SBLLmalloc Reference Manual

1.0

Generated by Doxygen 1.4.7

Fri Apr 23 19:57:27 2010

Contents

1 SBLLmalloc

Author:

Susmit Biswas

Memory size has long limited large-scale applications on high-performance computing (HPC) systems. Increasing core counts per chip and power density constraints, which limit the number of DIMMs per node, have exacerbated this problem. Mechanisms to manage memory usage —preferably transparently— more efficiently could increase effective DRAM capacity and, thus, the benefit of multicore nodes for HPC systems.

MPI application processes often exhibit significant data similarity. These data regions occupy multiple physical locations within a multicore node and thus offer a potential savings in memory capacity. These data, primarily residing in heap, are quite dynamic, which makes them difficult to manage statically.

SBLLmalloc is a memory allocation library that automatically identifies the replicated memory blocks and merges them into a single copy. SBLLmalloc does not require application or OS changes since we implement it as a user-level library that can be linked at runtime. Overall, we find that SBLLmalloc reduces the memory footprint of a wide range of MPI applications by 37.03% on average and up to 60.87%. Further, supports problem sizes for AMG and IRS over 18.5% and 21.6% larger than using standard memory management techniques, thus significantly increasing effective system size.

In the following, the usage of the library is described. In the *run* directory you will find a script called submitjob.sh. Modify it to your need. Examples of using this script is shown at the end of this section.

Name	Default	Description
PROFILE_MODE	1	profiling mode? 0: no profiling 1: create profiles 2: use profile for merging (EXPERIMENTAL)
MERGE_METRIC	1	merge metric? 0: disabled 1: alloc_frequency 2: threshold (Recommended) 3: buffered (Experimental)
MALLOC_MERGE_FREQ	1000	frequency for frequency based merge
MIN_MEM_TH	10	threshold for threshold based merge
ENABLE_BACKTRACE	0	enable backtrace? 1: enabled 0: disabled Used for finding the source location that allocated the merged page
SEM_KEY	1234	semaphore key
NOT_MPI_APP	0	define 1 if this does not call MPI_Init(). You need to modify the code. Please read the TODO list.

Table 1: Environment Variables

This file describes the usage of the heap merging library. In order to use merge capability, you need to set some environment variables which triggers merge operations. If the original commandline is

srun -nx -Ny <commandline>,

you will need to change it to the following command.

```
MPIRUN="srun -nx -Ny" TH=<threshold value> ENV_VAR1=<value1> \
[ENV_VAR2=<value2> ...] $TOPDIR/run/submitjob_v2.sh <commandline>
```

At the end of the section you will find a concrete example.

If address space layout randomization (ASLR) is enabled, for x86_64 machines run the program with setarch x86_64 –3gb -R. For i∗86 based servers, use i386 instead of x86_64. In most of the HPC clusters ASLR is disabled, so you may need to use `setarch` at all. Check the file `/proc/sys/kernel/randomize_va_space` to see if the value set 0 to disable ASLR.

In order to set parameters in the library you need to set some environment variables which are listed in Table 1

Example use:

```
bash$ cat run.amg.sh
```

```
#!/bin/bash
COMMANDLINE="./amg2006 -P 2 2 2 -n 80 80 80 -r 20 20 20 -27pt"
MPIRUN="srun -n8 -N1" TH=200 MERGE_METHOD=2 ENABLE_BACKTRACE=0 PROF=0 \
~/LLNL_WORK/ptmalloc/run/submitjob_v2.sh $COMMANDLINE
bash$ ./run.amg.sh
salloc: Granted job allocation 1006112
  Laplace type problem with a 27-point stencil
  (nx_global, ny_global, nz_global) = (160, 160, 160)
  (Px, Py, Pz) = (2, 2, 2)
   ...
```

If you get a fault due to mmap cap, issue the following command to change the default max map count to 512K. In default system configuration it is set as 64K. Check the value with the following command.

sysctl vm.max_map_count

To change the limit please issue the following command.

`sudo sysctl vm.max_map_count=$`((512∗1024))

2 SBLLmalloc Class Index

2.1 SBLLmalloc Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 SBLLmalloc File Index

3.1 SBLLmalloc File List

Here is a list of all files with brief descriptions:

4 SBLLmalloc Page Index

4.1 SBLLmalloc Related Pages

Here is a list of all related documentation pages:

5 SBLLmalloc Class Documentation

5.1 AVLTreeData Struct Reference

Data for an AVL tree.

Public Attributes

- AVLTreeNode ∗ root
- AVLComparator comparator
- int size

5.1.1 Detailed Description

Data for an AVL tree.

5.1.2 Member Data Documentation

5.1.2.1 AVLComparator AVLTreeData::comparator

The comparator function

5.1.2.2 AVLTreeNode∗ AVLTreeData::root

root of AVL tree

5.1.2.3 int AVLTreeData::size

size of the avl tree

The documentation for this struct was generated from the following file:

- AVL.h

5.2 AVLTreeNode Struct Reference

Structure to represent an AVL tree node.

Public Attributes

- const void ∗ key
- void ∗ value
- int height
- AVLTreeNode ∗ left
- AVLTreeNode ∗ right
- int dirty
- uintptr_t creator
- void ∗ callStack [MAX_STACK_DEPTH]

5.2.1 Detailed Description

Structure to represent an AVL tree node.

5.2.2 Member Data Documentation

5.2.2.1 void∗ AVLTreeNode::callStack[MAX_STACK_DEPTH]

call stack when the malloc was called

5.2.2.2 uintptr_t AVLTreeNode::creator

address of the code block that allocated this block

5.2.2.3 int AVLTreeNode::dirty

indicates if it is modified since last merge

5.2.2.4 int AVLTreeNode::height

height of the avl tree

5.2.2.5 const void∗ AVLTreeNode::key

used for comparison

5.2.2.6 struct AVLTreeNode∗ AVLTreeNode::left

left child

5.2.2.7 struct AVLTreeNode∗ AVLTreeNode::right

right child

5.2.2.8 void∗ AVLTreeNode::value

stored value

The documentation for this struct was generated from the following file:

- AVL.h

5.3 commandLineArgument Struct Reference

Structure for parsing arguments.

Public Attributes

- const char ∗ name
- int ∗ variable
- int default_val
- const char ∗ description

5.3.1 Detailed Description

Structure for parsing arguments.

5.3.2 Member Data Documentation

5.3.2.1 int commandLineArgument::default_val

Default value

5.3.2.2 const char∗ commandLineArgument::description

Description of the argument

5.3.2.3 const char∗ commandLineArgument::name

Name of the argument

5.3.2.4 int∗ commandLineArgument::variable

Address of the variable to store of argument

The documentation for this struct was generated from the following file:

- SharedHeap.h

5.4 MemStatStruct Struct Reference

The structure for storing merge info.

Public Attributes

- long int totalPrivateMem
- long int totalPtmallocMem
- long int totalZeroMem
- long int totalSharedMem

- long int totalUnmergedMem
- long int totalMergedMem
- int mergeTimeinMicrosec

5.4.1 Detailed Description

The structure for storing merge info.

5.4.2 Member Data Documentation

5.4.2.1 int MemStatStruct::mergeTimeinMicrosec

Time used for merging in microsecond

5.4.2.2 long int MemStatStruct::totalMergedMem

Memory footprint with merging enabled

5.4.2.3 long int MemStatStruct::totalPrivateMem

Total memory as private pages

5.4.2.4 long int MemStatStruct::totalPtmallocMem

Total memory used by internal allocator

5.4.2.5 long int MemStatStruct::totalSharedMem

Total shared memory usage

5.4.2.6 long int MemStatStruct::totalUnmergedMem

Memory footprint if merging is disabled

5.4.2.7 long int MemStatStruct::totalZeroMem

Total zero memory in current process

The documentation for this struct was generated from the following file:

- SharedHeap.h

5.5 MicroTimer Class Reference

Collects fine grain timing stats using gettimeofday().

Public Member Functions

- virtual void Start ()

  starts the timer
- virtual void Stop ()

  stops the timer
- virtual unsigned long GetDiff () const

  computes time

Private Member Functions

- void ComputeDiff ()

  stores (end time - start time)

Private Attributes

- timeval start_
- timeval end_
- timeval diff_

Friends

- std::ostream & operator<< (std::ostream &os, const MicroTimer &mt)

  prints the timer

5.5.1 Detailed Description

Collects fine grain timing stats using gettimeofday().

