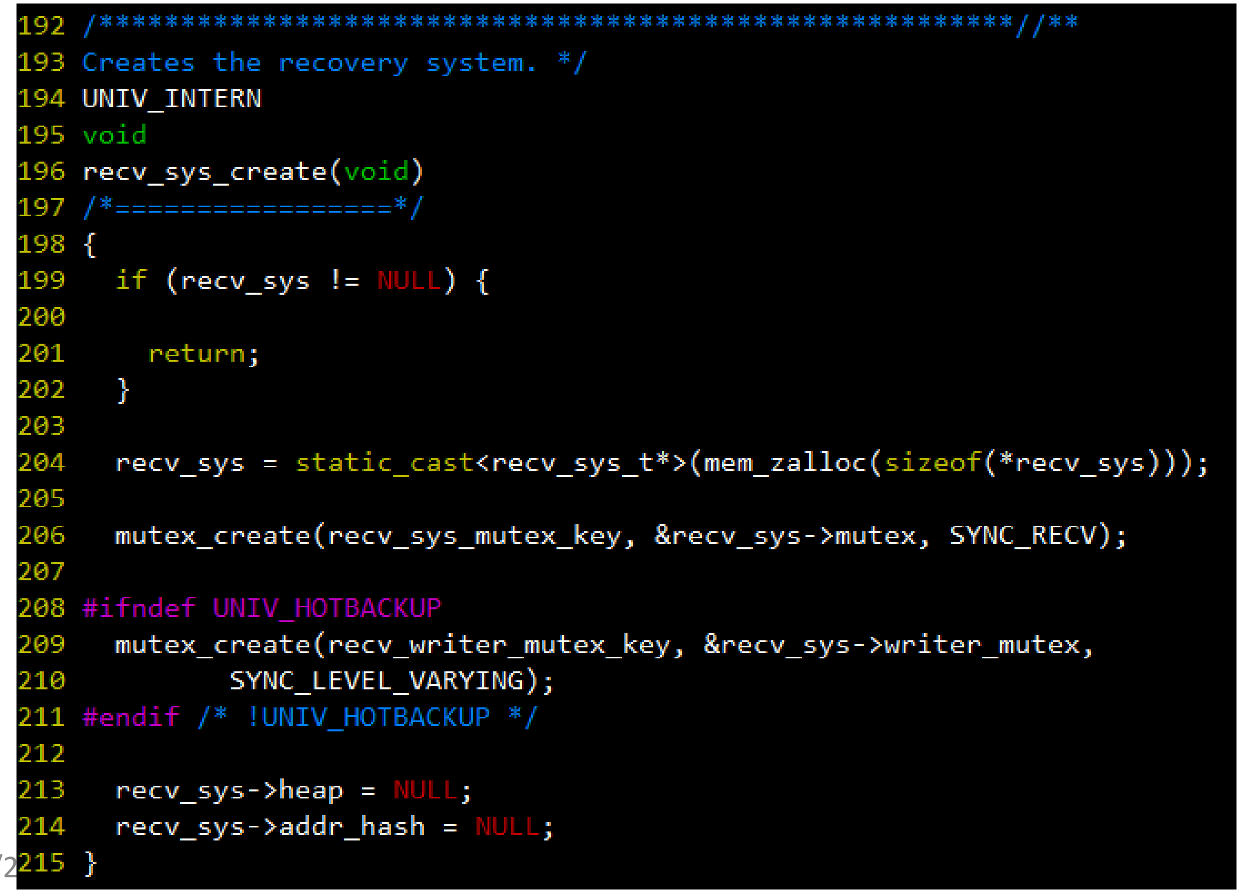


```
2056   recv_sys_create();
2057   recv_sys_init(buf_pool_get_curr_size());
2058
2059   err = open_or_create_data_files(&create_new_db,
2060 #ifdef UNIV_LOG_ARCHIVE
2061           &min_arch_log_no, &max_arch_log_no,
2062 #endif /* UNIV_LOG_ARCHIVE */
2063           &min_flushed_lsn, &max_flushed_lsn,
2064           &sum_of_new_sizes);
2065   if (err == DB_FAIL) {
2066
2067     ib_logf(IB_LOG_LEVEL_ERROR,
2068       "The system tablespace must be writable!");
2069
2070     return(DB_ERROR);
2071
2072   } else if (err != DB_SUCCESS) {
2073
2074     ib_logf(IB_LOG_LEVEL_ERROR,
2075       "Could not open or create the system tablespace. If "
2076       "you tried to add new data files to the system "
2077       "tablespace, and it failed here, you should now "
2078       "edit innodb_data_file_path in my.cnf back to what "
2079       "it was, and remove the new ibdata files InnoDB "
2080       "created in this failed attempt. InnoDB only wrote "
2081       "those files full of zeros, but did not yet use "
2082       "them in any way. But be careful: do not remove "
2083       "old data files which contain your precious data!");
2084
2085     return(err);
2086   }
```

create recovery system

Recovery system

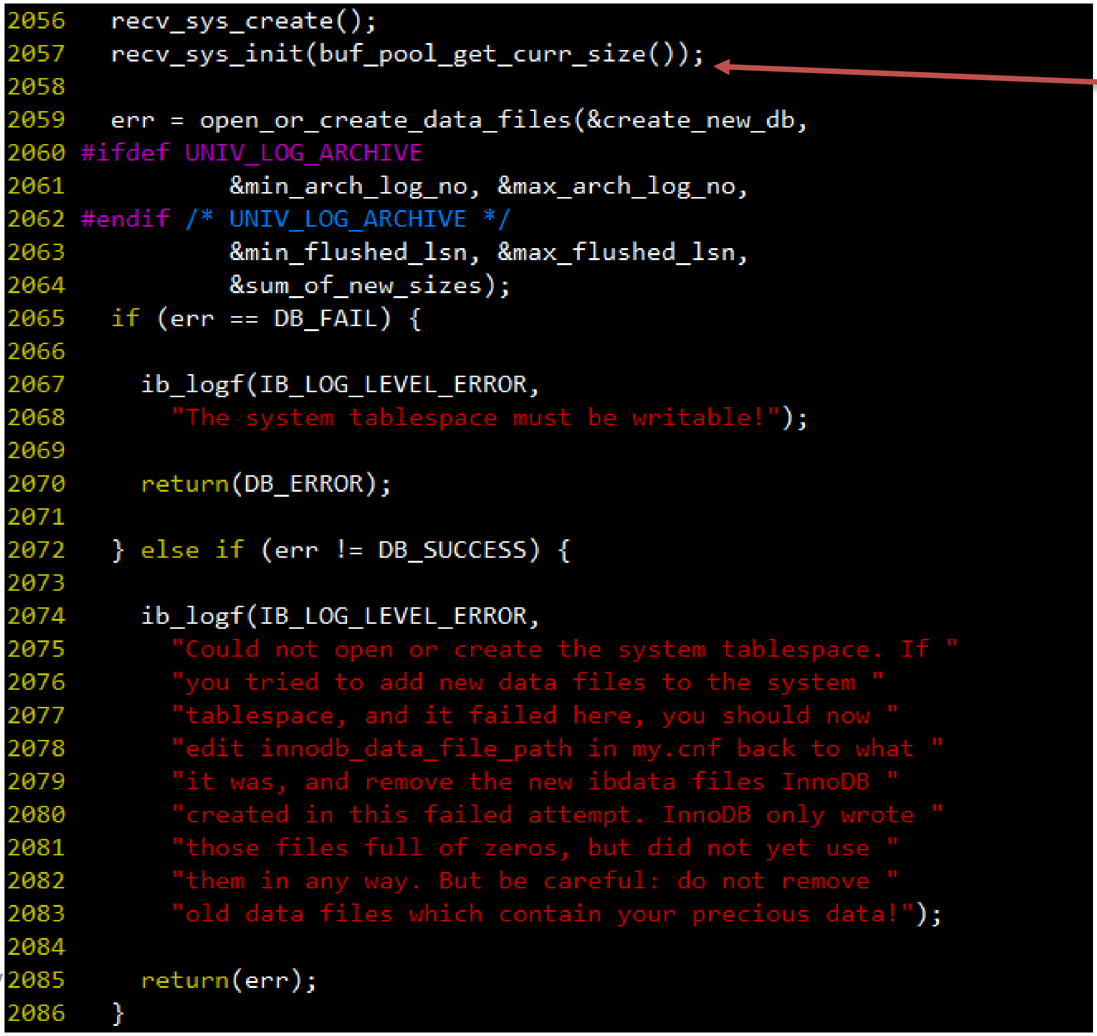
- `log/log0recv.cc:196, recv_sys_create()`
  - just make `recv_sys`



```
192 /********************************************************//**
193 Creates the recovery system. */
194 UNIV_INTERN
195 void
196 recv_sys_create(void)
197 /*=================*/
198 {
199   if (recv_sys != NULL) {
200 
201     return;
202   }
203 
204   recv_sys = static_cast<recv_sys_t*>(mem_zalloc(sizeof(*recv_sys)));
205 
206   mutex_create(recv_sys_mutex_key, &recv_sys->mutex, SYNC_RECV);
207 
208 #ifndef UNIV_HOTBACKUP
209   mutex_create(recv_writer_mutex_key, &recv_sys->writer_mutex,
210          SYNC_LEVEL_VARYING);
211 #endif /* !UNIV_HOTBACKUP */
212 
213   recv_sys->heap = NULL;
214   recv_sys->addr_hash = NULL;
215 }
```

Recovery system

- `srv/srv0start:2057, recv_sys_init()`

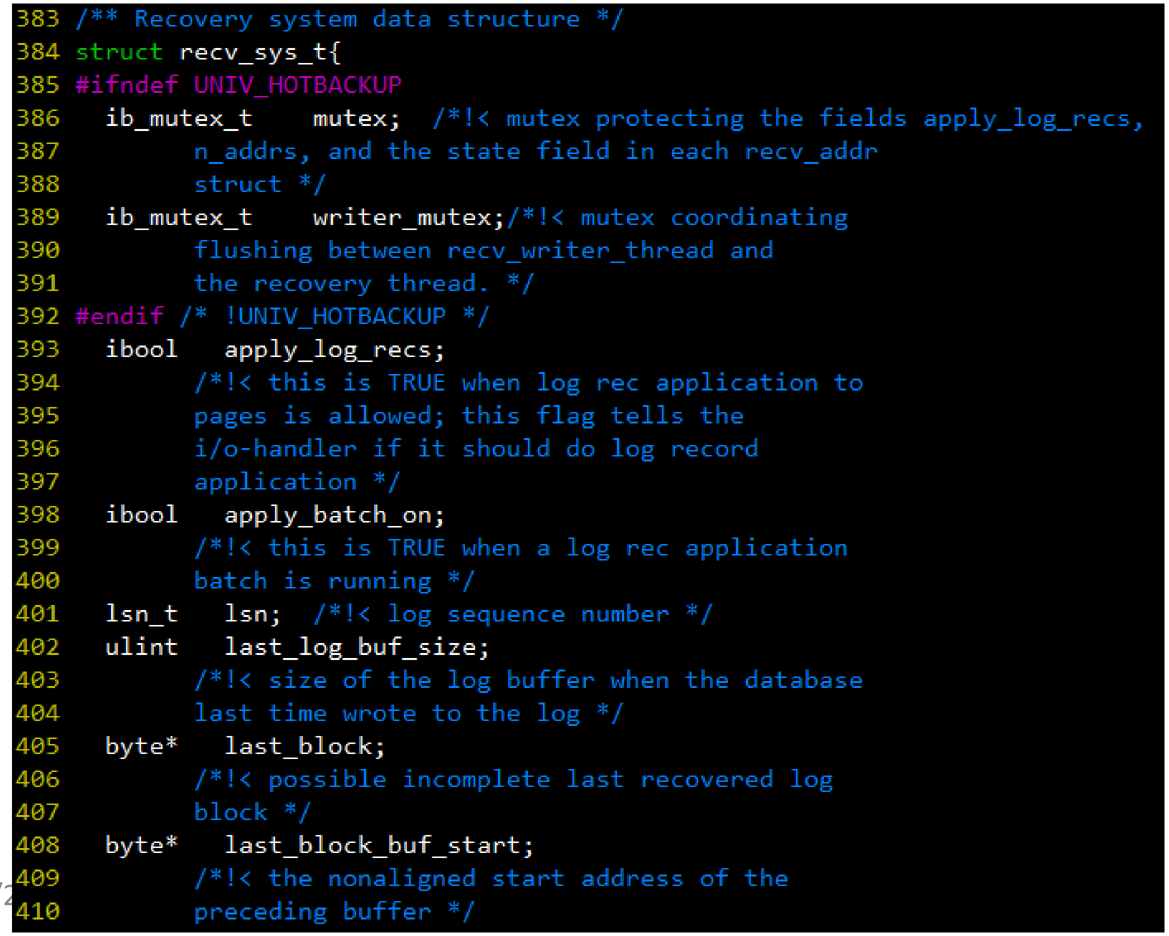


```
2056   recv_sys_create();
2057   recv_sys_init(buf_pool_get_curr_size());
2058
2059   err = open_or_create_data_files(&create_new_db,
2060 #ifdef UNIV_LOG_ARCHIVE
2061           &min_arch_log_no, &max_arch_log_no,
2062 #endif /* UNIV_LOG_ARCHIVE */
2063           &min_flushed_lsn, &max_flushed_lsn,
2064           &sum_of_new_sizes);
2065   if (err == DB_FAIL) {
2066
2067     ib_logf(IB_LOG_LEVEL_ERROR,
2068       "The system tablespace must be writable!");
2069
2070     return(DB_ERROR);
2071
2072   } else if (err != DB_SUCCESS) {
2073
2074     ib_logf(IB_LOG_LEVEL_ERROR,
2075       "Could not open or create the system tablespace. If "
2076       "you tried to add new data files to the system "
2077       "tablespace, and it failed here, you should now "
2078       "edit innodb_data_file_path in my.cnf back to what "
2079       "it was, and remove the new ibdata files InnoDB "
2080       "created in this failed attempt. InnoDB only wrote "
2081       "those files full of zeros, but did not yet use "
2082       "them in any way. But be careful: do not remove "
2083       "old data files which contain your precious data!");
2084
2085     return(err);
2086   }
```

init recovery system

Recovery system

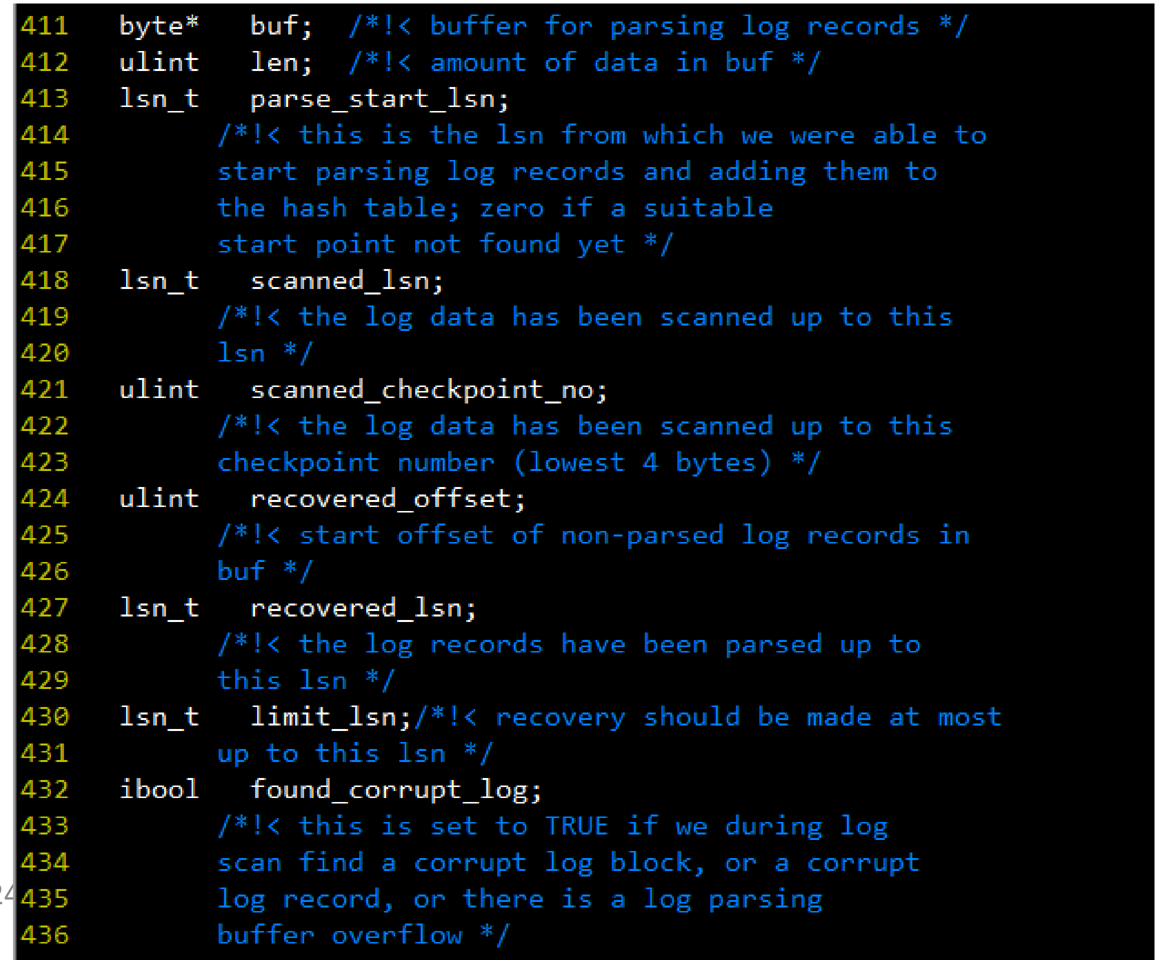
- Definition recovery system
  - include/log0recv.h:384,recv_sys_t



```
383 /** Recovery system data structure */
384 struct recv_sys_t{
385 #ifndef UNIV_HOTBACKUP
386   ib_mutex_t    mutex;  /*!< mutex protecting the fields apply_log_recs,
387         n_addrs, and the state field in each recv_addr
388         struct */
389   ib_mutex_t    writer_mutex;/*!< mutex coordinating
390         flushing between recv_writer_thread and
391         the recovery thread. */
392 #endif /* !UNIV_HOTBACKUP */
393   ibool   apply_log_recs;
394         /*!< this is TRUE when log rec application to
395         pages is allowed; this flag tells the
396         i/o-handler if it should do log record
397         application */
398   ibool   apply_batch_on;
399         /*!< this is TRUE when a log rec application
400         batch is running */
401   lsn_t   lsn;  /*!< log sequence number */
402   ulint   last_log_buf_size;
403         /*!< size of the log buffer when the database
404         last time wrote to the log */
405   byte*   last_block;
406         /*!< possible incomplete last recovered log
407         block */
408   byte*   last_block_buf_start;
409         /*!< the nonaligned start address of the
410         preceding buffer */
```

Recovery system

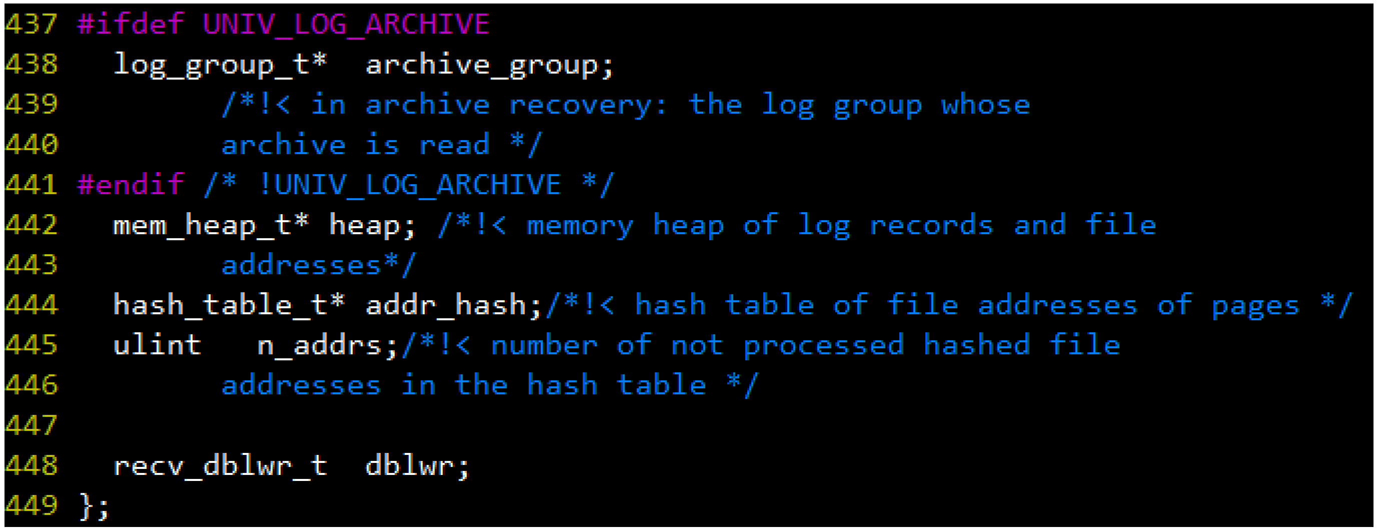
- Definition recovery system
  - include/log0recv.h:384,recv_sys_t



```
byte*   buf;  /*!< buffer for parsing log records */
ulint   len;  /*!< amount of data in buf */
lsn_t   parse_start_lsn;
      /*!< this is the lsn from which we were able to
      start parsing log records and adding them to
      the hash table; zero if a suitable
      start point not found yet */
lsn_t   scanned_lsn;
      /*!< the log data has been scanned up to this
      lsn */
ulint   scanned_checkpoint_no;
      /*!< the log data has been scanned up to this
      checkpoint number (lowest 4 bytes) */
ulint   recovered_offset;
      /*!< start offset of non-parsed log records in
      buf */
lsn_t   recovered_lsn;
      /*!< the log records have been parsed up to
      this lsn */
lsn_t   limit_lsn;/*!< recovery should be made at most
      up to this lsn */
ibool   found_corrupt_log;
      /*!< this is set to TRUE if we during log
      scan find a corrupt log block, or a corrupt
      log record, or there is a log parsing
      buffer overflow */
```

Recovery system

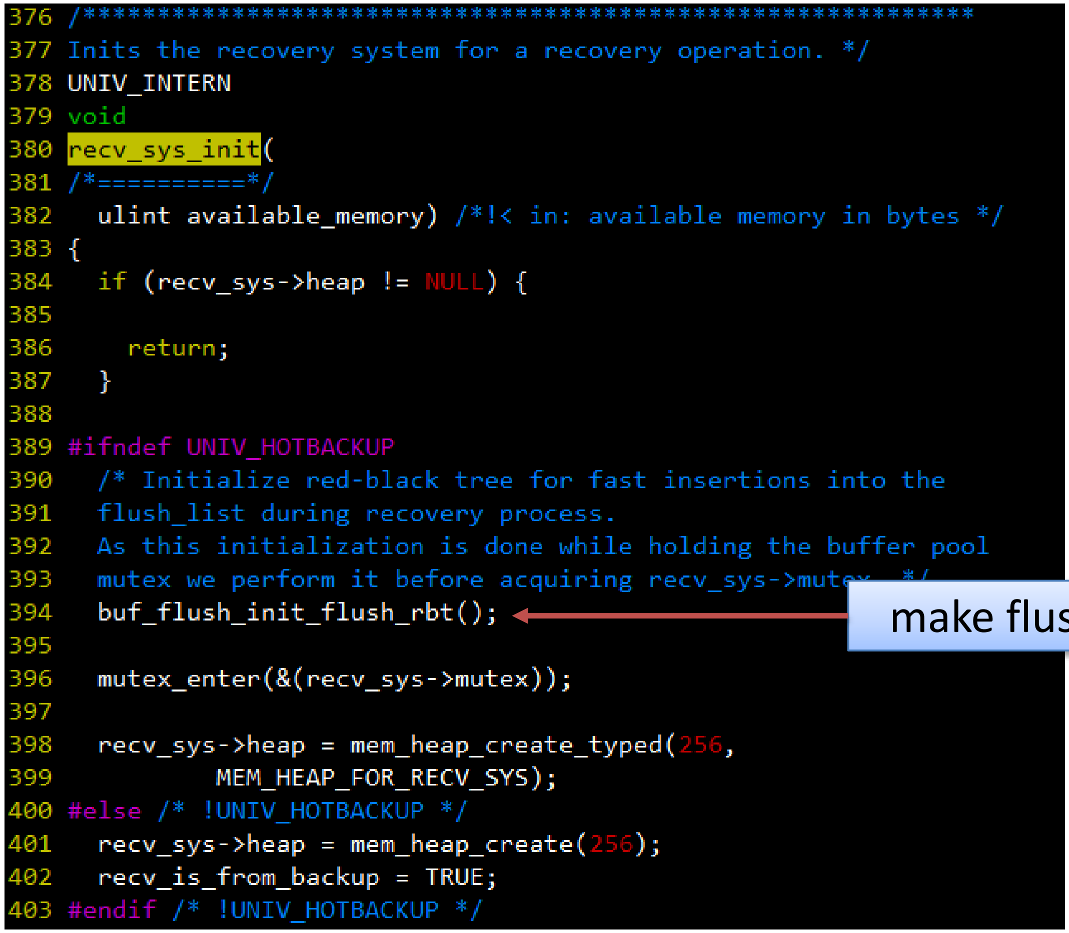
- Definition recovery system
  - `include/log0recv.h:384,recv_sys_t`



```
#ifdef UNIV_LOG_ARCHIVE
  log_group_t*  archive_group;
        /*!< in archive recovery: the log group whose
        archive is read */
#endif /* !UNIV_LOG_ARCHIVE */
  mem_heap_t* heap; /*!< memory heap of log records and file
        addresses*/
  hash_table_t* addr_hash;/*!< hash table of file addresses of pages */
  ulint   n_addrs;/*!< number of not processed hashed file
        addresses in the hash table */

  recv_dblwr_t  dblwr;
};
```

Recovery system

- log/log0recv.cc:380, recv_sys_init()

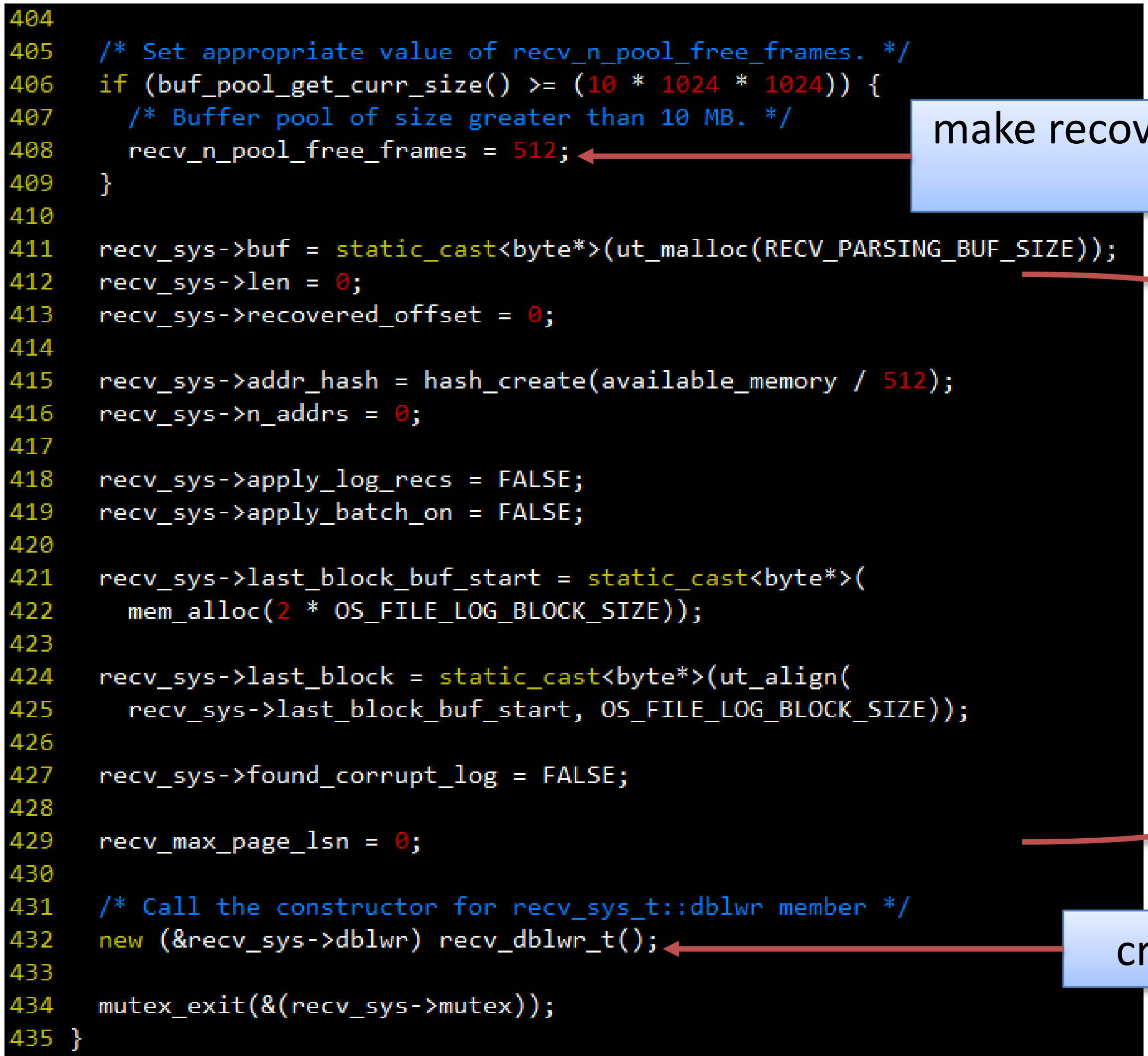


```
376 /******************************************************************
377 Inits the recovery system for a recovery operation. */
378 UNIV_INTERN
379 void
380 recv_sys_init(
381 /*==========*/
382   ulint available_memory) /*!< in: available memory in bytes */
383 {
384   if (recv_sys->heap != NULL) {
385 
386     return;
387   }
388 
389 #ifndef UNIV_HOTBACKUP
390   /* Initialize red-black tree for fast insertions into the
391   flush_list during recovery process.
392   As this initialization is done while holding the buffer pool
393   mutex we perform it before acquiring recv_sys->mutex. */
394   buf_flush_init_flush_rbt();
395 
396   mutex_enter(&(recv_sys->mutex));
397 
398   recv_sys->heap = mem_heap_create_typed(256,
399           MEM_HEAP_FOR_RECV_SYS);
400 #else /* !UNIV_HOTBACKUP */
401   recv_sys->heap = mem_heap_create(256);
402   recv_is_from_backup = TRUE;
403 #endif /* !UNIV_HOTBACKUP */
```

make flush list: see the details later

Recovery system

- `log/log0recv.cc:380, recv_sys_init()`



```

 /* Set appropriate value of recv_n_pool_free_frames. */
 if (buf_pool_get_curr_size() >= (10 * 1024 * 1024)) {
  /* Buffer pool of size greater than 10 MB. */
  recv_n_pool_free_frames = 512;
 }

 recv_sys->buf = static_cast<byte*>(ut_malloc(RECV_PARSING_BUF_SIZE));
 recv_sys->len = 0;
 recv_sys->recovered_offset = 0;

 recv_sys->addr_hash = hash_create(available_memory / 512);
 recv_sys->n_addrs = 0;

 recv_sys->apply_log_recs = FALSE;
 recv_sys->apply_batch_on = FALSE;

 recv_sys->last_block_buf_start = static_cast<byte*>(
  mem_alloc(2 * OS_FILE_LOG_BLOCK_SIZE));

 recv_sys->last_block = static_cast<byte*>(ut_align(
  recv_sys->last_block_buf_start, OS_FILE_LOG_BLOCK_SIZE));

 recv_sys->found_corrupt_log = FALSE;

 recv_max_page_lsn = 0;

 /* Call the constructor for recv_sys_t::dblwr member */
 new (&recv_sys->dblwr) recv_dblwr_t();

 mutex_exit(&(recv_sys->mutex));
}
```