5.5.2 Member Function Documentation

5.5.2.1 void MicroTimer::ComputeDiff () `[private]`

stores (end time - start time)

5.5.2.2 unsigned long MicroTimer::GetDiff () const `[inline, virtual]`

computes time

Returns:

difference of time between stop() and start() calls

5.5.2.3 void MicroTimer::Start () `[inline, virtual]`

starts the timer

5.5.2.4 virtual void MicroTimer::Stop () `[virtual]`

stops the timer

5.5.3 Friends And Related Function Documentation

5.5.3.1 std::ostream& operator<< (std::ostream & *os*, const MicroTimer & *mt*) `[friend]`

prints the timer

5.5.4 Member Data Documentation

5.5.4.1 timeval MicroTimer::diff_ `[private]`

5.5.4.2 timeval MicroTimer::end_ `[private]`

5.5.4.3 timeval MicroTimer::start_ `[private]`

The documentation for this class was generated from the following files:

- MicroTimer.h
- MicroTimer.cpp

6 SBLLmalloc File Documentation

6.1 AVL.cpp File Reference

AVL-tree implementation. In order to change what you want to store in the node, please add more elements in AVLTreeNode and pass it through the function templates.

Defines

- #define NDEBUG

Functions

- static void Destroy (AVLTreeNode ∗node)
- static void ∗ Insert (AVLTreeData ∗data, AVLTreeNode ∗∗node, const void ∗key, void ∗value)
- static void ∗ Remove (AVLTreeData ∗data, AVLTreeNode ∗∗node, const void ∗key)
- static AVLTreeNode ∗ RemoveLeftMost (AVLTreeNode ∗∗node)
- static AVLTreeNode ∗ RemoveRightMost (AVLTreeNode ∗∗node)
- static void Traverse (const AVLTreeNode ∗node, void(∗func)(const void ∗key, const void ∗value, const void ∗data, void ∗isDirty))
- static void Balance (AVLTreeNode ∗∗node)
- static int GetBalance (const AVLTreeNode ∗node)
- static int GetHeight (const AVLTreeNode ∗node)
- static void RotateSingleRight (AVLTreeNode ∗∗node)
- static void RotateSingleLeft (AVLTreeNode ∗∗node)
- static void RotateDoubleRight (AVLTreeNode ∗∗node)
- static void RotateDoubleLeft (AVLTreeNode ∗∗node)
- AVLTree ∗ CreateAVL (AVLComparator comparator)

  Create an empty AVL tree.

- void DestroyAVL (AVLTree ∗tree)

  Destroy an AVL tree.

- void ∗ InsertAVL (AVLTree ∗tree, const void ∗key, void ∗value)

  Insert an item to the AVL tree. Note that if the key is already in the tree the value will not be inserted.

- void ∗ RemoveAVL (AVLTree ∗tree, const void ∗key)

  Remove an item from the AVL tree.

- void ∗ FindAVL (const AVLTree ∗tree, const void ∗key)

  Find an item in the AVL tree.

- void ∗ FindRangeAVL (const AVLTree ∗tree, const void ∗key)

  Find if a value is in range of the AVL tree.

- void TraverseAVL (const AVLTree ∗tree, void(∗func)(const void ∗key, const void ∗value, const void ∗data, void ∗isDirty))

Traverse each element of the tree.

- int GetAVLSize (const AVLTree ∗tree)

  Get the number of elements in the tree.

- int GetAVLHeight (const AVLTree ∗tree)

  Get the height of the AVL tree.

6.1.1 Detailed Description

AVL-tree implementation. In order to change what you want to store in the node, please add more elements in AVLTreeNode and pass it through the function templates.

Author:

Joe Wingbermuehle

Date:

2007-06-11

6.1.2 Define Documentation

6.1.2.1 #define NDEBUG

6.1.3 Function Documentation

6.1.3.1 void Balance (AVLTreeNode ∗∗ *node*) `[static]`

6.1.3.2 AVLTree∗ CreateAVL (AVLComparator *comparator*)

Create an empty AVL tree.

Parameters:

comparator The comparator to use.

Returns:

An empty AVL tree.

6.1.3.3 void Destroy (AVLTreeNode ∗ *node*) `[static]`

6.1.3.4 void DestroyAVL (AVLTree * *tree*)

Destroy an AVL tree.

Parameters:

tree The AVL tree to destroy.

6.1.3.5 void* FindAVL (const AVLTree * *tree*, const void * *key*)

Find an item in the AVL tree.

Parameters:

tree The AVL tree.

key The key.

Returns:

The item (NULL if not found).

6.1.3.6 void* FindRangeAVL (const AVLTree * *tree*, const void * *key*)

Find if a value is in range of the AVL tree.

Parameters:

tree The AVL tree.

key The key.

Returns:

The item (NULL if not found).

6.1.3.7 int GetAVLHeight (const AVLTree * *tree*)

Get the height of the AVL tree.

Parameters:

tree The AVL tree.

Returns:

The height of the tree.

6.1.3.8 int GetAVLSize (const AVLTree * *tree*)

Get the number of elements in the tree.

Parameters:

tree The AVL tree.

Returns:

The number of elements in the tree.

6.1.3.9 int GetBalance (const AVLTreeNode * *node*) `[static]`

6.1.3.10 int GetHeight (const AVLTreeNode * *node*) `[static]`

6.1.3.11 void * Insert (AVLTreeData * *data*, AVLTreeNode ** *node*, const void * *key*, void * *value*) `[static]`

6.1.3.12 void* InsertAVL (AVLTree * *tree*, const void * *key*, void * *value*)

Insert an item to the AVL tree. Note that if the key is already in the tree the value will not be inserted.

Parameters:

tree The AVL tree.

key The key.

value The value.

Returns:

The value currently in the tree, if any.

6.1.3.13 void * Remove (AVLTreeData * *data*, AVLTreeNode ** *node*, const void * *key*) `[static]`

6.1.3.14 void* RemoveAVL (AVLTree * *tree*, const void * *key*)

Remove an item from the AVL tree.

Parameters:

tree The AVL tree.

key The key of the item to remove.

Returns:

The removed item (NULL if not found).

6.1.3.15 AVLTreeNode ∗ RemoveLeftMost (AVLTreeNode ∗∗ *node*) `[static]`

6.1.3.16 AVLTreeNode ∗ RemoveRightMost (AVLTreeNode ∗∗ *node*) `[static]`

6.1.3.17 void RotateDoubleLeft (AVLTreeNode ∗∗ *node*) `[static]`

6.1.3.18 void RotateDoubleRight (AVLTreeNode ∗∗ *node*) `[static]`

6.1.3.19 void RotateSingleLeft (AVLTreeNode ∗∗ *node*) `[static]`

6.1.3.20 void RotateSingleRight (AVLTreeNode ∗∗ *node*) `[static]`

6.1.3.21 void Traverse (const AVLTreeNode ∗ *node*, void(∗)(const void ∗key, const void ∗value, const void ∗data, void ∗isDirty) *func*) `[static]`

6.1.3.22 void TraverseAVL (const AVLTree ∗ *tree*, void(∗)(const void ∗key, const void ∗value, const void ∗data, void ∗isDirty) *func*)

Traverse each element of the tree.

Parameters:

tree The AVL tree.

func The traversal function.

data A value to be passed to the traversal function.

6.2 AVL.h File Reference

AVL-tree implementation.

Classes

- struct AVLTreeNode

  Structure to represent an AVL tree node.

- struct AVLTreeData

  Data for an AVL tree.

Defines

- #define MAX_STACK_DEPTH 20

  Maximum depth of call stack stored.

Typedefs

- typedef void ∗ AVLTree
- typedef int(∗) AVLComparator (const void ∗key1, const void ∗key2)

  Comparator for AVL tree keys.

Functions

- AVLTree ∗ CreateAVL (AVLComparator comparator)

  Create an empty AVL tree.

- void DestroyAVL (AVLTree ∗tree)

  Destroy an AVL tree.

- void ∗ InsertAVL (AVLTree ∗tree, const void ∗key, void ∗value)

  Insert an item to the AVL tree. Note that if the key is already in the tree the value will not be inserted.

- void ∗ RemoveAVL (AVLTree ∗tree, const void ∗key)

  Remove an item from the AVL tree.

- void ∗ FindAVL (const AVLTree ∗tree, const void ∗key)

  Find an item in the AVL tree.

- void ∗ FindRangeAVL (const AVLTree ∗tree, const void ∗key)

  Find if a value is in range of the AVL tree.

- void TraverseAVL (const AVLTree ∗tree, void(∗func)(const void ∗key, const void ∗value, const void ∗data, void ∗isDirty))

  Traverse each element of the tree.

- int GetAVLSize (const AVLTree ∗tree)

  Get the number of elements in the tree.

- int GetAVLHeight (const AVLTree ∗tree)

  Get the height of the AVL tree.

- uintptr_t GetBacktrace ()

  Returns the creator of the region.

- void GetCallStack (void ∗∗stack, int depth)

  Stores call stack in stack. depth is the maximum size supported.

6.2.1 Detailed Description

AVL-tree implementation.

Author:

Joe Wingbermuehle

Date:

2007-06-11

6.2.2 Define Documentation

6.2.2.1 #define MAX_STACK_DEPTH 20

Maximum depth of call stack stored.