make recovery system buffer pool for log scan

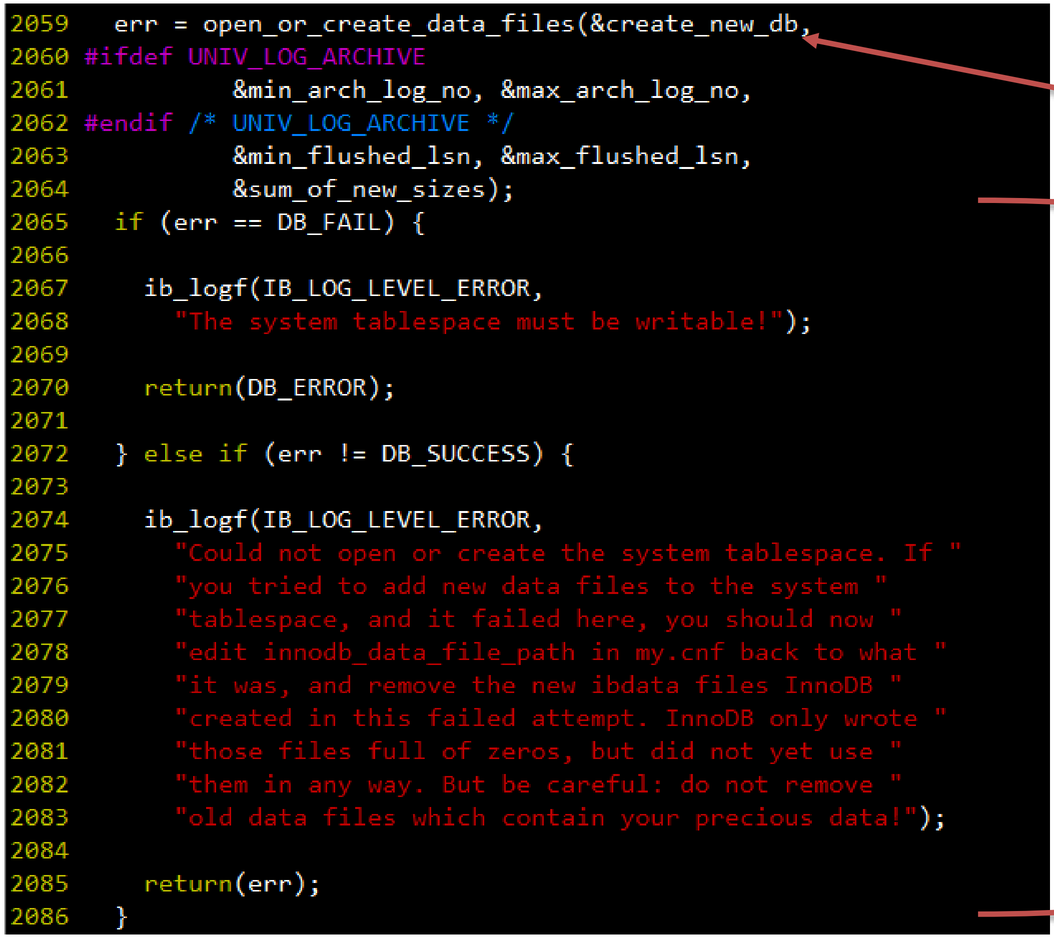
Init default variables and make buffers

create double write system

OPEN DATA FILES AND LOG FILE

check data files

- `srv/srv0start.cc:2059`



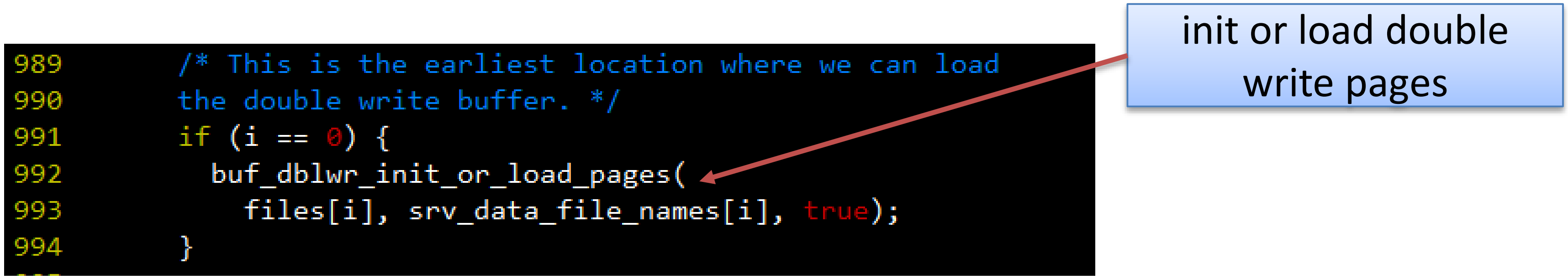
```
2059   err = open_or_create_data_files(&create_new_db,
2060 #ifdef UNIV_LOG_ARCHIVE
2061             &min_arch_log_no, &max_arch_log_no,
2062 #endif /* UNIV_LOG_ARCHIVE */
2063             &min_flushed_lsn, &max_flushed_lsn,
2064             &sum_of_new_sizes);
2065   if (err == DB_FAIL) {
2066 
2067     ib_logf(IB_LOG_LEVEL_ERROR,
2068       "The system tablespace must be writable!");
2069 
2070     return(DB_ERROR);
2071 
2072   } else if (err != DB_SUCCESS) {
2073 
2074     ib_logf(IB_LOG_LEVEL_ERROR,
2075       "Could not open or create the system tablespace. If "
2076       "you tried to add new data files to the system "
2077       "tablespace, and it failed here, you should now "
2078       "edit innodb_data_file_path in my.cnf back to what "
2079       "it was, and remove the new ibdata files InnoDB "
2080       "created in this failed attempt. InnoDB only wrote "
2081       "those files full of zeros, but did not yet use "
2082       "them in any way. But be careful: do not remove "
2083       "old data files which contain your precious data!");
2084 
2085     return(err);
2086   }
```

Open data files

open data files error handling

check data files

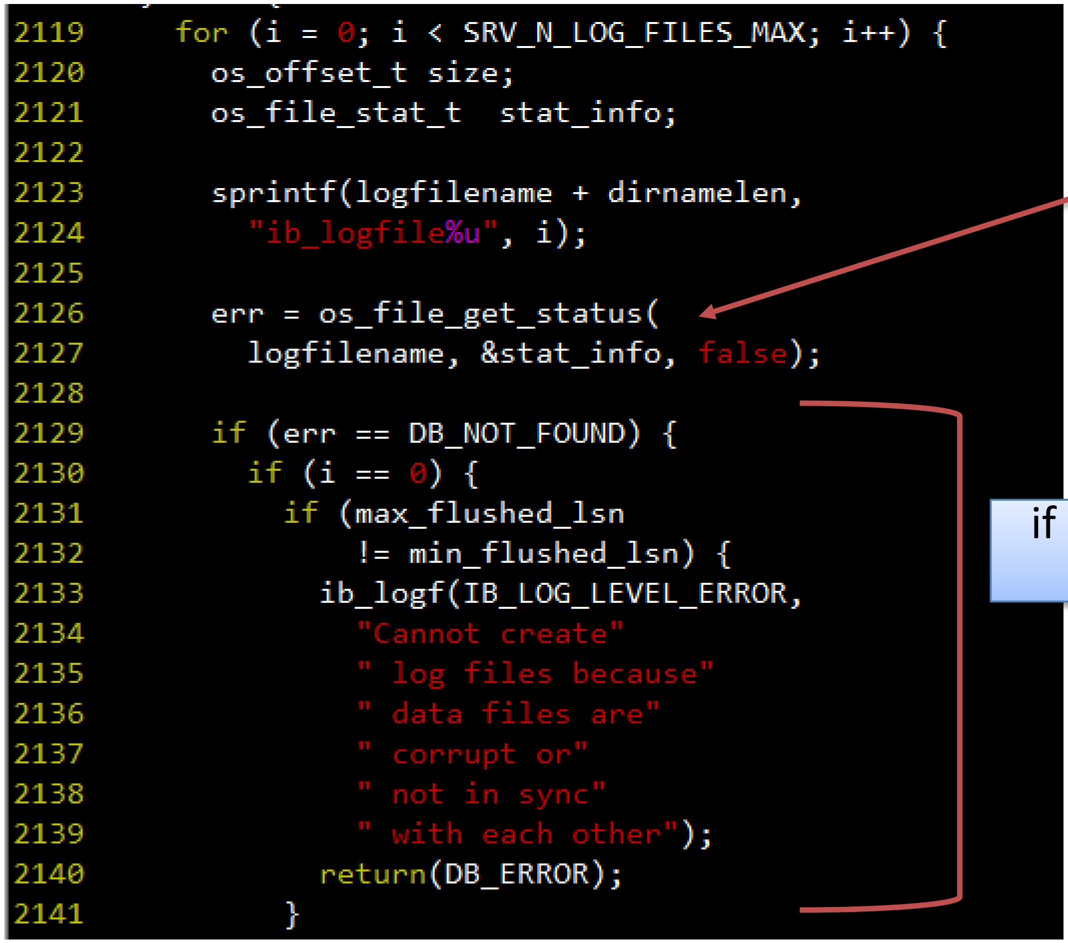
- `srv/srv0start.cc:759, open_or_create_data_files()`
  - Skip



```
        /* This is the earliest location where we can load
        the double write buffer. */
        if (i == 0) {
          buf_dblwr_init_or_load_pages(
            files[i], srv_data_file_names[i], true);
        }
```

check log files

- srv/srv0start.cc



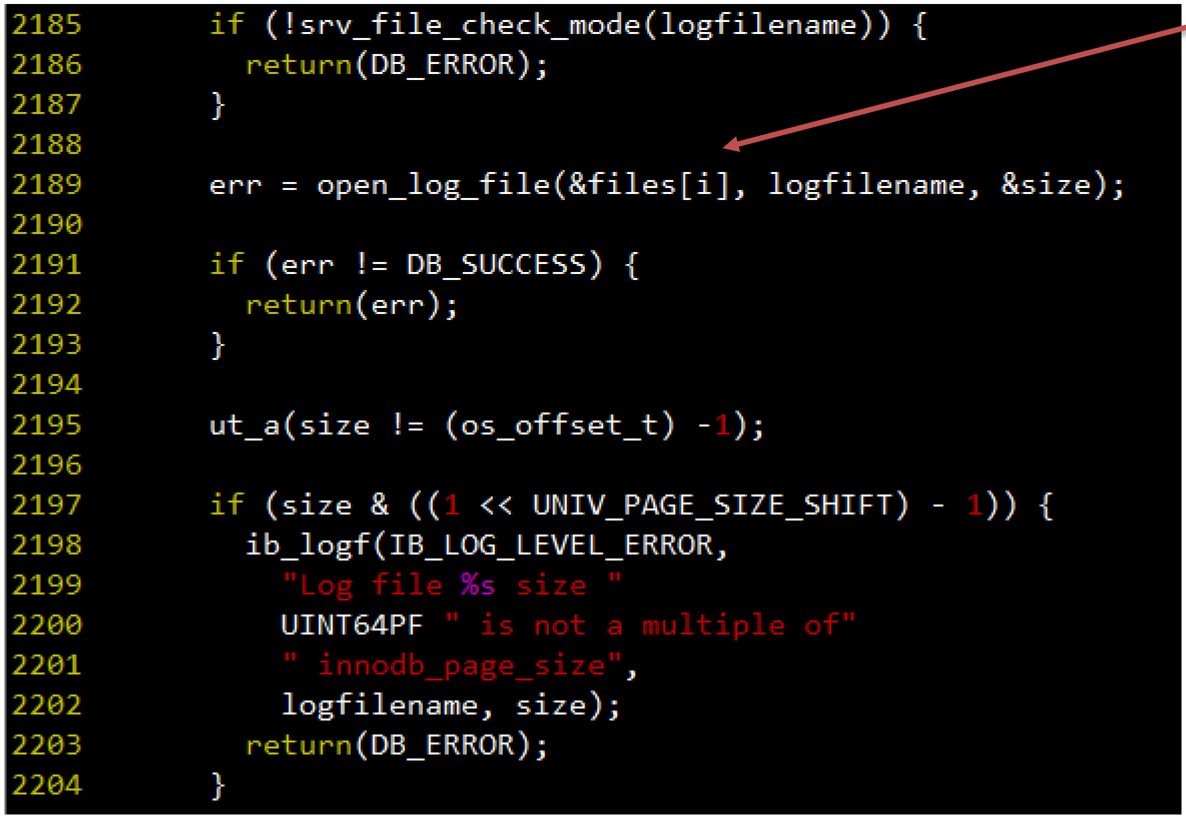
```
    for (i = 0; i < SRV_N_LOG_FILES_MAX; i++) {
      os_offset_t size;
      os_file_stat_t  stat_info;

      sprintf(logfilename + dirnamelen,
        "ib_logfile%u", i);

      err = os_file_get_status(
        logfilename, &stat_info, false);

      if (err == DB_NOT_FOUND) {
        if (i == 0) {
          if (max_flushed_lsn
              != min_flushed_lsn) {
            ib_logf(IB_LOG_LEVEL_ERROR,
              "Cannot create"
              " log files because"
              " data files are"
              " corrupt or"
              " not in sync"
              " with each other");
            return(DB_ERROR);
          }
```

get log file status

if log file not found, then create new one

check log files

- `srv/srv0start.cc:2189`

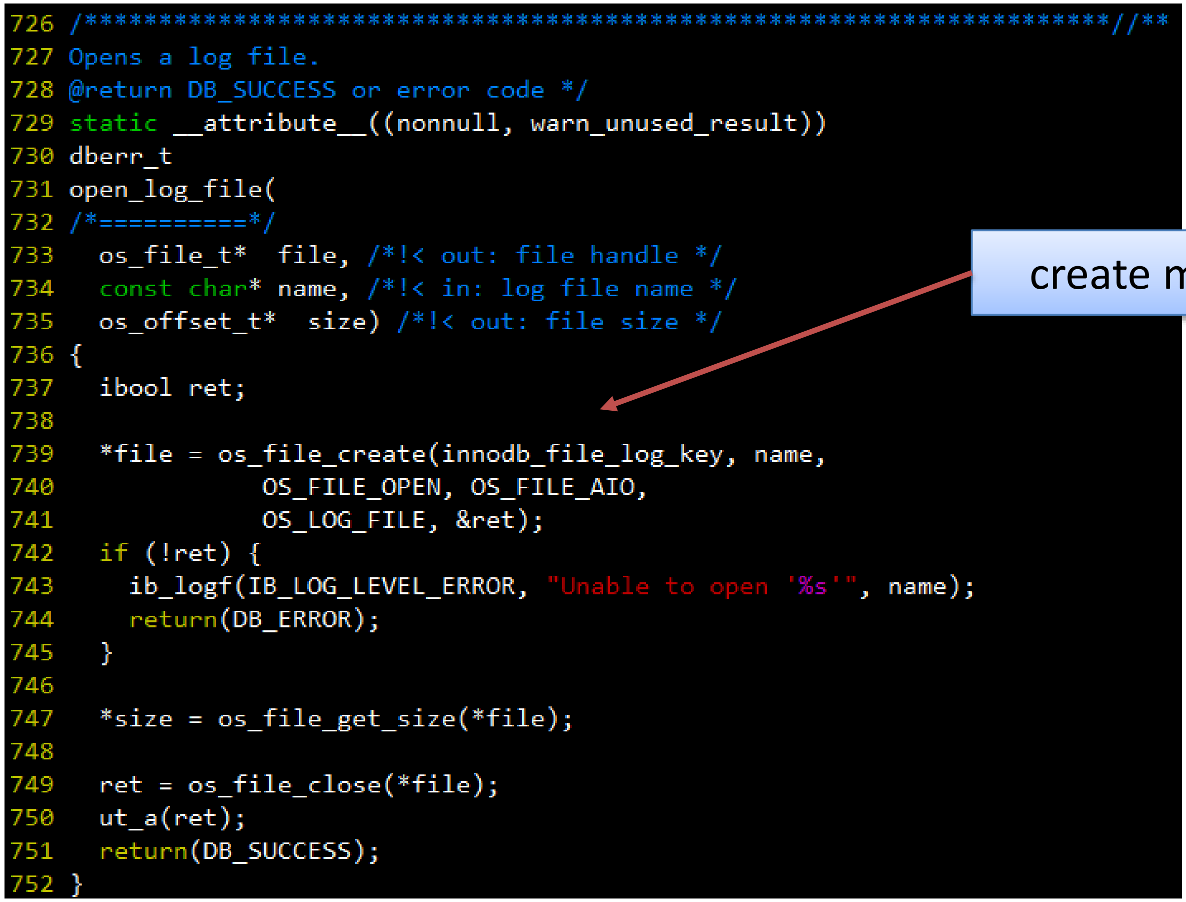


```
2185        if (!srv_file_check_mode(logfilename)) {
2186          return(DB_ERROR);
2187        }
2188
2189        err = open_log_file(&files[i], logfilename, &size);
2190
2191        if (err != DB_SUCCESS) {
2192          return(err);
2193        }
2194
2195        ut_a(size != (os_offset_t) -1);
2196
2197        if (size & ((1 << UNIV_PAGE_SIZE_SHIFT) - 1)) {
2198          ib_logf(IB_LOG_LEVEL_ERROR,
2199            "Log file %s size "
2200            UINT64PF " is not a multiple of"
2201            " innodb_page_size",
2202            logfilename, size);
2203          return(DB_ERROR);
2204        }
```

get log file status

check log files

- srv/srv0start.cc:3219, open_log_file()

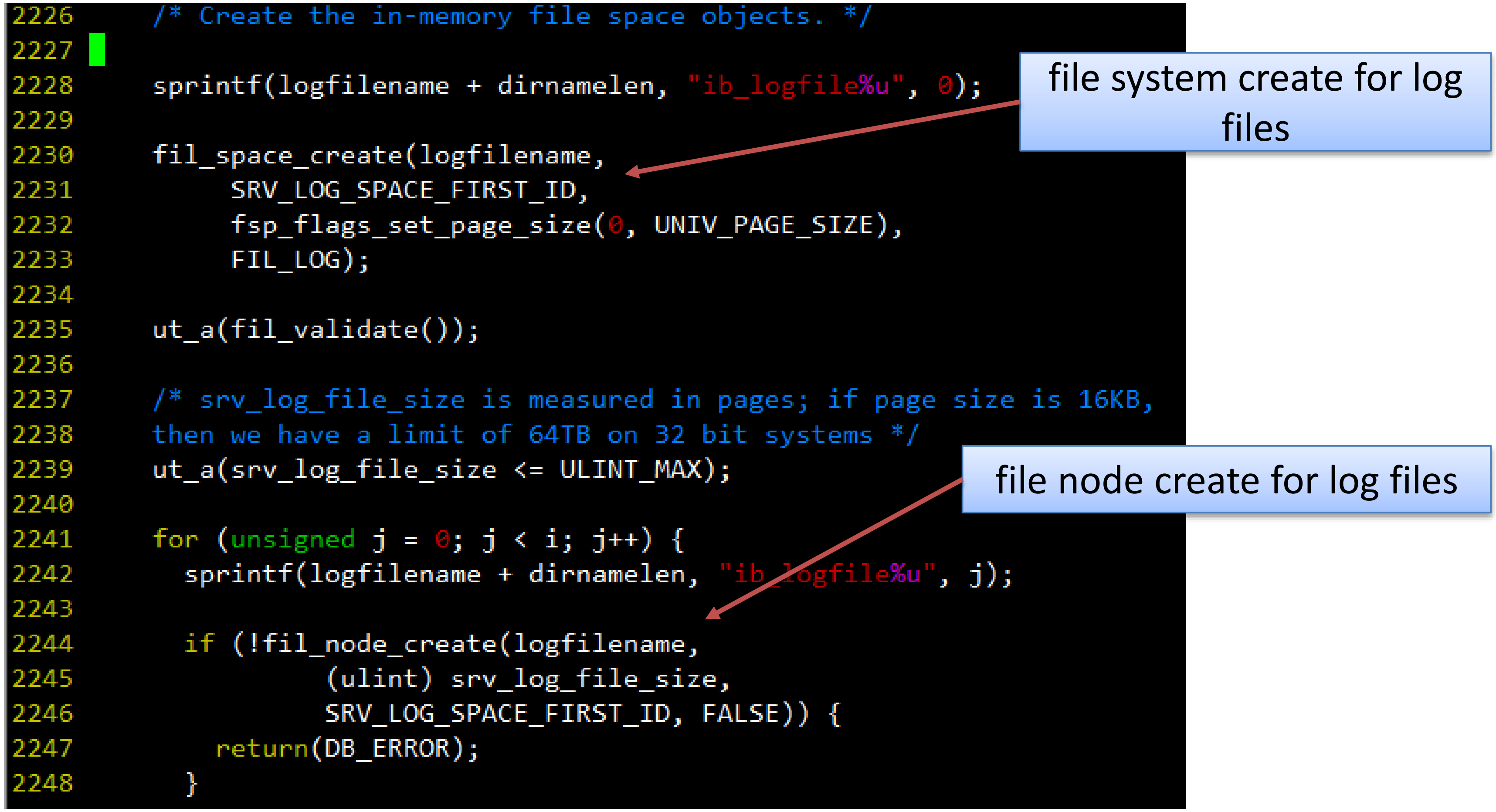


```
/********************************************************************//**
Opens a log file.
@return DB_SUCCESS or error code */
static __attribute__((nonnull, warn_unused_result))
dberr_t
open_log_file(
/*==========*/
  os_file_t*  file, /*!< out: file handle */
  const char* name, /*!< in: log file name */
  os_offset_t*  size) /*!< out: file size */
{
  ibool ret;

  *file = os_file_create(innodb_file_log_key, name,
             OS_FILE_OPEN, OS_FILE_AIO,
             OS_LOG_FILE, &ret);
  if (!ret) {
    ib_logf(IB_LOG_LEVEL_ERROR, "Unable to open '%s'", name);
    return(DB_ERROR);
  }

  *size = os_file_get_size(*file);

  ret = os_file_close(*file);
  ut_a(ret);
  return(DB_SUCCESS);
}
```

create mode = OS_FILE_OPEN

check log files

- `srv/srv0start.cc:2230`

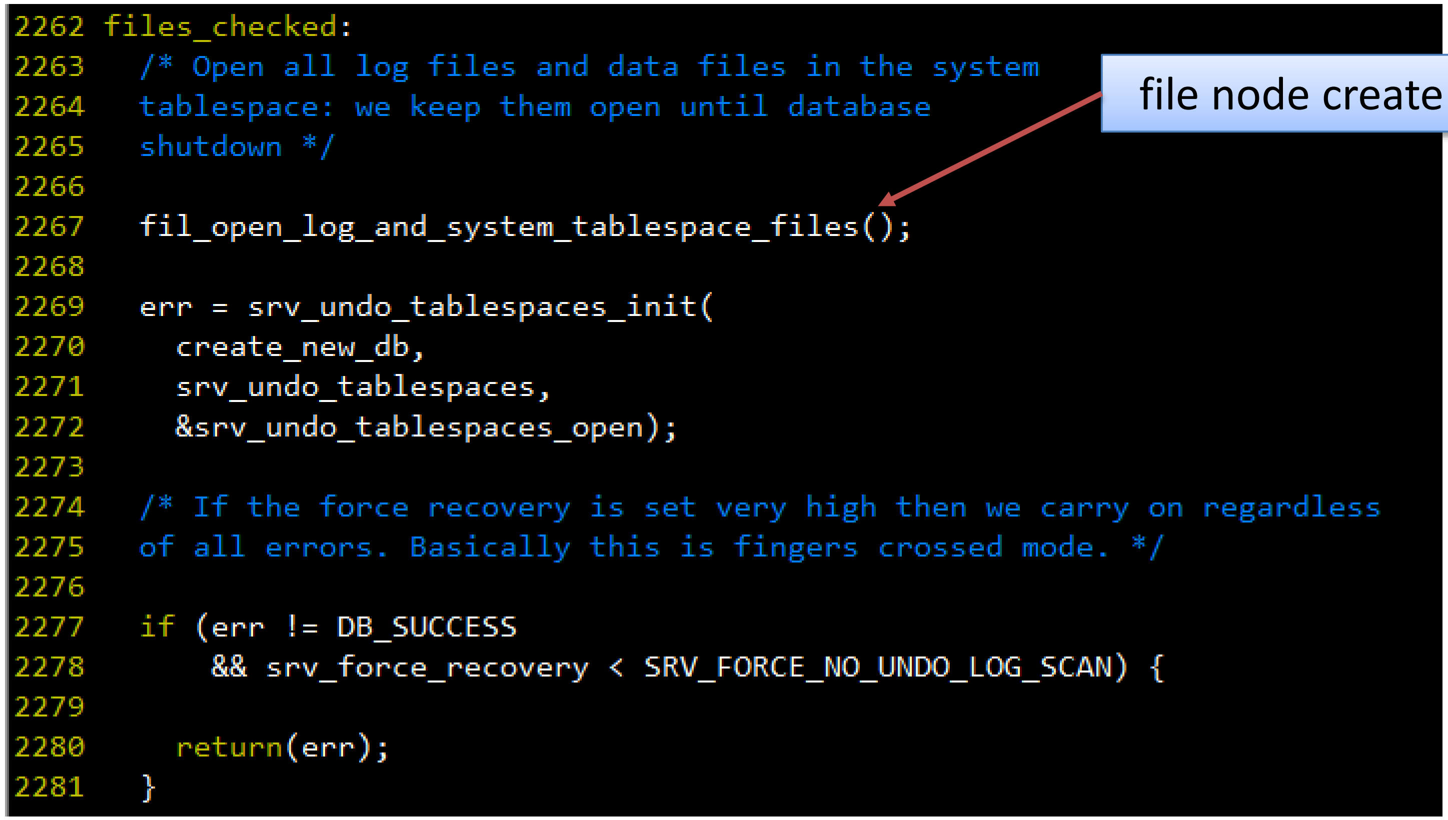


file system create for log files
file node create for log files

```
2226     /* Create the in-memory file space objects. */
2227
2228     sprintf(logfilename + dirnamelen, "ib_logfile%u", 0);
2229
2230     fil_space_create(logfilename,
2231          SRV_LOG_SPACE_FIRST_ID,
2232          fsp_flags_set_page_size(0, UNIV_PAGE_SIZE),
2233          FIL_LOG);
2234
2235     ut_a(fil_validate());
2236
2237     /* srv_log_file_size is measured in pages; if page size is 16KB,
2238     then we have a limit of 64TB on 32 bit systems */
2239     ut_a(srv_log_file_size <= ULINT_MAX);
2240
2241     for (unsigned j = 0; j < i; j++) {
2242       sprintf(logfilename + dirnamelen, "ib_logfile%u", j);
2243
2244       if (!fil_node_create(logfilename,
2245                (ulint) srv_log_file_size,
2246                SRV_LOG_SPACE_FIRST_ID, FALSE)) {
2247         return(DB_ERROR);
2248       }
```

Open log and system table space

- srv/srv0start.cc:2267, fil_open_log_and_system_tablespace_files()

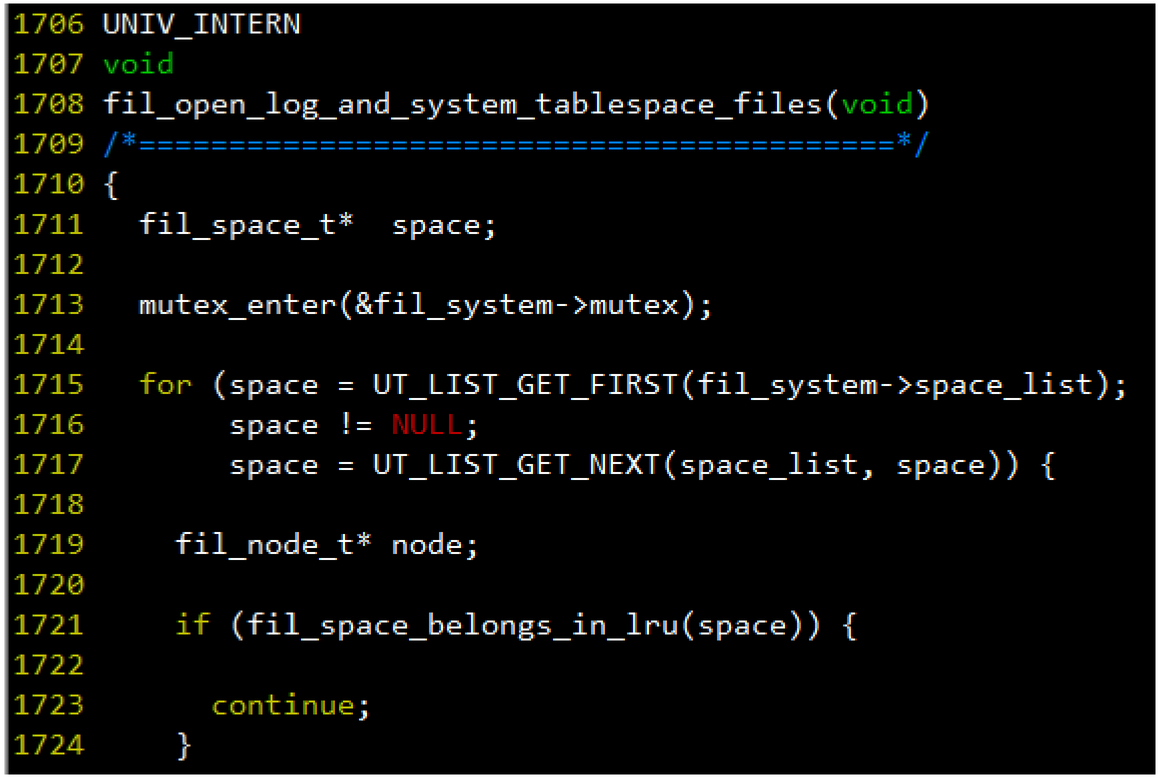


```
2262 files_checked:
2263   /* Open all log files and data files in the system
2264   tablespace: we keep them open until database
2265   shutdown */
2266
2267   fil_open_log_and_system_tablespace_files();
2268
2269   err = srv_undo_tablespaces_init(
2270     create_new_db,
2271     srv_undo_tablespaces,
2272     &srv_undo_tablespaces_open);
2273
2274   /* If the force recovery is set very high then we carry on regardless
2275   of all errors. Basically this is fingers crossed mode. */
2276
2277   if (err != DB_SUCCESS
2278       && srv_force_recovery < SRV_FORCE_NO_UNDO_LOG_SCAN) {
2279
2280     return(err);
2281   }
```

file node create for log files

Open log and system table space

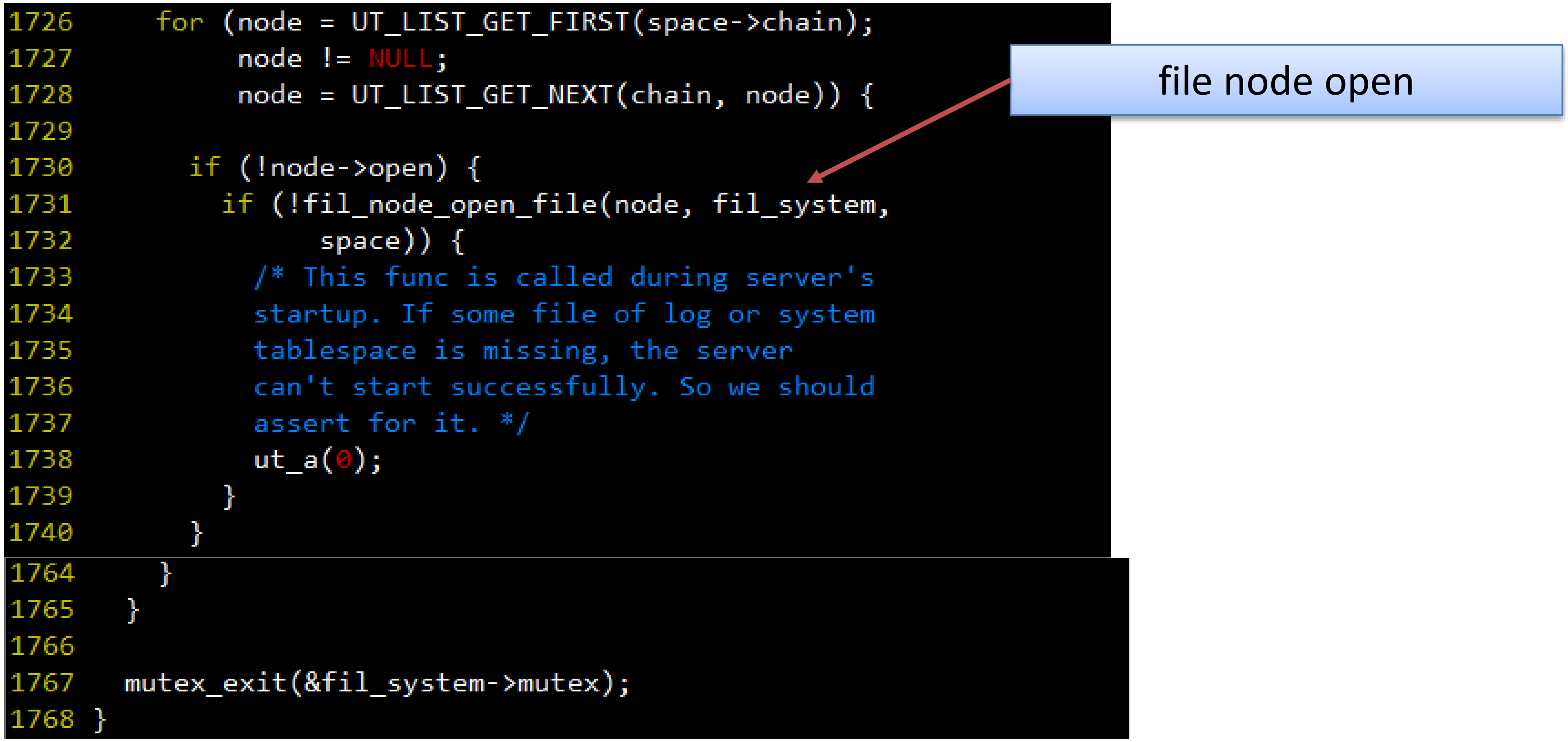
- `fil/fil0fil.cc:1708, fil_open_log_and_system_tablespace_files()`



```
1706 UNIV_INTERN
1707 void
1708 fil_open_log_and_system_tablespace_files(void)
1709 /*=========================================*/
1710 {
1711   fil_space_t*  space;
1712 
1713   mutex_enter(&fil_system->mutex);
1714 
1715   for (space = UT_LIST_GET_FIRST(fil_system->space_list);
1716        space != NULL;
1717        space = UT_LIST_GET_NEXT(space_list, space)) {
1718 
1719     fil_node_t* node;
1720 
1721     if (fil_space_belongs_in_lru(space)) {
1722 
1723       continue;
1724     }
```