6.2.3 Typedef Documentation

6.2.3.1 typedef int(∗) AVLComparator(const void ∗key1, const void ∗key2)

Comparator for AVL tree keys.

Parameters:

key1 The first key.

key2 The second key.

Returns:

The result of the comparison.

- < 0 if key1 is less than key2.
- $= 0$ if key1 equals key2.
- > 0 if key1 is greater than key2.

6.2.3.2 typedef void* AVLTree

AVL tree data type.

6.2.4 Function Documentation

6.2.4.1 AVLTree* CreateAVL (AVLComparator *comparator*)

Create an empty AVL tree.

Parameters:

comparator The comparator to use.

Returns:

An empty AVL tree.

6.2.4.2 void DestroyAVL (AVLTree * *tree*)

Destroy an AVL tree.

Parameters:

tree The AVL tree to destroy.

6.2.4.3 void* FindAVL (const AVLTree * *tree*, const void * *key*)

Find an item in the AVL tree.

Parameters:

tree The AVL tree.
key The key.

Returns:

The item (NULL if not found).

6.2.4.4 void∗ FindRangeAVL (const AVLTree ∗ *tree*, const void ∗ *key*)

Find if a value is in range of the AVL tree.

Parameters:

tree The AVL tree.

key The key.

Returns:

The item (NULL if not found).

6.2.4.5 int GetAVLHeight (const AVLTree ∗ *tree*)

Get the height of the AVL tree.

Parameters:

tree The AVL tree.

Returns:

The height of the tree.

6.2.4.6 int GetAVLSize (const AVLTree ∗ *tree*)

Get the number of elements in the tree.

Parameters:

tree The AVL tree.

Returns:

The number of elements in the tree.

6.2.4.7 uintptr_t GetBacktrace ()

Returns the creator of the region.

Returns:

address outside this library that created this region.

6.2.4.8 void GetCallStack (void ∗∗ *stack*, int *depth*)

Stores call stack in stack. depth is the maximum size supported.

Parameters:

> ***stack*** an array to store the pointers the length of the array stack

6.2.4.9 void∗ InsertAVL (AVLTree ∗ *tree*, const void ∗ *key*, void ∗ *value*)

Insert an item to the AVL tree. Note that if the key is already in the tree the value will not be inserted.

Parameters:

> ***tree*** The AVL tree.
> ***key*** The key.
> ***value*** The value.

Returns:

> The value currently in the tree, if any.

6.2.4.10 void∗ RemoveAVL (AVLTree ∗ *tree*, const void ∗ *key*)

Remove an item from the AVL tree.

Parameters:

> ***tree*** The AVL tree.
> ***key*** The key of the item to remove.

Returns:

> The removed item (NULL if not found).

6.2.4.11 void TraverseAVL (const AVLTree ∗ *tree*, void(∗)(const void ∗key, const void ∗value, const void ∗data, void ∗isDirty) *func*)

Traverse each element of the tree.

Parameters:

> ***tree*** The AVL tree.
> ***func*** The traversal function.
> ***data*** A value to be passed to the traversal function.

6.3 Globals.h File Reference

Contains global definitions and public interface.

Defines

- #define ptmalloc internal_malloc
- #define ptfree internal_free
- #define ptmalloc_get_mem_usage internal_footprint

Functions

- void ∗ ShmMallocWrapper (size_t sz)
  public interface for allocating shared pages
- void ∗ ShmReallocWrapper (void ∗ptr, size_t sz)
  public interface for reallocating shared pages
- int ShmFreeWrapper (void ∗ptr)
  public interface for freeing shared pages
- size_t ShmGetSizeWrapper (void ∗ptr)
  Gets size of an allocated region.

6.3.1 Detailed Description

Contains global definitions and public interface.

Author:

Susmit Biswas

Version:

1.0

Date:

2009-2010

6.3.2 Define Documentation

6.3.2.1 #define ptfree internal_free

6.3.2.2 #define ptmalloc internal_malloc

6.3.2.3 #define ptmalloc_get_mem_usage internal_footprint

6.3.3 Function Documentation

6.3.3.1 int ShmFreeWrapper (void ∗ *ptr*)

public interface for freeing shared pages

Parameters:

ptr Address of the allocation

Returns:

-1 if not allocated using SH_MMAP, 1 otherwise

6.3.3.2 size_t ShmGetSizeWrapper (void ∗ *ptr*)

Gets size of an allocated region.

Parameters:

ptr Address of the allocation

Returns:

size of the region

6.3.3.3 void∗ ShmMallocWrapper (size_t *sz*)

public interface for allocating shared pages

Parameters:

sz Size of allocation

Returns:

Address of the region if successful, NULL otherwise

6.3.3.4 void∗ ShmReallocWrapper (void ∗ *ptr*, size_t *sz*)

public interface for reallocating shared pages

Parameters:

ptr Address of the old allocation

sz Size of allocation

Returns:

Address of the region if successful, NULL otherwise

6.4 MicroTimer.cpp File Reference

Contains implementation of fine grain timer. You may not need to change this code. It uses gettimeofday() to obtain the duration of some operation. At the beginning of the code block, please declare a timer, start it and at the end of the block, stop it and get the duration by calling GetDiff.

Functions

- std::ostream & operator<< (std::ostream &os, const MicroTimer &mt)

6.4.1 Detailed Description

Contains implementation of fine grain timer. You may not need to change this code. It uses gettimeofday() to obtain the duration of some operation. At the beginning of the code block, please declare a timer, start it and at the end of the block, stop it and get the duration by calling GetDiff.

Author:

Susmit Biswas

Version:

1.0

Date:

2009-2010

e.g.

```
MicroTimer mt;
mt.Start();
{
// Code Block
...
}
mt.Stop();

double duration_in_usec = mt.GetDiff();
```

6.4.2 Function Documentation

6.4.2.1 std::ostream& operator<< (std::ostream & *os*, const MicroTimer & *mt*)

6.5 MicroTimer.h File Reference

Contains definitions of fine grain timer.

Classes

- class MicroTimer

    Collects fine grain timing stats using gettimeofday().

6.5.1 Detailed Description

Contains definitions of fine grain timer.

Author:

Susmit Biswas

Version:

1.0

Date:

2009-2010

6.6 README.tex File Reference

6.7 SharedHeap.cpp File Reference

Implementation of memory allocator using shared memory.

Defines

- #define PROF_MERGE_VERSION 2

  Profile guided merge: **disabled**.

- #define MAX_MERGES 10000

  Size of buffer used for storing memory usage stats.

- #define LMAX 4096
- #define compare_pages(a, b) memcmp((const void∗) a, (const void∗) b, PAGE_-SIZE)
- #define SIZE 100
- #define PRINT_PROFILE_DATA(start, end, caddr)
- #define FLUSH_OUTSTANDING_MERGES(start, end, lps, lpm, lpz, caddr)

  Map/Move outstanding pages.

- #define MMAP_BUFFER_SIZE (4∗1024∗1024)

Functions

- void UpdateMergeStat (const long int tpm, const long int tptm, const long int tzm, const long int tsm, const long int tum, const long int tmm, const int mtm)

  Stores memory usage in a buffer and flushes to file when the buffer is full.

- void PrintMergeStat ()

  Flushes merge stat buffer to file.

- void StoreMemUsageStat ()

  Collects current memory usage Not locking for aliveProcs as it is updated in the beginning of startup.

- int MPI_Init (int ∗argc, char ∗∗∗argv)

  Replaces `MPI_Init` *of mpi library and initializes shared memory, metadata etc. param argc argc from main(int argc, char ∗∗argv) param argv argv from main(int argc, char ∗∗argv).*

- int MPI_Finalize ()

  Replaces the `MPI_Finalize` *to call* `PMPI_Finalize()`.

- void InitAddrSpace ()

  Initializes shared region and sets segfault handler.

- void AllocateSharedMetadata ()

allocates a shared region to account for merged pages.

- void Fatal ()

  Aborts execution. Called upon encountering error.

- void GetMemRange ()

  Reads proc maps and finds the address range where this malloc library is allocated.

- void CheckEnv ()

  Checks environment variables for sanity.

- void InitEnv ()

  Initializes the parameters for the library.

- int FloorLog2 (unsigned long n)

  Computes `floor`*(log2(n)) Works by finding position of MSB set.*

- int CeilLog2 (unsigned long n)

  Computes `ceil`*(log2(n)).*

- uintptr_t TranslateMmapAddr (uintptr_t addr)

  Translate mmapped address to file offset.

- bool CheckMPIInitialized ()

  Checks if `MPI_Init` *has been called or is not a MPI app.*

- bool IsCloseToMmapLimit (int newRequest)

  Checks if number of mmap calls are close to system limit.