Open log and system table space

- `fil/fil0fil.cc:1708, fil_open_log_and_system_tablespace_files()`

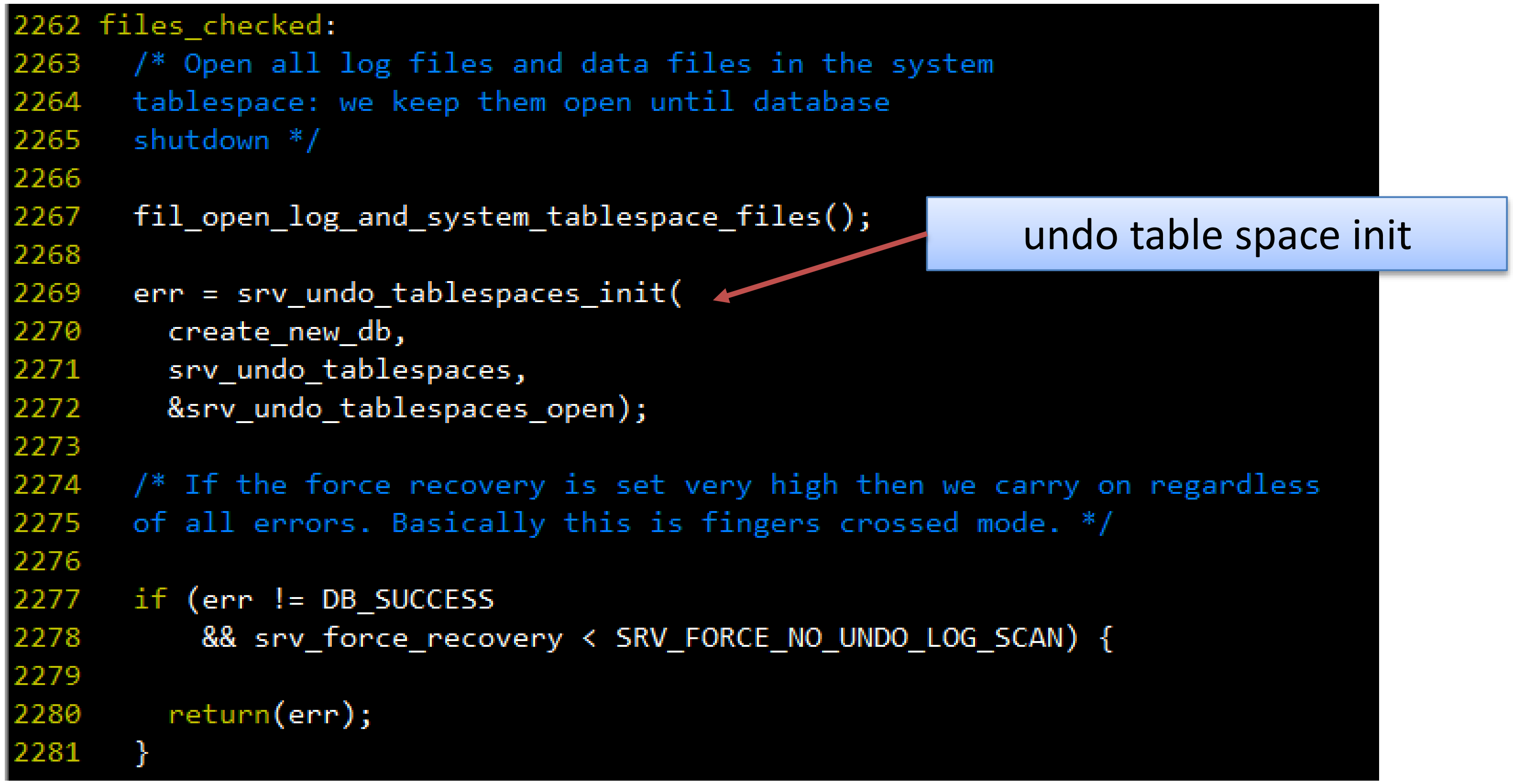


```
    for (node = UT_LIST_GET_FIRST(space->chain);
         node != NULL;
         node = UT_LIST_GET_NEXT(chain, node)) {

      if (!node->open) {
        if (!fil_node_open_file(node, fil_system,
              space)) {
          /* This func is called during server's
          startup. If some file of log or system
          tablespace is missing, the server
          can't start successfully. So we should
          assert for it. */
          ut_a(0);
        }
      }
    }
  }

  mutex_exit(&fil_system->mutex);
}
```

INIT UNDO TABLESPACE

Undo table space init

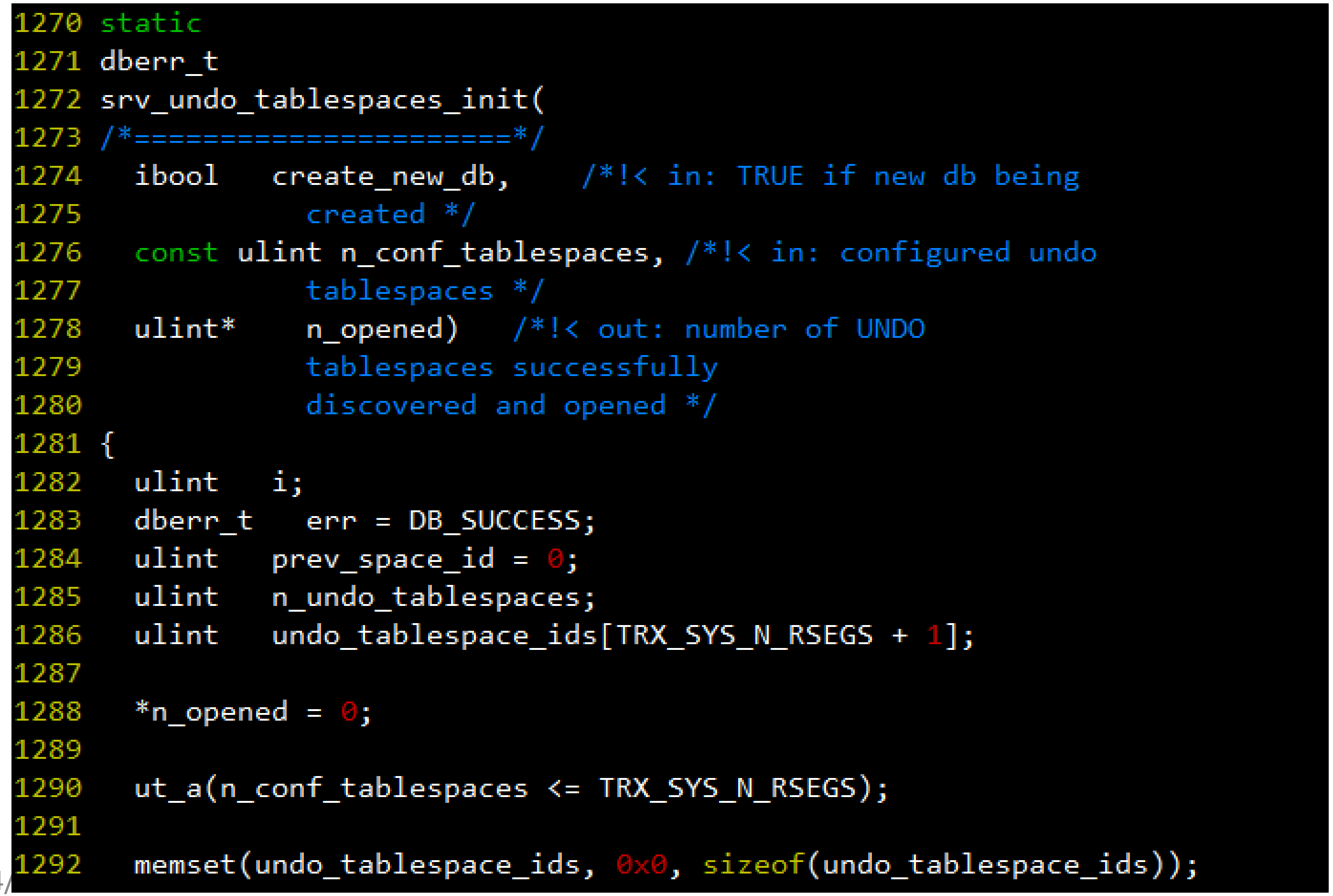
- `srv/srv0start.cc:2269, srv_undo_tablespace_init()`



```
2262 files_checked:
2263   /* Open all log files and data files in the system
2264   tablespace: we keep them open until database
2265   shutdown */
2266
2267   fil_open_log_and_system_tablespace_files();
2268
2269   err = srv_undo_tablespaces_init(
2270     create_new_db,
2271     srv_undo_tablespaces,
2272     &srv_undo_tablespaces_open);
2273
2274   /* If the force recovery is set very high then we carry on regardless
2275   of all errors. Basically this is fingers crossed mode. */
2276
2277   if (err != DB_SUCCESS
2278       && srv_force_recovery < SRV_FORCE_NO_UNDO_LOG_SCAN) {
2279
2280     return(err);
2281   }
```

Undo table space init

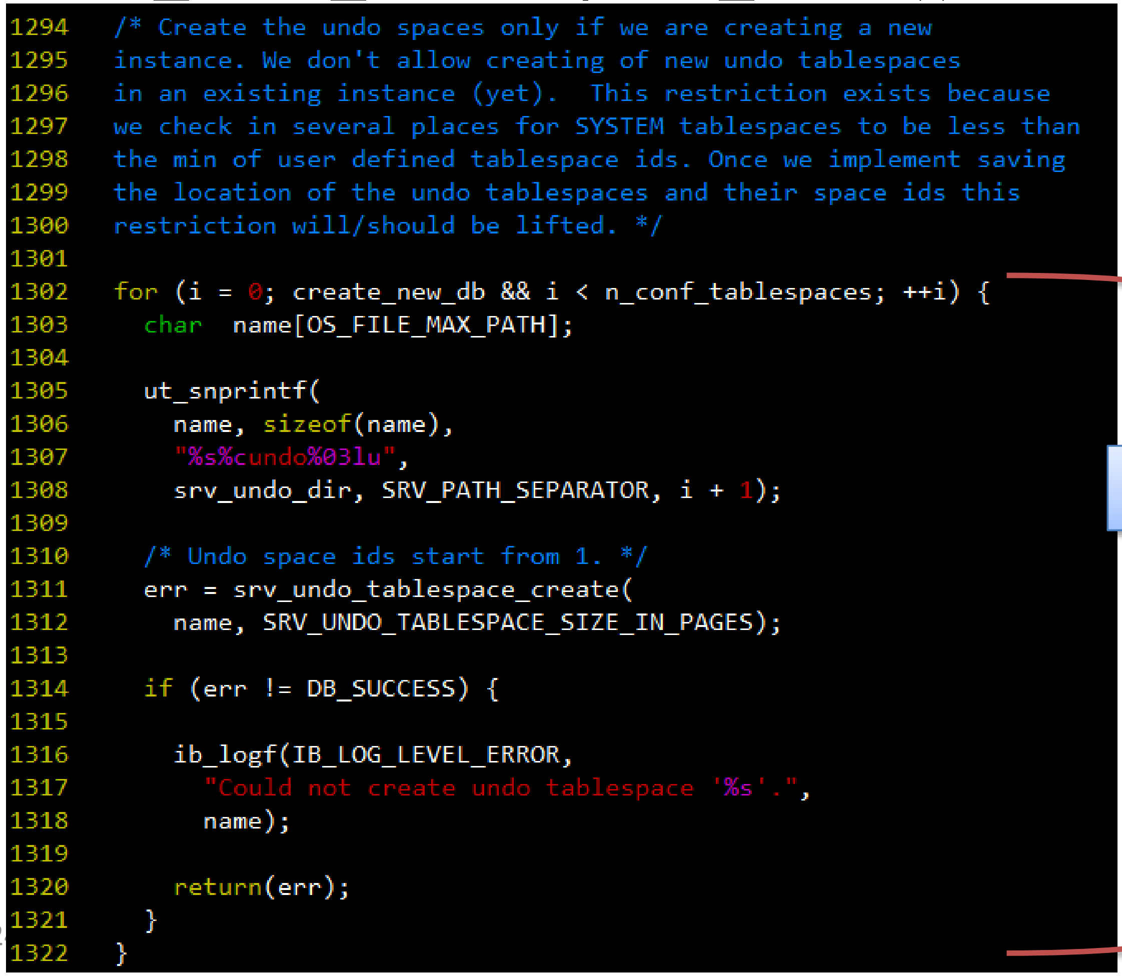
- srv/srv0start.cc:1272, srv_undo_tablespace_init()



```
1270 static
1271 dberr_t
1272 srv_undo_tablespaces_init(
1273 /*=======================*/
1274   ibool    create_new_db,    /*!< in: TRUE if new db being
1275              created */
1276   const ulint n_conf_tablespaces, /*!< in: configured undo
1277              tablespaces */
1278   ulint*     n_opened)   /*!< out: number of UNDO
1279              tablespaces successfully
1280              discovered and opened */
1281 {
1282   ulint    i;
1283   dberr_t    err = DB_SUCCESS;
1284   ulint    prev_space_id = 0;
1285   ulint    n_undo_tablespaces;
1286   ulint    undo_tablespace_ids[TRX_SYS_N_RSEGS + 1];
1287 
1288   *n_opened = 0;
1289 
1290   ut_a(n_conf_tablespaces <= TRX_SYS_N_RSEGS);
1291 
1292   memset(undo_tablespace_ids, 0x0, sizeof(undo_tablespace_ids));
```

Undo table space init

- `srv/srv0start.cc:1272, srv_undo_tablespace_init()`

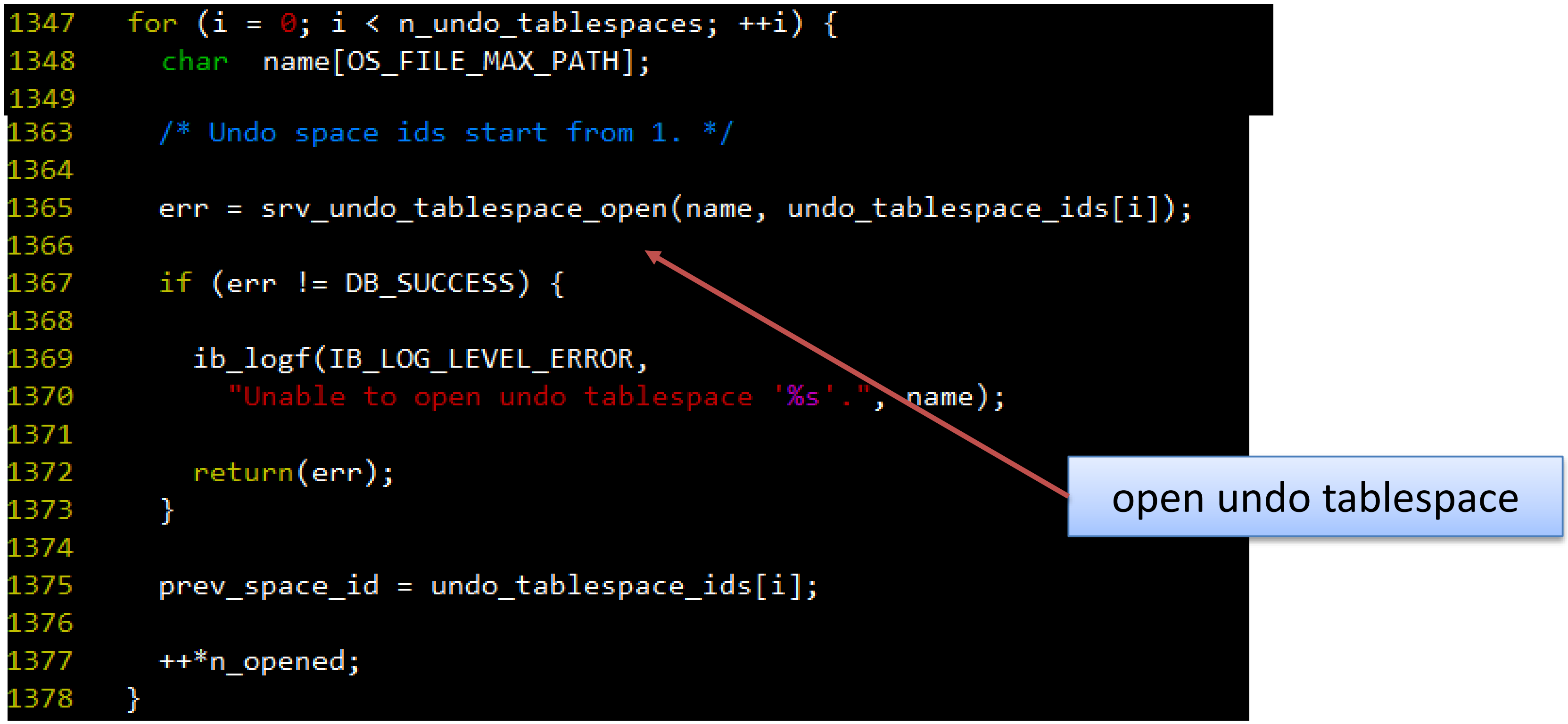


```
/* Create the undo spaces only if we are creating a new
instance. We don't allow creating of new undo tablespaces
in an existing instance (yet).  This restriction exists because
we check in several places for SYSTEM tablespaces to be less than
the min of user defined tablespace ids. Once we implement saving
the location of the undo tablespaces and their space ids this
restriction will/should be lifted. */

for (i = 0; create_new_db && i < n_conf_tablespaces; ++i) {
  char  name[OS_FILE_MAX_PATH];

  ut_snprintf(
    name, sizeof(name),
    "%s%cundo%03lu",
    srv_undo_dir, SRV_PATH_SEPARATOR, i + 1);

  /* Undo space ids start from 1. */
  err = srv_undo_tablespace_create(
    name, SRV_UNDO_TABLESPACE_SIZE_IN_PAGES);

  if (err != DB_SUCCESS) {

    ib_logf(IB_LOG_LEVEL_ERROR,
      "Could not create undo tablespace '%s'.",
      name);

    return(err);
  }
}
```

create new db case, skip

Undo table space init

- `srv/srv0start.cc:1272, srv_undo_tablespace_init()`

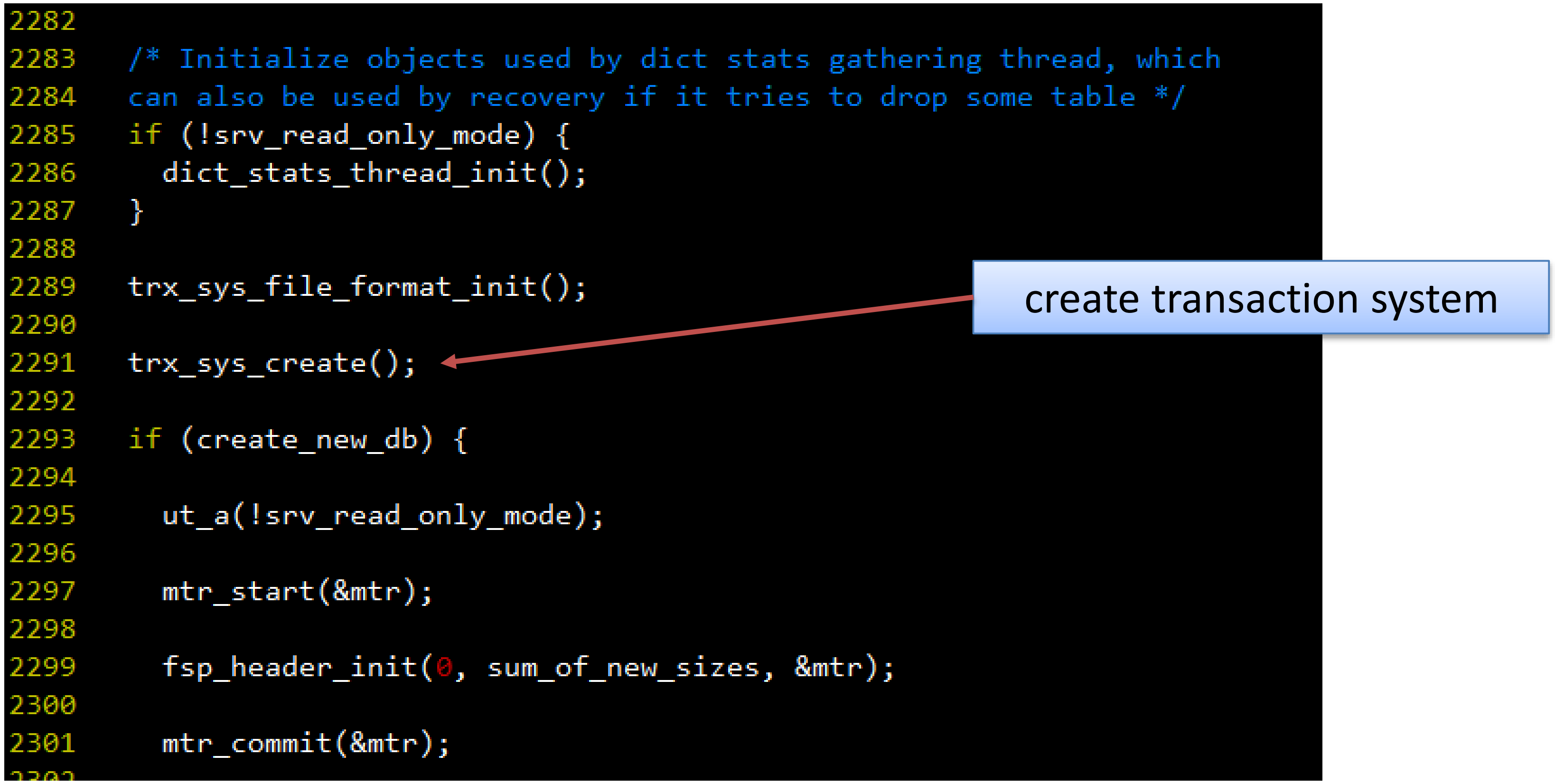


```
  for (i = 0; i < n_undo_tablespaces; ++i) {
    char  name[OS_FILE_MAX_PATH];

    /* Undo space ids start from 1. */

    err = srv_undo_tablespace_open(name, undo_tablespace_ids[i]);

    if (err != DB_SUCCESS) {

      ib_logf(IB_LOG_LEVEL_ERROR,
        "Unable to open undo tablespace '%s'.", name);

      return(err);
    }

    prev_space_id = undo_tablespace_ids[i];

    ++*n_opened;
  }
```

START TRANSACTION SYSTEM

Create transaction system

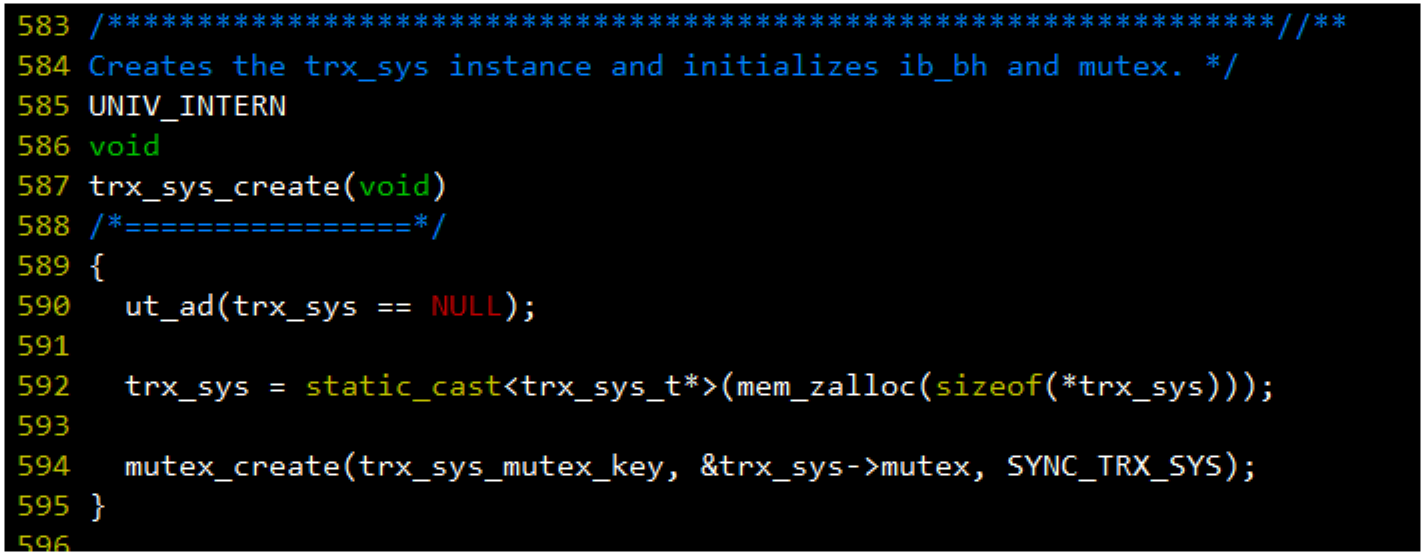
- srv/srv0start.cc:2291, trx_sys_create()



```
2282
2283   /* Initialize objects used by dict stats gathering thread, which
2284   can also be used by recovery if it tries to drop some table */
2285   if (!srv_read_only_mode) {
2286     dict_stats_thread_init();
2287   }
2288
2289   trx_sys_file_format_init();
2290
2291   trx_sys_create();
2292
2293   if (create_new_db) {
2294
2295     ut_a(!srv_read_only_mode);
2296
2297     mtr_start(&mtr);
2298
2299     fsp_header_init(0, sum_of_new_sizes, &mtr);
2300
2301     mtr_commit(&mtr);
```

Create transaction system

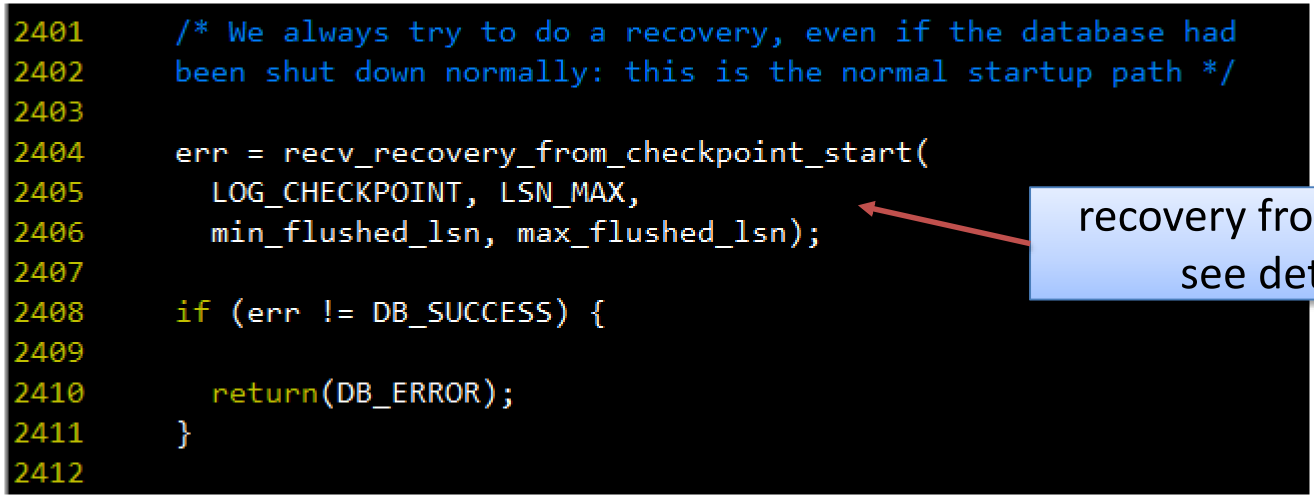
- `trx/trx0sys.cc:587, trx_sys_create()`



```
583 /********************************************************************//**
584 Creates the trx_sys instance and initializes ib_bh and mutex. */
585 UNIV_INTERN
586 void
587 trx_sys_create(void)
588 /*================*/
589 {
590   ut_ad(trx_sys == NULL);
591
592   trx_sys = static_cast<trx_sys_t*>(mem_zalloc(sizeof(*trx_sys)));
593
594   mutex_create(trx_sys_mutex_key, &trx_sys->mutex, SYNC_TRX_SYS);
595 }
596
```

Do recovery

- srv/srv0start.cc:2404, recv_recovery_from_checkpoint_start()

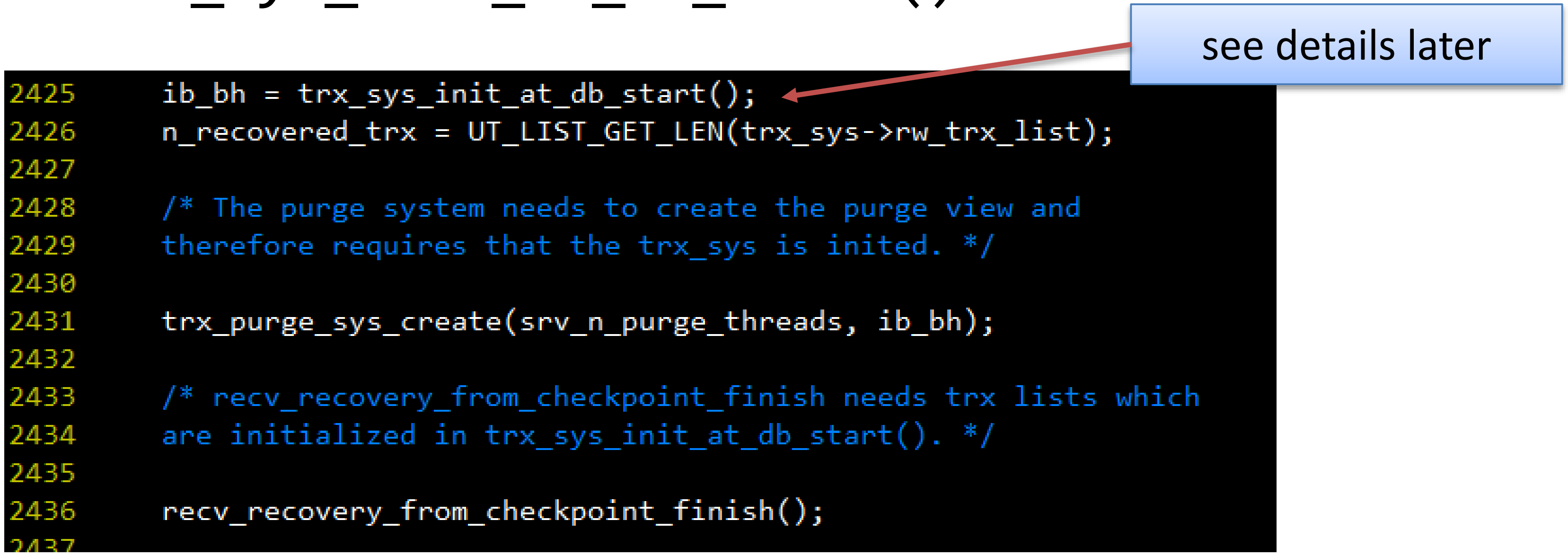


```
2401      /* We always try to do a recovery, even if the database had
2402      been shut down normally: this is the normal startup path */
2403
2404      err = recv_recovery_from_checkpoint_start(
2405        LOG_CHECKPOINT, LSN_MAX,
2406        min_flushed_lsn, max_flushed_lsn);
2407
2408      if (err != DB_SUCCESS) {
2409
2410        return(DB_ERROR);
2411      }
2412
```

recovery from checkpoint: see details later

Initialize transaction system

- srv/srv0start.cc:2425, trx_sys_init_at_db_start()

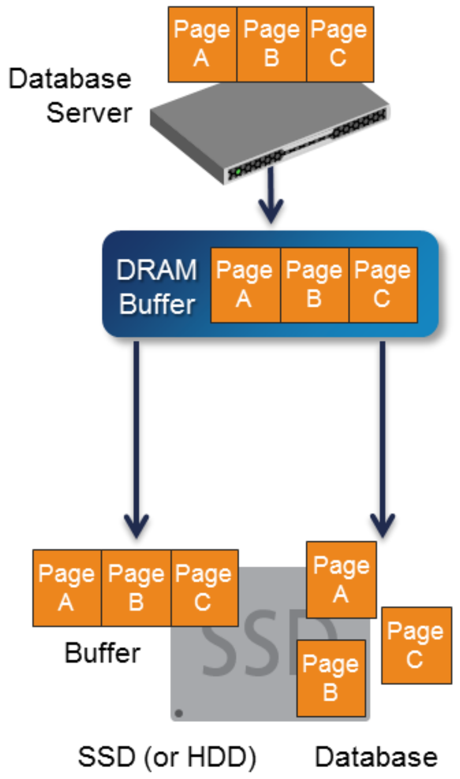


```
ib_bh = trx_sys_init_at_db_start();
n_recovered_trx = UT_LIST_GET_LEN(trx_sys->rw_trx_list);

/* The purge system needs to create the purge view and
therefore requires that the trx_sys is inited. */

trx_purge_sys_create(srv_n_purge_threads, ib_bh);

/* recv_recovery_from_checkpoint_finish needs trx lists which
are initialized in trx_sys_init_at_db_start(). */

recv_recovery_from_checkpoint_finish();
```

CREATE DOUBLE WRITE BUFFER

Double write buffer

Traditional MySQL Writes



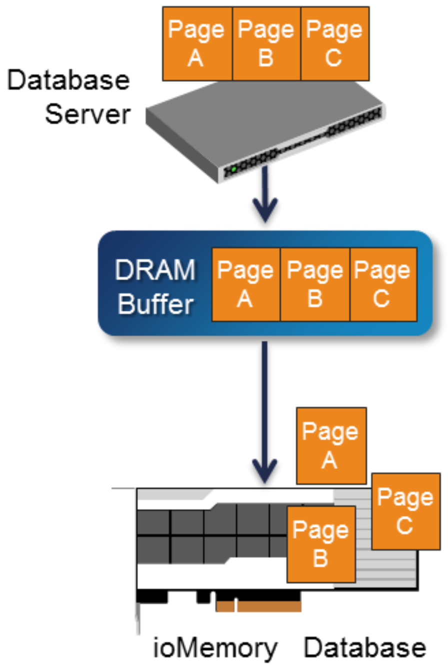
1 Application initiates updates to pages A, B, and C.