- void InitSem (char ∗SEMKEY, sem_t ∗∗mutex)

  Initializes a `POSIX` *semaphore after getting it In linux, semkey has to be an existing filename beginning with / but not having more than 14 chars and more slashes.*

- void SignalSem (sem_t ∗mutex)

  operation **V:** *used for semaphore handling*

- void WaitSem (sem_t ∗mutex)

  operation **P:** *used for semaphore handling*

- void SigSegvHandler (int32_t signo, siginfo_t ∗si, void ∗sc)
- void SigIntHandler (int32_t signo, siginfo_t ∗si, void ∗sc)
- void SigBusHandler (int32_t signo, siginfo_t ∗si, void ∗sc)
- uintptr_t Addr2PageIndex (void ∗address)

Translates page address to page number.

- int CountSharingProcs (void ∗addr)

  Counts the number of tasks sharing a page.

- void SetSharingBit (void ∗addr)

  Sets the sharing bit corresponding to page having address addr for currest process.

- void UnsetSharingBit (void ∗addr)

  Unsets the sharing bit corresponding to page having address addr for currest process.

- bool GetSharingBit (void ∗addr)

  Gets the sharing bit corresponding to page having address addr for currest process.

- bool IsOtherSharing (void ∗addr)

  Checks any other process shares the page having address addr by checking sharing-ProcessesInfo bits.

- bool GetBit (char ∗array, char ∗page_address)

  Gets bit corresponding to an address from bitvector array.

- void SetBit (char ∗array, char ∗page_address)
- void UnsetBit (char ∗array, void ∗page_address)
- void SetMultiBits (char ∗array, char ∗page_address, size_t size)
- bool SetAndReturnBit (char ∗array, char ∗page_address)

  Sets and returns old value of bit in array corresponding to page address.

- bool ResetAndReturnBit (char ∗array, char ∗page_address)

  Resets and returns old value of bit in array corresponding to page address.

- int AspaceAvlInsertWrapper (uintptr_t start_addr, size_t size)

  Inserts a node in address space AVL tree with start_addr and size. called upon malloc.

- intptr_t AspaceAvlRemoveWrapper (uintptr_t start_addr)

  Deletes a node from AVL tree with start_addr. Called upon free.

- uintptr_t AspaceAvlSearchWrapper (uintptr_t start_addr)

  Searches a node with start_addr.

- uintptr_t AspaceAvlSearchRangeWrapper (uintptr_t start_addr)

  Searches a range with start_addr included in an existing range.

- int MyComparator (const void ∗key1, const void ∗key2)

AVL data structure compartor.

- void MergeByBUFFERED ()

  merges pages when the buffer of dirty pages becomes full Experimental. NOT EXTENSIVELY TESTED

- void MergeByALLOC_FREQUENCY ()

  Merges pages based on frequency of mallocs. When the number of outstanding malloc/free becomes more than x where x is the merging frequency, merge operation is triggered.

- void MergeByTHRESHOLD ()

  Merges pages based on threshold When the amount of used memory changes by more than 1000 pages i.e. 4MB, merge operation is triggered.

- void MakeReadOnlyWrapper (void ∗addr, size_t len)

  makes a region readonly

- void MakeReadWriteWrapper (void ∗addr, size_t len)

  makes a region writeable

- void print_node (const void ∗key, const void ∗value, const void ∗data)

  *Prints a node of AVL tree (***unused***).*

- void MergeNode (const void ∗key, const void ∗value, const void ∗data)

  a node corresponding to <key, value> pair is checked so that identical data pages can be merged

- void MergeNode2 (const void ∗key, const void ∗value, const void ∗data, void ∗isDirty)

  A node corresponding to <key, value> pair is checked so that identical data pages can be merged. Merges many pages at once.

- void FreeNode (const void ∗key, const void ∗value, const void ∗data, void ∗isDirty)

  Frees up a node corresponding to <key, value> pair.

- void ∗ GetSharedRegion (void ∗addr, bool isFixed, size_t size)

  Allocates a shared region.

- void ∗ GetSharedPage (void ∗addr, bool isFixed)

  Allocates a shared page Calls GetSharedRegion with addr, isFixed, PAGE_SIZE.

- void CleanUpSharedData ()

Cleans up shared data: deletes AVL, closes files, unmaps shared region, destroys semaphores.

- void GetCallStack (void ∗∗stack, int depth)

  Stores the call stack when malloc is called. It excludes the library addresses from the call trace.

- uintptr_t GetBacktrace ()

  Finds address of the source that called malloc.

- int CopyAndRemapRegion (void ∗start, size_t size)

  Copies and maps pages from private region to shared space.

- int RemapRegion (void ∗start, size_t size)

  Maps pages from private region to shared space Some other process shared this memory region, so just remapping required here.

- int RemapToZero (void ∗start, size_t size)

  Remaps the pages to the zero page.

- int MergeManyPages (uintptr_t start_addr, size_t size, const void ∗data)

  Merges many pages This is the main routine that identifies the type of merge and invokes appropriate functions. It is a state machine based implementation to reduce fragmentation on the number of mmaps.

- int MergePages (void ∗p, uintptr_t creator_addr)

  Tries to merge pages of current process.

- void ∗ ShmMallocWrapper (size_t sz)

  public interface for allocating shared pages

- void ∗ ShmReallocWrapper (void ∗ptr, size_t size)

  public interface for reallocating shared pages

- size_t ShmGetSizeWrapper (void ∗ptr)

  Gets size of an allocated region.

- int ShmFreeWrapper (void ∗ptr)

  public interface for freeing shared pages

Variables

- static bool isMPIInitialized = false
- static bool isMPIFinalized = false
- static int maxMmapCount = 65536
- static int mmapCount = 0
- static int myRank = -1
- static int numProc = 0
- static int sharedFileDescr = -1
- static int ∗ sharingProcessesInfo = NULL
- static unsigned long currProcMask = 0x01
- static unsigned long currProcMaskInverted = (unsigned long)(-1)
- static int notMPIApp = 0
- static int mergeMetric = THRESHOLD
- static int mergeMinMemTh = 10000
- static int mallocRefFreq = MALLOC_REF_FREQ
- static unsigned long mallocRefCounter = 0
- static uintptr_t bufferOfDirtyPages [BUFFER_LENGTH]
- static int bufferPtr = 0
- static uintptr_t lowLoadAddr = (unsigned long)(-1)
- static uintptr_t highLoadAddr = 0
- static int enableBacktrace = 0
- static key_t semKey = 1234
- static char semName [200]
- static sem_t ∗ mutex = NULL
- static AVLTreeData ∗ allocRecord = NULL
- static int ∗ aliveProcs = NULL
- static int PAGE_SIZE = 4096
- static unsigned log2PAGE_SIZE = 12
- static int ∗ sharedPageCount = NULL
- static int ∗ allProcPrivatePageCount = NULL
- static int ∗ baseCaseTotalPageCount = NULL
- static char initializedPagesBV [98304]
- static char zeroPagesBV [98304]
- static char ∗ zeroPage = NULL
- static int zeroPageCount = 0
- static FILE ∗ outFile = NULL
- static int maxBaseCaseTotalPageCount = 0
- static MemStatStruct memStat [MAX_MERGES]

  Buffer for storing memory usage stats.

- static int memStatCounter = 0

  Currnt index in the buffer for storing memory usage stats.

6.7.1 Detailed Description

Implementation of memory allocator using shared memory.

Author:

Susmit Biswas

Version:

1.0

Date:

2009-2010

The public interfaces are defined in `Globals.h`. In this file two functions - `MPI_Init()` and `MPI_Finalize()` override the functions from MPI library. Also, the malloc calls from the internal malloc (ptmalloc v3 used in this package) replace the system `malloc()`,`free()`, `realloc()` calls. If you want to use a different internal malloc, you need to make the public routines (malloc, calloc, free etc.) as wrappers and move the code to some internal routine. After that define those internal routines in a file that you need to include in Globals.h. For example, check out `internal-routines.h` which defines the `internal_malloc()`, `internal_free()` etc. internal_malloc.h is included in `Globals.h`.

Memory usage in a demand paging system increases when the pages get written for the first time or a shared page becomes private. In the threshold based merge technique, at the end of SigSegvHandler, `MergeByTHRESHOLD()` routine is called with iteratively calls MergeNode2() by traversing the AVL tree. If a AVL tree node is dirty, `MergeManyPages()` routine is called which merges identical pages from that node. In `MergeManyPages()` many pages are handled at once i.e. their permission bits are changed, they are mapped/unmapped in-order to reduce overhead. `FLUSH_OUTSTANDING_MERGES()` is used in this state machine which keeps contiguous page addresses using a *start* and *size* field.

Precondition:

- There are at most x MPI tasks per node where x is the number of cores.
- There is no need to rely on MPI task rank. rather use shared memory value as rank.
- OS ensures that `MAP_FIXED` or `MREMAP_FIXED` replaces previous mappings (tested to be true in Ubuntu linux).