2 MySQL copies updated pages to memory buffer.

3 MySQL writes to double-write buffer on the media.

4 Once step 3 is acknowledged, MySQL writes the updates to the actual tablespace.

MySQL with Atomic Writes



1 Application initiates updates to pages A, B, and C.

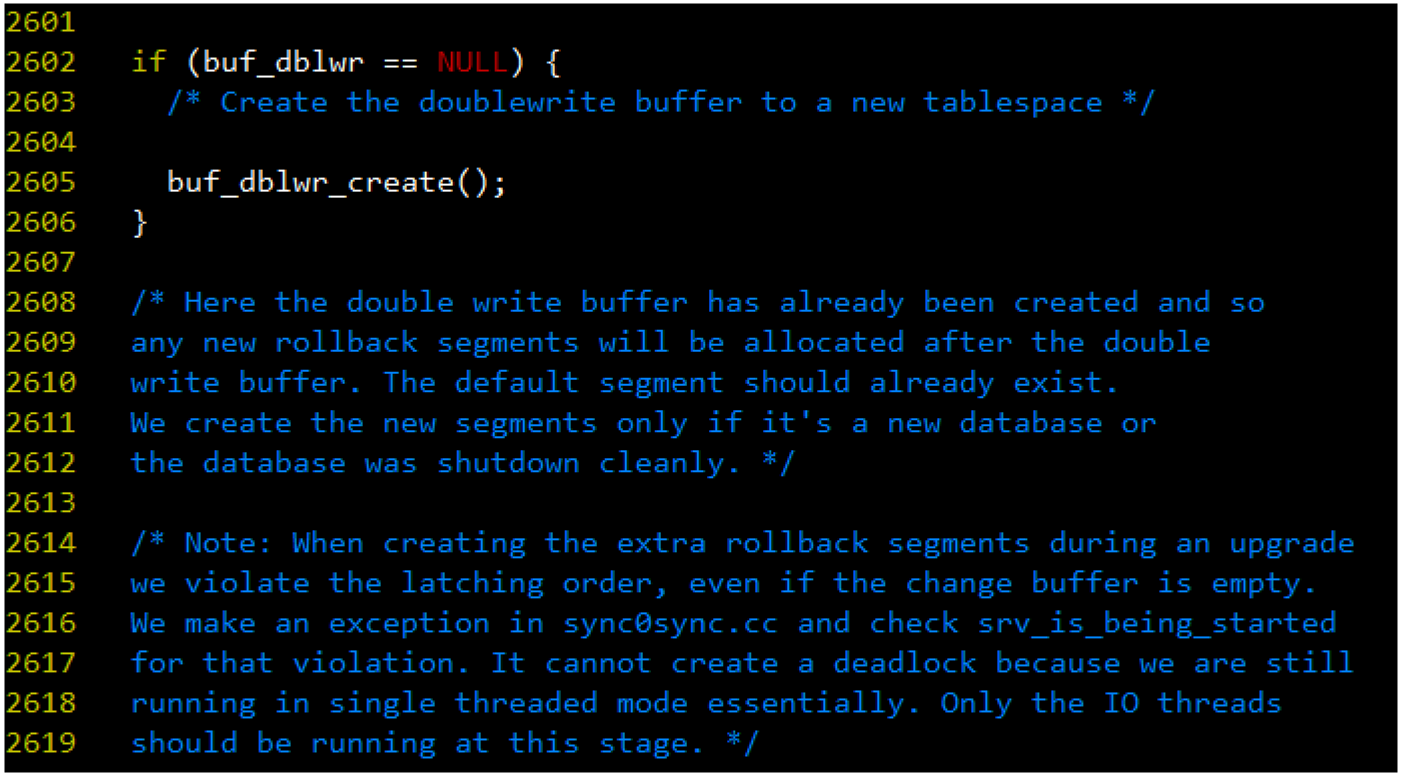
2 MySQL copies updated pages to memory buffer.

3 MySQL writes to actual tablespace, bypassing the double-write buffer step due to inherent atomicity guaranteed by the (intelligent device).

http://www.fusionio.com/load/-media-/29ek9p/imagesLibrary/03_atomic_writes.png

double write create

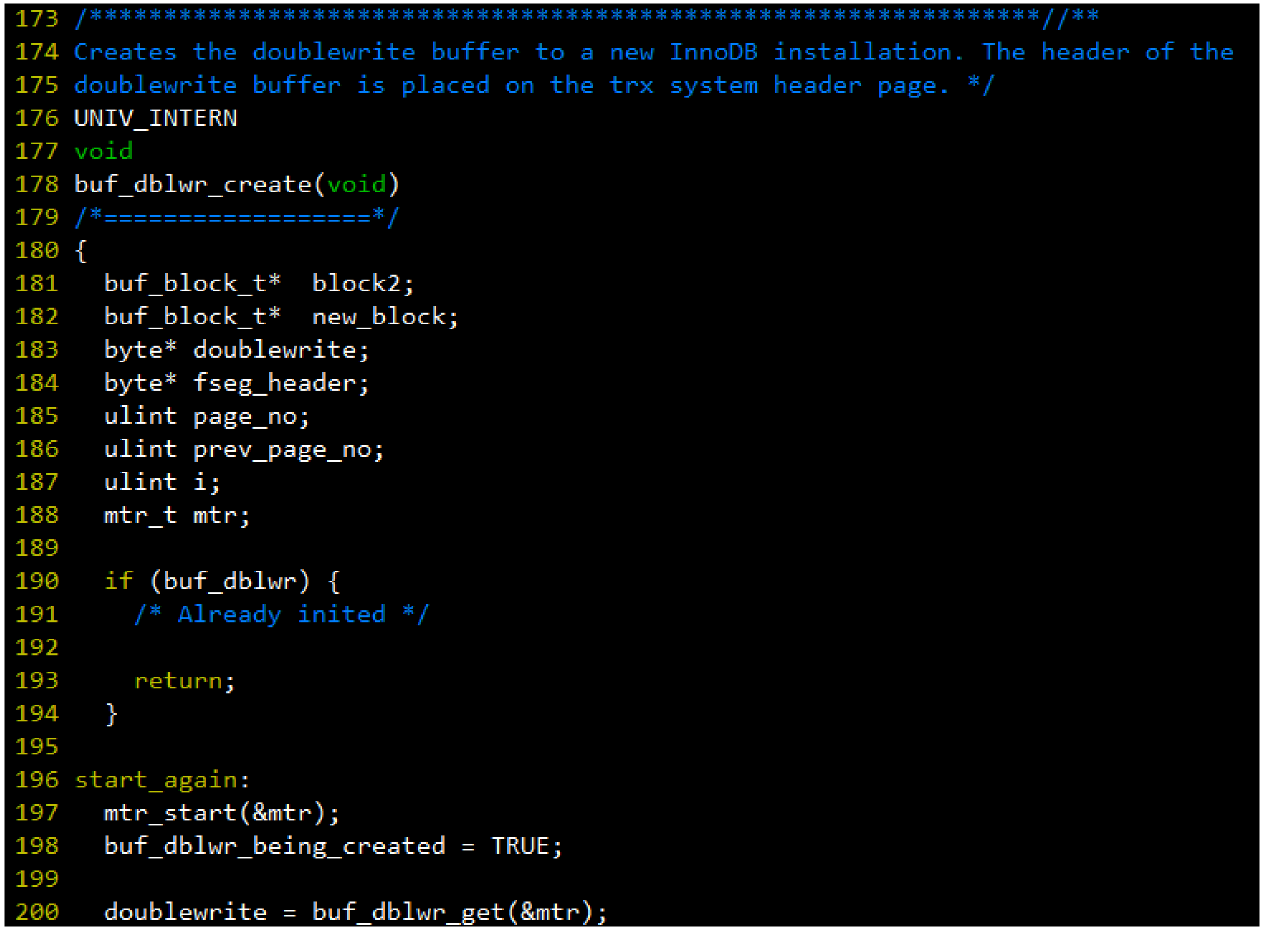
- `srv/srv0start.cc:2605, buf_dblwr_create()`



```
2601
2602   if (buf_dblwr == NULL) {
2603     /* Create the doublewrite buffer to a new tablespace */
2604
2605     buf_dblwr_create();
2606   }
2607
2608   /* Here the double write buffer has already been created and so
2609   any new rollback segments will be allocated after the double
2610   write buffer. The default segment should already exist.
2611   We create the new segments only if it's a new database or
2612   the database was shutdown cleanly. */
2613
2614   /* Note: When creating the extra rollback segments during an upgrade
2615   we violate the latching order, even if the change buffer is empty.
2616   We make an exception in sync0sync.cc and check srv_is_being_started
2617   for that violation. It cannot create a deadlock because we are still
2618   running in single threaded mode essentially. Only the IO threads
2619   should be running at this stage. */
```

double write create

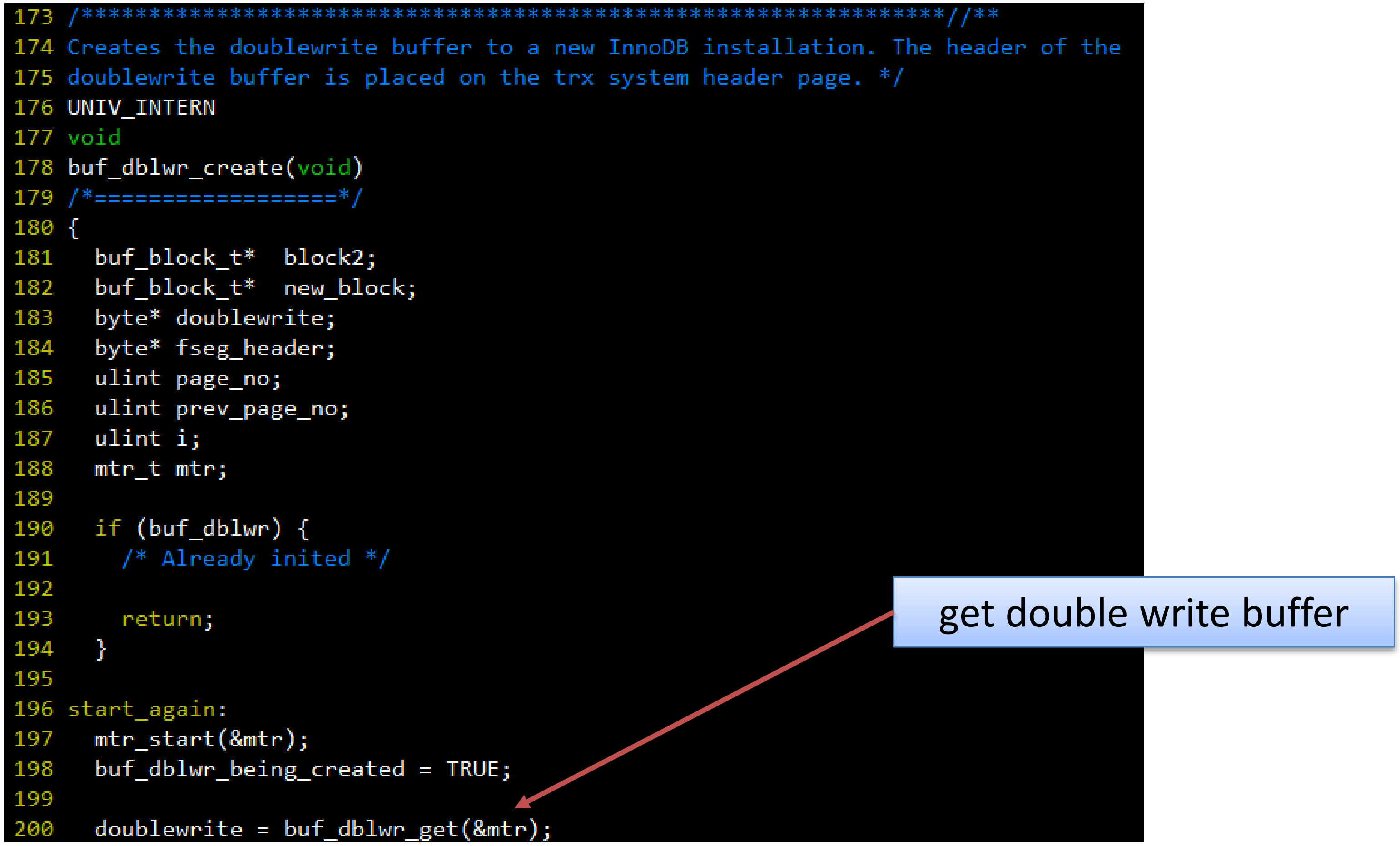
- buf/buf0dblwr.cc:178, buf_dblwr_create()



```
173 /********************************************************************//**
174 Creates the doublewrite buffer to a new InnoDB installation. The header of the
175 doublewrite buffer is placed on the trx system header page. */
176 UNIV_INTERN
177 void
178 buf_dblwr_create(void)
179 /*==================*/
180 {
181   buf_block_t*  block2;
182   buf_block_t*  new_block;
183   byte* doublewrite;
184   byte* fseg_header;
185   ulint page_no;
186   ulint prev_page_no;
187   ulint i;
188   mtr_t mtr;
189 
190   if (buf_dblwr) {
191     /* Already inited */
192 
193     return;
194   }
195 
196 start_again:
197   mtr_start(&mtr);
198   buf_dblwr_being_created = TRUE;
199 
200   doublewrite = buf_dblwr_get(&mtr);
```

double write create

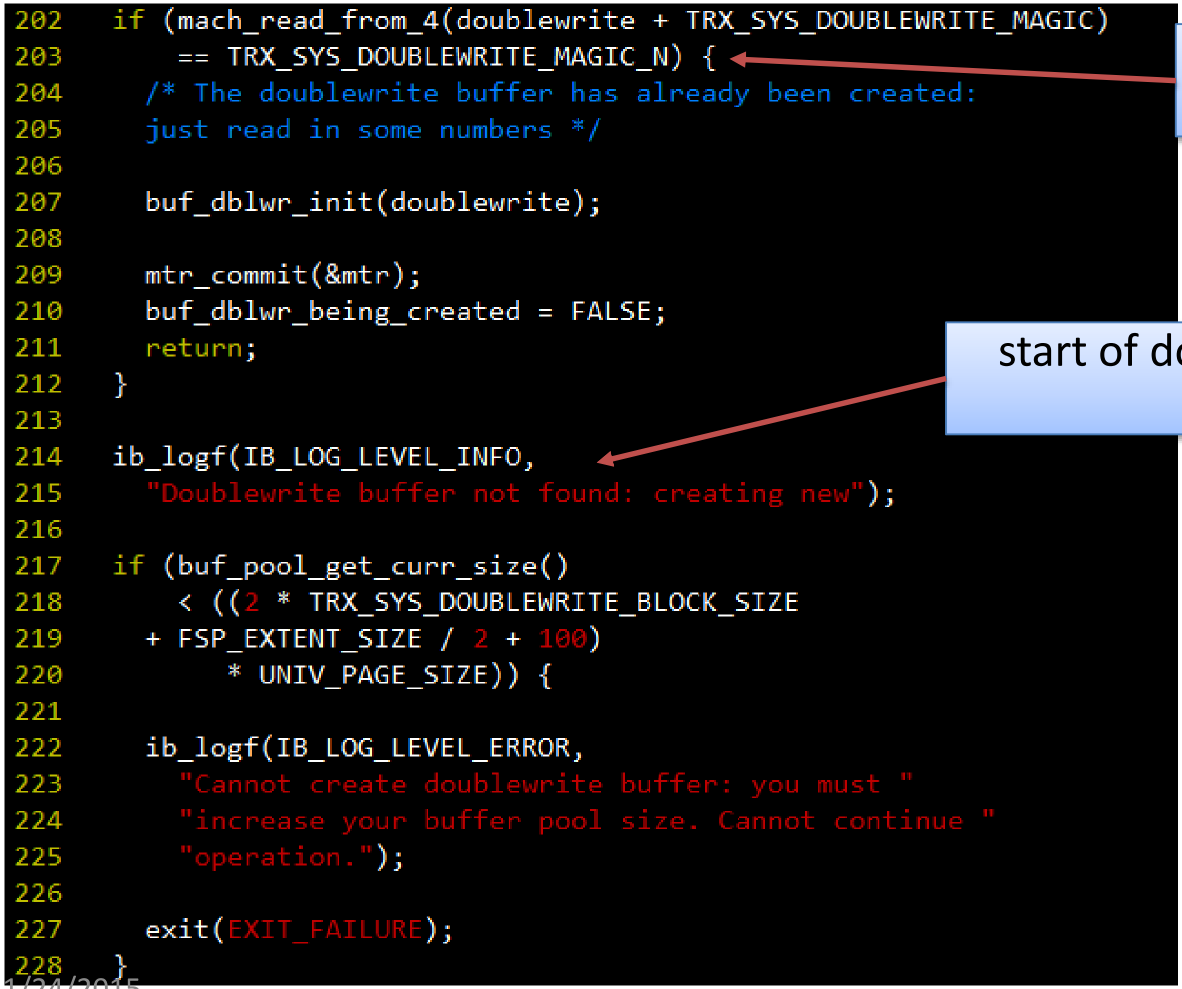
- buf/buf0dblwr.cc:178, buf_dblwr_create()



```
/********************************************************************//**
Creates the doublewrite buffer to a new InnoDB installation. The header of the
doublewrite buffer is placed on the trx system header page. */
UNIV_INTERN
void
buf_dblwr_create(void)
/*==================*/
{
  buf_block_t*  block2;
  buf_block_t*  new_block;
  byte* doublewrite;
  byte* fseg_header;
  ulint page_no;
  ulint prev_page_no;
  ulint i;
  mtr_t mtr;

  if (buf_dblwr) {
    /* Already inited */

    return;
  }

start_again:
  mtr_start(&mtr);
  buf_dblwr_being_created = TRUE;

  doublewrite = buf_dblwr_get(&mtr);
```

double write create

- buf/buf0dblwr.cc:178, buf_dblwr_create()