Todo

- Modify backtrace capability:
  1. Allocate an array for each of the AVL node instead of just creator (Done)

2. Store 20 entries from the callstack (Done)
3. At every merge point: find what pages are merged and what is the source i.e. the creator (Done)
4. Dump out the addresses. (Done)
5. After exit, run `addr2line` to find out the stack traces (Done)
6. Report top contributors as a graph.

- Recover from mmap failure
  1. We really cannot recover, we can just keep our finger crossed that no other library calls mmap.
- Support non-MPI apps by changing `MPI_Init` to a routine that runs at program startup
  1. Use `__attribute__` ((constructor)) `TheStartRoutineForNonMPIapp();`

6.7.2 Define Documentation

6.7.2.1 #define compare_pages(a, b) memcmp((const void*) a, (const void*) b, PAGE_SIZE)

6.7.2.2 #define FLUSH_OUTSTANDING_MERGES(start, end, lps, lpm, lpz, caddr)

Value:

```
{\
    if(start){\
        if(lps) RemapRegion         (start, (size_t)(ptr2offset(end) - ptr2offset(start)));\
        if(lpm) CopyAndRemapRegion  (start, (size_t)(ptr2offset(end) - ptr2offset(start)));\
        if(lpz) RemapToZero         (start, (size_t)(ptr2offset(end) - ptr2offset(start)));\
        if(lps || lpz){ \
            PRINT_PROFILE_DATA(start, end, caddr);\
            counter_pages_merged += (ptr2offset(end) - ptr2offset(start))/PAGE_SIZE;\
        }\
    }\
    lps = lpm = lpz = false;\
    start = NULL;\
}
```

Map/Move outstanding pages.

Parameters:

start Start addr

end End addr

lps Boolean flag whether last page shareable

lpm Boolean flag whether last page mergeable

lpz Boolean flag whether last page zero

caddr Creator addr for these set of pages

6.7.2.3 #define LMAX 4096

6.7.2.4 #define MAX_MERGES 10000

Size of buffer used for storing memory usage stats.

6.7.2.5 #define MMAP_BUFFER_SIZE (4∗1024∗1024)

Maximum size of the buffer used for mapping and managing pages in bulk

6.7.2.6 #define PRINT_PROFILE_DATA(start, end, caddr)

6.7.2.7 #define PROF_MERGE_VERSION 2

Profile guided merge: **disabled**.

6.7.2.8 #define SIZE 100

6.7.3 Function Documentation

6.7.3.1 uintptr_t Addr2PageIndex (void ∗) `[inline]`

Translates page address to page number.

Returns:

Page number

6.7.3.2 void AllocateSharedMetadata ()

allocates a shared region to account for merged pages.

Returns:

None

6.7.3.3 int AspaceAvlInsertWrapper (uintptr_t *start_addr*, size_t *size*)

Inserts a node in address space AVL tree with start_addr and size. called upon malloc.

Returns:

0

6.7.3.4 intptr_t AspaceAvlRemoveWrapper (uintptr_t *start_addr*)

Deletes a node from AVL tree with start_addr. Called upon free.

Returns:

The size of the region, 0 if not found.

6.7.3.5 uintptr_t AspaceAvlSearchRangeWrapper (uintptr_t *start_addr*)

Searches a range with start_addr included in an existing range.

Returns:

0 if not found, the size of the region if found

6.7.3.6 uintptr_t AspaceAvlSearchWrapper (uintptr_t *start_addr*)

Searches a node with start_addr.

Returns:

0 if not found, the size of the region if found

6.7.3.7 int CeilLog2 (unsigned *long*) `[inline]`

Computes `ceil`(log2(n)).

See also:

FloorLog2

Returns:

-1 if n == 0.

6.7.3.8 void CheckEnv ()

Checks environment variables for sanity.

6.7.3.9 bool CheckMPIInitialized () `[inline]`

Checks if `MPI_Init` has been called or is not a MPI app.

Returns:

true when MPI has been Initialized/ not MPI app .

6.7.3.10 void CleanUpSharedData ()

Cleans up shared data: deletes AVL, closes files, unmaps shared region, destroys semaphores.

6.7.3.11 int CopyAndRemapRegion (void ∗ *start*, size_t *size*)

Copies and maps pages from private region to shared space.

Parameters:

start Address of the start of the region

size Size of the region

Returns:

-1 if failure, 0 if successful

6.7.3.12 int CountSharingProcs (void ∗ *addr*)

Counts the number of tasks sharing a page.

Parameters:

addr Address of the page

Returns:

the Number of sharing processes

6.7.3.13 void Fatal ()

Aborts execution. Called upon encountering error.

6.7.3.14 int FloorLog2 (unsigned *long*) `[inline]`

Computes `floor`(log2(n)) Works by finding position of MSB set.

Returns:

-1 if n == 0.

6.7.3.15 void FreeNode (const void ∗ *key*, const void ∗ *value*, const void ∗ *data*, void ∗ *isDirty*) `[inline]`

Frees up a node corresponding to <key, value> pair.

Parameters:

key Address of start address of the region to check for merging

value Size of the region

data Address of the creator of the region (not used)

isDirty Flag indicating whether the region is dirty (not used)

6.7.3.16 uintptr_t GetBacktrace ()

Finds address of the source that called malloc.

Returns:

address outside this library that created this region.

6.7.3.17 bool GetBit (char ∗ *array*, char ∗ *page_addr*) `[inline]`

Gets bit corresponding to an address from bitvector array.

Parameters:

array Bit vector

page_addr Address of the page

Returns:

The bit for address page_addr from array bit vector

6.7.3.18 void GetCallStack (void ∗∗ *stack*, int *depth*)

Stores the call stack when malloc is called. It excludes the library addresses from the call trace.

Parameters:

stack an array to store the pointers the length of the array stack

6.7.3.19 void GetMemRange ()

Reads proc maps and finds the address range where this malloc library is allocated.

6.7.3.20 void∗ GetSharedPage (void ∗ *addr*, bool *isFixed*) `[inline]`

Allocates a shared page Calls GetSharedRegion with addr, isFixed, PAGE_SIZE.

See also:

GetSharedRegion

Returns:

Address of the page

6.7.3.21 void∗ GetSharedRegion (void ∗ *addr*, bool *isFixed*, size_t *size*)

Allocates a shared region.

Parameters:

isFixed If set, map at a fixed address

addr

size Size of the region

Returns:

Address of the mapped region or MAP_FAILED if error encountered

6.7.3.22 bool GetSharingBit (void ∗ *addr*) `[inline]`

Gets the sharing bit corresponding to page having address addr for currest process.

6.7.3.23 void InitAddrSpace ()

Initializes shared region and sets segfault handler.

Returns:

None

6.7.3.24 void InitEnv ()

Initializes the parameters for the library.

6.7.3.25 void InitSem (char ∗ *SEMKEY*, sem_t ∗∗ *mutex*)

Initializes a `POSIX` semaphore after getting it In linux, semkey has to be an existing filename beginning with / but not having more than 14 chars and more slashes.

Parameters:

SEMKEY Name of the semaphore

mutex The semaphore address

6.7.3.26 bool IsCloseToMmapLimit (int *newRequest* = `0`) `[inline]`

Checks if number of mmap calls are close to system limit.

Returns:

true if close to limit

Deprecated

By setting `vm.max_map_count` to a large value (e.g. 512K), we do not need this check

6.7.3.27 bool IsOtherSharing (void ∗ *addr*) `[inline]`

Checks any other process shares the page having address addr by checking sharingProcessesInfo bits.

6.7.3.28 void MakeReadOnlyWrapper (void ∗ *addr*, size_t *len*) `[inline]`

makes a region readonly

Parameters:

addr Start address of the region
len Size of the region

6.7.3.29 void MakeReadWriteWrapper (void ∗ *addr*, size_t *len*) `[inline]`

makes a region writeable

Parameters:

addr Start address of the region
len Size of the region

6.7.3.30 void MergeByALLOC_FREQUENCY ()

Merges pages based on frequency of mallocs. When the number of outstanding malloc/free becomes more than x where x is the merging frequency, merge operation is triggered.

6.7.3.31 void MergeByBUFFERED ()

merges pages when the buffer of dirty pages becomes full Experimental. NOT EXTENSIVELY TESTED

6.7.3.32 void MergeByTHRESHOLD ()

Merges pages based on threshold When the amount of used memory changes by more than 1000 pages i.e. 4MB, merge operation is triggered.

6.7.3.33 int MergeManyPages (uintptr_t *start_addr*, size_t *size*, const void ∗ *data*)

Merges many pages This is the main routine that identifies the type of merge and invokes appropriate functions. It is a state machine based implementation to reduce fragmentation on the number of mmaps.

Parameters:

start Address of the start of the region
size Size of the region
data Call stack of the region

Returns:

0 if successful, -1 otherwise

6.7.3.34 void MergeNode (const void ∗ *key*, const void ∗ *value*, const void ∗ *data*)

a node corresponding to <key, value> pair is checked so that identical data pages can be merged

Parameters:

key Address of start address of the region to check for merging

value Size of the region

data Address of the creator of the region

Deprecated

`MergeNode2` is more efficiently implemented

See also:

MergeNode2

6.7.3.35 void MergeNode2 (const void ∗ *key*, const void ∗ *value*, const void ∗ *data*, void ∗ *isDirty*)

A node corresponding to <key, value> pair is checked so that identical data pages can be merged. Merges many pages at once.

Parameters:

key Address of start address of the region to check for merging

value Size of the region

data Call stack when the region was created

isDirty Flag indicating if the region is dirty

6.7.3.36 int MergePages (void ∗ *p*, uintptr_t *creator_addr*)

Tries to merge pages of current process.

Parameters:

p Address of the page to be compared

creator_addr Address of the creator of page p

Deprecated

This routine is not tested thouroughly, and `MergeByTHRESHOLD` is more apt.

Returns:

0 if not merged. 1 if merged

6.7.3.37 int MPI_Finalize ()

Replaces the `MPI_Finalize` to call `PMPI_Finalize()`.

6.7.3.38 int MPI_Init (int ∗ *argc*, char ∗∗∗ *argv*)

Replaces `MPI_Init` of mpi library and initializes shared memory, metadata etc. param argc argc from main(int argc, char ∗∗argv) param argv argv from main(int argc, char ∗∗argv).

Returns:

Result of PMPI_Init

6.7.3.39 int MyComparator (const void ∗ *key1*, const void ∗ *key2*) `[inline]`

AVL data structure compartor.

Returns:

0 if identical, otherwise the difference as 1 or -1

6.7.3.40 void print_node (const void ∗ *key*, const void ∗ *value*, const void ∗ *data*) `[inline]`

Prints a node of AVL tree (**unused**).

6.7.3.41 void PrintMergeStat () `[inline]`

Flushes merge stat buffer to file.

6.7.3.42 int RemapRegion (void ∗ *start*, size_t *size*)

Maps pages from private region to shared space Some other process shared this memory region, so just remapping required here.

Parameters:

start Address of the start of the region

size Size of the region

Returns:

-1 if failure, 0 if successful

6.7.3.43 int RemapToZero (void ∗ *start*, size_t *size*)

Remaps the pages to the zero page.

Parameters:

start Address of the start of the region

size Size of the region

Returns:

0 if successful, -1 otherwise

6.7.3.44 bool ResetAndReturnBit (char ∗ *array*, char ∗ *page_address*) `[inline]`

Resets and returns old value of bit in array corresponding to page address.

6.7.3.45 bool SetAndReturnBit (char ∗ *array*, char ∗ *page_address*) `[inline]`

Sets and returns old value of bit in array corresponding to page address.

6.7.3.46 void SetBit (char ∗ *array*, char ∗ *page_addr*) `[inline]`

@ Sets bit corresponding to an address from bitvector array

Parameters:

array Bit vector

page_addr Address of the page

Returns:

None

6.7.3.47 void SetMultiBits (char * *array*, char * *page_addr*, size_t *size*)

@ Sets multiple bits corresponding to a region, more than 1 page from bitvector array

Parameters:

array Bit vector
page_addr Start address of the region
size Size of the region

Returns:

None

6.7.3.48 void SetSharingBit (void * *addr*) `[inline]`

Sets the sharing bit corresponding to page having address addr for currest process.

6.7.3.49 int ShmFreeWrapper (void * *ptr*)

public interface for freeing shared pages

Parameters:

ptr Address of the allocation

Returns:

-1 if not allocated using SH_MMAP, 1 otherwise

6.7.3.50 size_t ShmGetSizeWrapper (void * *ptr*)

Gets size of an allocated region.

Parameters:

ptr Address of the allocation

Returns:

size of the region

6.7.3.51 void∗ ShmMallocWrapper (size_t *sz*)

public interface for allocating shared pages

Parameters:

sz Size of allocation

Returns:

Address of the region if successful, NULL otherwise

6.7.3.52 void∗ ShmReallocWrapper (void ∗ *ptr*, size_t *sz*)

public interface for reallocating shared pages

Parameters:

ptr Address of the old allocation
sz Size of allocation

Returns:

Address of the region if successful, NULL otherwise

6.7.3.53 void SigBusHandler (int32_t *signo*, siginfo_t ∗ *si*, void ∗ *sc*)

6.7.3.54 void SigIntHandler (int32_t *signo*, siginfo_t ∗ *si*, void ∗ *sc*)

6.7.3.55 void SignalSem (sem_t ∗ *mutex*) `[inline]`

operation **V:** used for semaphore handling

Parameters:

mutex Address of the semaphore ∗

6.7.3.56 void SigSegvHandler (int32_t *signo*, siginfo_t ∗ *si*, void ∗ *sc*)

6.7.3.57 void StoreMemUsageStat ()

Collects current memory usage Not locking for aliveProcs as it is updated in the beginning of startup.

Returns:

None

6.7.3.58 uintptr_t TranslateMmapAddr (uintptr_t) `[inline]`

Translate mmapped address to file offset.

Parameters:

addr Address of the page to be translated

Returns:

0 if out of address range, translated address otherwise

6.7.3.59 void UnsetBit (char ∗ *array*, void ∗ *page_addr*) `[inline]`

@ Unsets bit corresponding to an address from bitvector array

Parameters:

array Bit vector

page_addr Address of the page

Returns:

None

6.7.3.60 void UnsetSharingBit (void ∗ *addr*) `[inline]`

Unsets the sharing bit corresponding to page having address addr for currest process.

6.7.3.61 void UpdateMergeStat (const long int *tpm*, const long int *tptm*, const long int *tzm*, const long int *tsm*, const long int *tum*, const long int *tmm*, const int *mtm*) `[inline]`

Stores memory usage in a buffer and flushes to file when the buffer is full.

Parameters:

$\leftarrow$ ***tpm*** Total memory as private pages

$\leftarrow$ ***tptm*** Total memory used by internal allocator

$\leftarrow$ ***tzm*** Total zero memory in current process

$\leftarrow$ ***tsm*** Total shared memory usage

$\leftarrow$ ***tum*** Memory footprint if merging is disabled

$\leftarrow$ ***tmm*** Memory footprint with merging enabled

$\leftarrow$ ***mtm*** Time used for merging in microsecond

6.7.3.62 void WaitSem (sem_t ∗ *mutex*) `[inline]`

operation **P:** used for semaphore handling

Parameters:

mutex Address of the semaphore

6.7.4 Variable Documentation

6.7.4.1 int∗ aliveProcs = NULL `[static]`

Number of active processes

6.7.4.2 AVLTreeData∗ allocRecord = NULL `[static]`

Avl tree used to keep track of allocated regions

6.7.4.3 int∗ allProcPrivatePageCount = NULL `[static]`

Total number of private pages across all tasks

6.7.4.4 int∗ baseCaseTotalPageCount = NULL `[static]`

Total number of pages in base case

6.7.4.5 uintptr_t bufferOfDirtyPages[BUFFER_LENGTH] `[static]`

In buffer based approach, stores the dirty pages addr

6.7.4.6 int bufferPtr = 0 `[static]`

Indicates where we should put the dirty page address in the buffer

6.7.4.7 unsigned long currProcMask = 0x01 `[static]`

Used for faster bitwise ops, created from myRank

6.7.4.8 unsigned long currProcMaskInverted = (unsigned long)(-1) `[static]`

Used for faster bitwise ops, created from myRank

6.7.4.9 int enableBacktrace = 0 `[static]`

Find the sources of merging regions i.e. who created the region

6.7.4.10 uintptr_t highLoadAddr = 0 `[static]`

Min value of lib loaded address

6.7.4.11 char initializedPagesBV[98304] `[static]`

3GB, 1 bit per page i.e. 0.75/8 MB

6.7.4.12 bool isMPIFinalized = false `[static]`

Flag indicating whether mpi is finalized

6.7.4.13 bool isMPIInitialized = false `[static]`

Flag indicating whether mpi is initialized

6.7.4.14 unsigned log2PAGE_SIZE = 12 `[static]`

4 KB default page

6.7.4.15 uintptr_t lowLoadAddr = (unsigned long)(-1) `[static]`

Max value of lib loaded address

6.7.4.16 unsigned long mallocRefCounter = 0 `[static]`

Counts the # of mallocs to trigger merging

6.7.4.17 int mallocRefFreq = MALLOC_REF_FREQ `[static]`

Frequency of merging

6.7.4.18 int maxBaseCaseTotalPageCount = 0 `[static]`

Total amount of memory used with default allocator, counted even if merge is disabled

6.7.4.19 int maxMmapCount = 65536 `[static]`

OS limit on max mmaps

6.7.4.20 MemStatStruct memStat[MAX_MERGES] `[static]`

Buffer for storing memory usage stats.