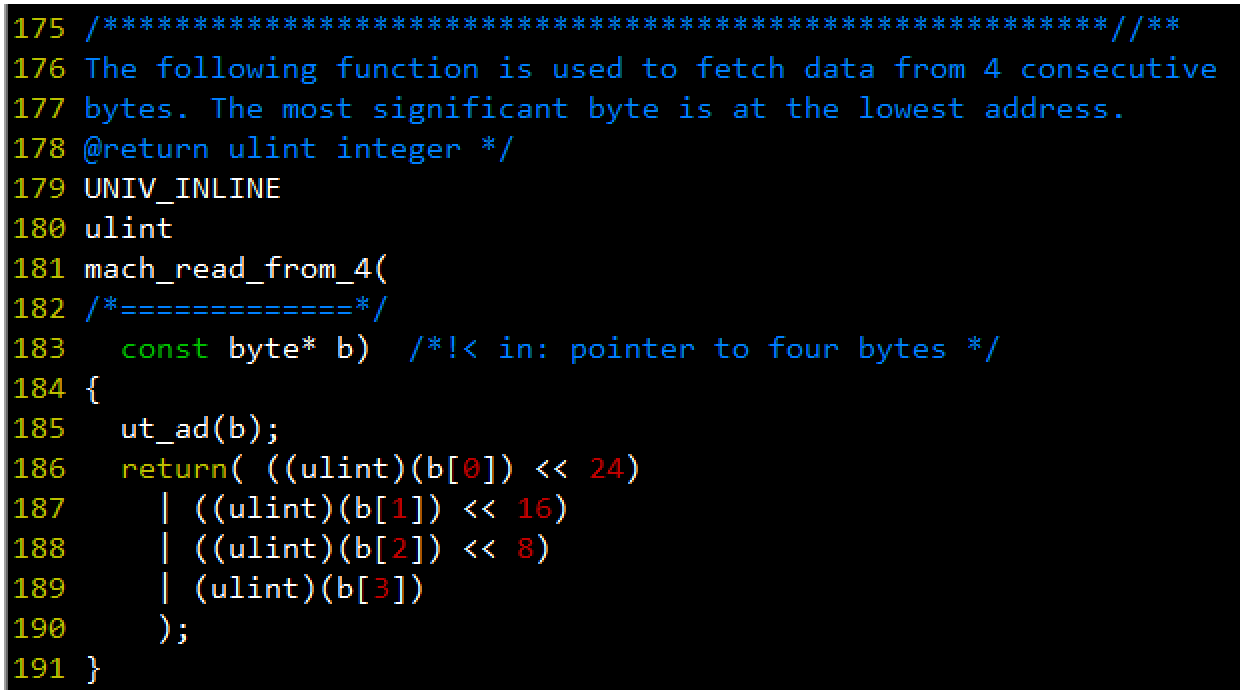
```
	if (mach_read_from_4(doublewrite + TRX_SYS_DOUBLEWRITE_MAGIC)
	    == TRX_SYS_DOUBLEWRITE_MAGIC_N) {
		/* The doublewrite buffer has already been created:
		just read in some numbers */

		buf_dblwr_init(doublewrite);

		mtr_commit(&mtr);
		buf_dblwr_being_created = FALSE;
		return;
	}

	ib_logf(IB_LOG_LEVEL_INFO,
		"Doublewrite buffer not found: creating new");

	if (buf_pool_get_curr_size()
	    < ((2 * TRX_SYS_DOUBLEWRITE_BLOCK_SIZE
		+ FSP_EXTENT_SIZE / 2 + 100)
	       * UNIV_PAGE_SIZE)) {

		ib_logf(IB_LOG_LEVEL_ERROR,
			"Cannot create doublewrite buffer: you must "
			"increase your buffer pool size. Cannot continue "
			"operation.");

		exit(EXIT_FAILURE);
	}
```

doublewrite buffer already created

start of doublewrite buffer creation: skip

Note

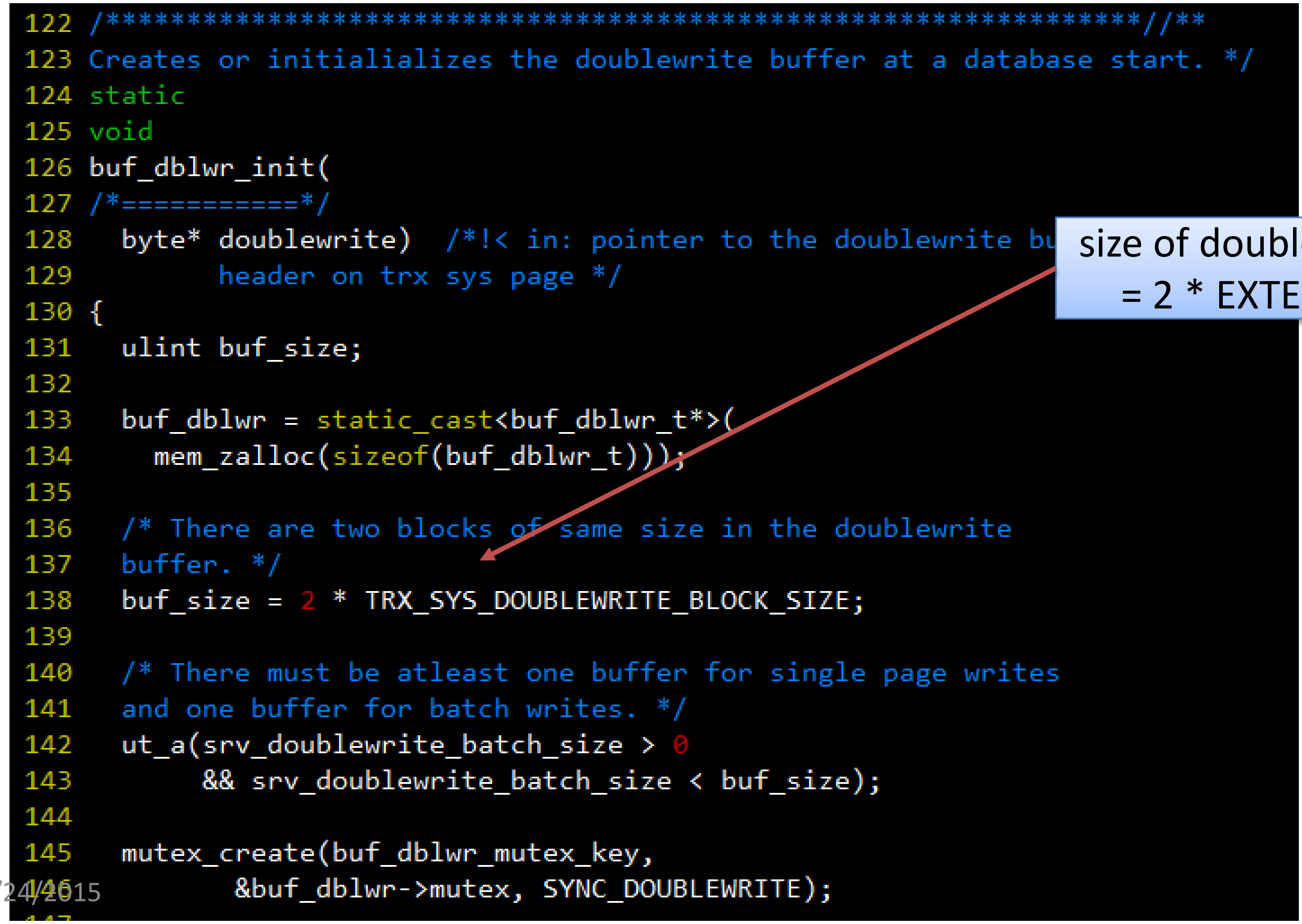
- marc_read_from_X
  - read X bytes
  - considering architecture specific endian.



```
/**********************************************************//**
The following function is used to fetch data from 4 consecutive
bytes. The most significant byte is at the lowest address.
@return ulint integer */
UNIV_INLINE
ulint
mach_read_from_4(
/*=============*/
  const byte* b)  /*!< in: pointer to four bytes */
{
  ut_ad(b);
  return( ((ulint)(b[0]) << 24)
    | ((ulint)(b[1]) << 16)
    | ((ulint)(b[2]) << 8)
    | (ulint)(b[3])
    );
}
```

double write create

- buf/buf0dblwr.cc:126, buf_dblwr_init()

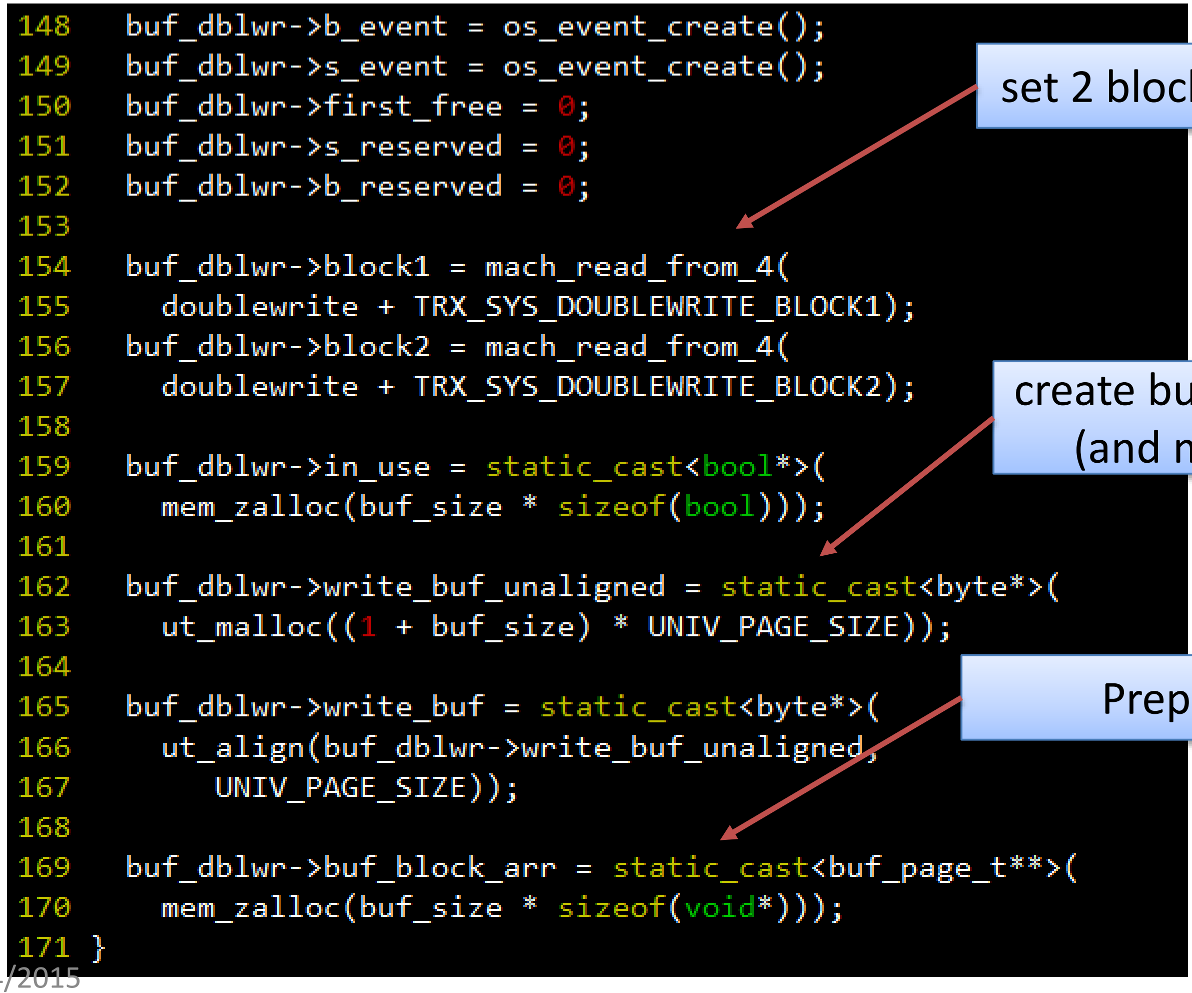


```
122 /********************************************************************//**
123 Creates or initializes the doublewrite buffer at a database start. */
124 static
125 void
126 buf_dblwr_init(
127 /*===========*/
128   byte* doublewrite)  /*!< in: pointer to the doublewrite bu
129         header on trx sys page */
130 {
131   ulint buf_size;
132 
133   buf_dblwr = static_cast<buf_dblwr_t*>(
134     mem_zalloc(sizeof(buf_dblwr_t)));
135 
136   /* There are two blocks of same size in the doublewrite
137   buffer. */
138   buf_size = 2 * TRX_SYS_DOUBLEWRITE_BLOCK_SIZE;
139 
140   /* There must be atleast one buffer for single page writes
141   and one buffer for batch writes. */
142   ut_a(srv_doublewrite_batch_size > 0
143       && srv_doublewrite_batch_size < buf_size);
144 
145   mutex_create(buf_dblwr_mutex_key,
146          &buf_dblwr->mutex, SYNC_DOUBLEWRITE);
```

size of doublewrite buffer
= 2 * EXTENTS (1MB)

double write create

- buf/buf0dblwr.cc:126, buf_dblwr_init()



```
  buf_dblwr->b_event = os_event_create();
  buf_dblwr->s_event = os_event_create();
  buf_dblwr->first_free = 0;
  buf_dblwr->s_reserved = 0;
  buf_dblwr->b_reserved = 0;

  buf_dblwr->block1 = mach_read_from_4(
    doublewrite + TRX_SYS_DOUBLEWRITE_BLOCK1);
  buf_dblwr->block2 = mach_read_from_4(
    doublewrite + TRX_SYS_DOUBLEWRITE_BLOCK2);

  buf_dblwr->in_use = static_cast<bool*>(
    mem_zalloc(buf_size * sizeof(bool)));

  buf_dblwr->write_buf_unaligned = static_cast<byte*>(
    ut_malloc((1 + buf_size) * UNIV_PAGE_SIZE));

  buf_dblwr->write_buf = static_cast<byte*>(
    ut_align(buf_dblwr->write_buf_unaligned,
       UNIV_PAGE_SIZE));

  buf_dblwr->buf_block_arr = static_cast<buf_page_t**>(
    mem_zalloc(buf_size * sizeof(void*)));
}
```

set 2 blocks of double write header

create buffer frame for doublewrite (and make it page sized align)

Prepare buf block array

Q&A

- Any Questions ?

Reference

- Source Code : MySQL Community Server 5.6.21
- Transaction Model and Locking
  - https://dev.mysql.com/doc/refman/5.6/en/innodb-transaction-model.html
  - http://dev.cs.uni-magdeburg.de/db/mysql/InnoDB-transaction-model.html