6.7.4.21 int memStatCounter = 0 `[static]`

Currnt index in the buffer for storing memory usage stats.

6.7.4.22 int mergeMetric = THRESHOLD `[static]`

How to perform merge? allocation frequency/threshold/buffer

6.7.4.23 int mergeMinMemTh = 10000 `[static]`

Threshold value for threshold based merge

6.7.4.24 int mmapCount = 0 `[static]`

Used for keeping track of mmap counts and checking limits

6.7.4.25 sem_t* mutex = NULL `[static]`

Semaphore used for coherence

6.7.4.26 int myRank = -1 `[static]`

Rank of current task

6.7.4.27 int notMPIApp = 0 `[static]`

A stand alone program needs to define corresponding env var

6.7.4.28 int numProc = 0 `[static]`

Number of processes in local node

6.7.4.29 FILE* outFile = NULL `[static]`

Output file for storing results

6.7.4.30 int PAGE_SIZE = 4096 `[static]`

4 KB default page

6.7.4.31 key_t semKey = 1234 `[static]`

Key used to create a new semaphore name

6.7.4.32 char semName[200] `[static]`

Used for posix semaphore

6.7.4.33 int sharedFileDescr = -1 `[static]`

mmapped file used for sharing

6.7.4.34 int∗ sharedPageCount = NULL `[static]`

Number of pages shared

6.7.4.35 int∗ sharingProcessesInfo = NULL `[static]`

Bitvectors for indicating sharing status of pages

6.7.4.36 char∗ zeroPage = NULL `[static]`

Addr of zero page

6.7.4.37 int zeroPageCount = 0 `[static]`

Number of zero pages for current task

6.7.4.38 char zeroPagesBV[98304] `[static]`

Is it a zero page, 3GB, 1 bit per page i.e. 0.75/8 MB

6.8 SharedHeap.h File Reference

Header file for SBLLmalloc.

Classes

- struct MemStatStruct

  The structure for storing merge info.

- struct commandLineArgument

  Structure for parsing arguments.

Defines

- #define MALLOC_REF_FREQ 1000

    Used for frequency based merge as default frequency.

- #define BUFFER_LENGTH 10000

    Used for buffer based merge as default frequency.

- #define ptr2offset(x) ((uintptr_t)x)

    Converts pointer to uintptr_t.

- #define offset2ptr(x) ((void ∗)(x))

    Converts uintptr_t to pointer.

- #define SHARED_STATS

    Enables shared variables for statistics collection.

- #define COLLECT_MALLOC_STAT

    Enabled statistics collection for mallocs.

- #define SHARED_STATS

    Enables shared variables for statistics collection.

- #define PRINT_STATS

    Enables profiling Collect sub-block merging stats Flags for controlling msg level.

- #define warn(msg)

    Issues warning message.

- #define die(m) {perror(m); fflush(stderr); MPI_Abort(MPI_COMM_WORLD, MPI_ERR_OTHER);}

    Issues warning message m *and dies with* `MPI_Abort`.

- #define NDEBUG
- #define ReportError(addr)

    Reports error by printing error address and backtrace.

- #define CheckForError()

    Reports error if errno is set by printing source info and backtrace.

- #define ASSERTX(a) assert(a)

    If the evaluated expression is false, prints backtrace and exits.

Enumerations

- enum _MERGE_METRICS {
  MERGE_DISABLED, ALLOC_FREQUENCY, THRESHOLD, BUFFERED,
  NUM_METRIC }

  Different merge metric. Set 0 to disable.

- enum _PROFILE_MODES { NONE, CREATE_PROF, USE_PROF, NUM_-MODES }

  Different profile modes. Set 0 to disable.

Functions

- void ∗ SH_MMAP (...)
- int SH_UNMAP (...)(mmapCount-
- return munmap (__VA_ARGS__)
- void StoreMemUsageStat ()

  Collects current memory usage Not locking for aliveProcs as it is updated in the beginning of startup.

- void UpdateMergeStat (const long int tpm, const long int tptm, const long int tzm, const long int tsm, const long int tum, const long int tmm, const int mtm)

  Stores memory usage in a buffer and flushes to file when the buffer is full.

- void PrintMergeStat ()

  Flushes merge stat buffer to file.

- int MPI_Init (int ∗argc, char ∗∗∗argv)

  Replaces `MPI_Init` *of mpi library and initializes shared memory, metadata etc. param argc argc from main(int argc, char ∗∗argv) param argv argv from main(int argc, char ∗∗argv).*

- int MPI_Finalize ()

  Replaces the `MPI_Finalize` *to call* `PMPI_Finalize()`.

- void InitAddrSpace ()

  Initializes shared region and sets segfault handler.

- void CleanUpSharedData ()

  Cleans up shared data: deletes AVL, closes files, unmaps shared region, destroys semaphores.

- void AllocateSharedMetadata ()

  allocates a shared region to account for merged pages.

- void Fatal ()

  Aborts execution. Called upon encountering error.

- void GetMemRange ()

  Reads proc maps and finds the address range where this malloc library is allocated.

- void CheckEnv ()

  Checks environment variables for sanity.

- void InitEnv ()

  Initializes the parameters for the library.

- int FloorLog2 (unsigned long)

  Computes `floor`*(log2(n)) Works by finding position of MSB set.*

- int CeilLog2 (unsigned long)

  Computes `ceil`*(log2(n)).*

- uintptr_t TranslateMmapAddr (uintptr_t)

  Translate mmapped address to file offset.

- bool CheckMPIInitialized ()

  Checks if `MPI_Init` *has been called or is not a MPI app.*

- bool IsCloseToMmapLimit (int newRequest=0)

  Checks if number of mmap calls are close to system limit.

- void InitSem (char ∗SEMKEY, sem_t ∗∗mutex)

  Initializes a `POSIX` *semaphore after getting it In linux, semkey has to be an existing filename beginning with / but not having more than 14 chars and more slashes.*

- void SignalSem (sem_t ∗mutex)

  operation **V:** *used for semaphore handling*

- void WaitSem (sem_t ∗mutex)

  operation **P:** *used for semaphore handling*

- void SigSegvHandler (int signo, siginfo_t ∗si, void ∗sc)

SIGSEGV signal handler. Handles write faults for readonly marked shared pages. If the page was never touched, the permission bit is changed and returned. Otherwise, a copy of the page is created in private region.

- void SigBusHandler (int signo, siginfo_t ∗si, void ∗sc)

  SIGBUS signal handler.

- void SigIntHandler (int signo, siginfo_t ∗si, void ∗sc)

  SIGINT signal handler.

- uintptr_t Addr2PageIndex (void ∗)

  Translates page address to page number.

- int CountSharingProcs (void ∗addr)

  Counts the number of tasks sharing a page.

- void SetSharingBit (void ∗addr)

  Sets the sharing bit corresponding to page having address addr for currest process.

- void UnsetSharingBit (void ∗addr)

  Unsets the sharing bit corresponding to page having address addr for currest process.

- bool GetSharingBit (void ∗addr)

  Gets the sharing bit corresponding to page having address addr for currest process.

- bool IsOtherSharing (void ∗addr)

  Checks any other process shares the page having address addr by checking sharing-ProcessesInfo bits.

- bool GetBit (char ∗array, char ∗page_addr)

  Gets bit corresponding to an address from bitvector array.

- void SetBit (char ∗array, char ∗page_addr)
- void UnsetBit (char ∗array, void ∗page_addr)
- void SetMultiBits (char ∗array, char ∗page_addr, size_t size)
- bool ResetAndReturnBit (char ∗, char ∗)

  Resets and returns old value of bit in array corresponding to page address.

- bool SetAndReturnBit (char ∗, char ∗)

  Sets and returns old value of bit in array corresponding to page address.

- int AspaceAvlInsertWrapper (uintptr_t start_addr, size_t size)

  Inserts a node in address space AVL tree with start_addr and size. called upon malloc.

- intptr_t AspaceAvlRemoveWrapper (uintptr_t start_addr)

  Deletes a node from AVL tree with start_addr. Called upon free.

- uintptr_t AspaceAvlSearchWrapper (uintptr_t start_addr)

  Searches a node with start_addr.

- uintptr_t AspaceAvlSearchRangeWrapper (uintptr_t start_addr)

  Searches a range with start_addr included in an existing range.

- int MyComparator (const void ∗key1, const void ∗key2)

  AVL data structure compartor.

- void MergeByALLOC_FREQUENCY ()

  Merges pages based on frequency of mallocs. When the number of outstanding malloc/free becomes more than x where x is the merging frequency, merge operation is triggered.

- void MergeByTHRESHOLD ()

  Merges pages based on threshold When the amount of used memory changes by more than 1000 pages i.e. 4MB, merge operation is triggered.

- void MergeByBUFFERED ()

  merges pages when the buffer of dirty pages becomes full Experimental. NOT EXTENSIVELY TESTED

- int MergePages (void ∗p, uintptr_t creator_addr)

  Tries to merge pages of current process.

- void MergeNode (const void ∗key, const void ∗value, const void ∗data)

  a node corresponding to <key, value> pair is checked so that identical data pages can be merged

- void FreeNode (const void ∗key, const void ∗value, const void ∗data, void ∗isDirty)

  Frees up a node corresponding to <key, value> pair.

- void ∗ GetSharedRegion (void ∗addr, bool isFixed, size_t size)

  Allocates a shared region.

- void ∗ GetSharedPage (void ∗addr, bool isFixed)

  Allocates a shared page Calls GetSharedRegion with addr, isFixed, PAGE_SIZE.

- void MergeNode2 (const void ∗key, const void ∗value, const void ∗data, void ∗isDirty)

  A node corresponding to <key, value> pair is checked so that identical data pages can be merged. Merges many pages at once.

- void MakeReadOnlyWrapper (void ∗addr, size_t len)

  makes a region readonly

- void MakeReadWriteWrapper (void ∗addr, size_t len)

  makes a region writeable

- uintptr_t GetBacktrace ()

  Finds address of the source that called malloc.

- void GetCallStack (void ∗∗stack, int depth)

  Stores the call stack when malloc is called. It excludes the library addresses from the call trace.

- int CopyAndRemapRegion (void ∗start, size_t size)

  Copies and maps pages from private region to shared space.

- int RemapRegion (void ∗start, size_t size)

  Maps pages from private region to shared space Some other process shared this memory region, so just remapping required here.

- int RemapToZero (void ∗start, size_t size)

  Remaps the pages to the zero page.

- int MergeManyPages (uintptr_t start_addr, size_t size, const void ∗data)

  Merges many pages This is the main routine that identifies the type of merge and invokes appropriate functions. It is a state machine based implementation to reduce fragmentation on the number of mmaps.

- bool CheckIfMergeable (void ∗pageAddr, uint32_t currTime)

  Checks merge profile to decide if the page should be merged If the page is mergeable, updates time.

- void UpdateMergeHist (void ∗pageAddr, uint32_t currTime)

  Updates merge profile. If the split is before a threshold of time, it is considered a failure.

6.8.1 Detailed Description

Header file for SBLLmalloc.

Author:

Susmit Biswas

Version:

1.0

Date:

2009-2010

6.8.2 Define Documentation

6.8.2.1 #define ASSERTX(a) assert(a)

If the evaluated expression is false, prints backtrace and exits.

6.8.2.2 #define BUFFER_LENGTH 10000

Used for buffer based merge as default frequency.

6.8.2.3 #define CheckForError()

Reports error if errno is set by printing source info and backtrace.

6.8.2.4 #define COLLECT_MALLOC_STAT

Enabled statistics collection for mallocs.

Attention:

Requires setting `SHARED_STATS`

6.8.2.5 #define die(m) {perror(m); fflush(stderr); MPI_Abort(MPI_COMM_-WORLD, MPI_ERR_OTHER);}

Issues warning message *m* and dies with `MPI_Abort`.

6.8.2.6 #define MALLOC_REF_FREQ 1000

Used for frequency based merge as default frequency.

6.8.2.7 #define NDEBUG

6.8.2.8 #define offset2ptr(x) ((void ∗)(x))

Converts uintptr_t to pointer.

6.8.2.9 #define PRINT_STATS

Enables profiling Collect sub-block merging stats Flags for controlling msg level.

6.8.2.10 #define ptr2offset(x) ((uintptr_t)x)

Converts pointer to uintptr_t.

6.8.2.11 #define ReportError(addr)

Reports error by printing error address and backtrace.

6.8.2.12 #define SHARED_STATS

Enables shared variables for statistics collection.

Attention:

REQUIRED if you plan on using threshold based merge (**recommended**)

6.8.2.13 #define SHARED_STATS

Enables shared variables for statistics collection.

Attention:

REQUIRED if you plan on using threshold based merge (**recommended**)

6.8.2.14 #define warn(msg)

Value:

```
{\
    fprintf(stderr, "%d:warning! %s:%d: %s\n", myRank, __FILE__, __LINE__, msg); \
    fflush(stderr); \
}
```

Issues warning message.

6.8.3 Enumeration Type Documentation

6.8.3.1 enum _MERGE_METRICS

Different merge metric. Set 0 to disable.

Enumerator:

- ***MERGE_DISABLED*** 0:Disable merging
- ***ALLOC_FREQUENCY*** 1:Frequency based merging
- ***THRESHOLD*** 2:Threshold based merging (recommended)
- ***BUFFERED*** 3:Buffered merging (**EXPERIMENTAL**, please do not use)
- ***NUM_METRIC*** Number of merge policies

6.8.3.2 enum _PROFILE_MODES

Different profile modes. Set 0 to disable.

Enumerator:

- ***NONE*** No profiling
- ***CREATE_PROF*** Create merge profile
- ***USE_PROF*** Use profiles for merging (**NOT** SUPPORTED)
- ***NUM_MODES*** Num profiling modes

6.8.4 Function Documentation

6.8.4.1 uintptr_t Addr2PageIndex (void $*$) `[inline]`

Translates page address to page number.

Returns:

Page number

6.8.4.2 void AllocateSharedMetadata ()

allocates a shared region to account for merged pages.

Returns:

None

6.8.4.3 int AspaceAvlInsertWrapper (uintptr_t *start_addr*, size_t *size*)

Inserts a node in address space AVL tree with start_addr and size. called upon malloc.

Returns:

0

6.8.4.4 intptr_t AspaceAvlRemoveWrapper (uintptr_t *start_addr*)

Deletes a node from AVL tree with start_addr. Called upon free.

Returns:

The size of the region, 0 if not found.

6.8.4.5 uintptr_t AspaceAvlSearchRangeWrapper (uintptr_t *start_addr*)

Searches a range with start_addr included in an existing range.

Returns:

0 if not found, the size of the region if found

6.8.4.6 uintptr_t AspaceAvlSearchWrapper (uintptr_t *start_addr*)

Searches a node with start_addr.

Returns:

0 if not found, the size of the region if found

6.8.4.7 int CeilLog2 (unsigned *long*) `[inline]`

Computes `ceil`(log2(n)).

See also:

FloorLog2

Returns:

-1 if n == 0.

6.8.4.8 void CheckEnv ()

Checks environment variables for sanity.

6.8.4.9 bool CheckIfMergeable (void ∗ *pageAddr*, uint32_t *currTime*)

Checks merge profile to decide if the page should be merged If the page is mergeable, updates time.

Parameters:

pageAddr Address of the page under consideration for merge.

currTime Current Time. lastMergeTime is updated if the page can be merged.

Returns:

Whether the page is mergeable

6.8.4.10 bool CheckMPIInitialized () `[inline]`

Checks if `MPI_Init` has been called or is not a MPI app.

Returns:

true when MPI has been Initialized/ not MPI app .

6.8.4.11 void CleanUpSharedData ()

Cleans up shared data: deletes AVL, closes files, unmaps shared region, destroys semaphores.

6.8.4.12 int CopyAndRemapRegion (void ∗ *start*, size_t *size*)

Copies and maps pages from private region to shared space.

Parameters:

start Address of the start of the region

size Size of the region

Returns:

-1 if failure, 0 if successful

6.8.4.13 int CountSharingProcs (void * *addr*)

Counts the number of tasks sharing a page.

Parameters:

addr Address of the page

Returns:

the Number of sharing processes

6.8.4.14 void Fatal ()

Aborts execution. Called upon encountering error.

6.8.4.15 int FloorLog2 (unsigned *long*) `[inline]`

Computes `floor`(log2(n)) Works by finding position of MSB set.

Returns:

-1 if n == 0.

6.8.4.16 void FreeNode (const void * *key*, const void * *value*, const void * *data*, void * *isDirty*) `[inline]`

Frees up a node corresponding to <key, value> pair.

Parameters:

key Address of start address of the region to check for merging

value Size of the region

data Address of the creator of the region (not used)

isDirty Flag indicating whether the region is dirty (not used)

6.8.4.17 uintptr_t GetBacktrace ()

Finds address of the source that called malloc.

Returns:

address outside this library that created this region.

6.8.4.18 bool GetBit (char $*$ *array*, char $*$ *page_addr*) `[inline]`

Gets bit corresponding to an address from bitvector array.

Parameters:

array Bit vector

page_addr Address of the page

Returns:

The bit for address page_addr from array bit vector

6.8.4.19 void GetCallStack (void $**$ *stack*, int *depth*)

Stores the call stack when malloc is called. It excludes the library addresses from the call trace.

Parameters:

stack an array to store the pointers the length of the array stack

6.8.4.20 void GetMemRange ()

Reads proc maps and finds the address range where this malloc library is allocated.

6.8.4.21 void$*$ GetSharedPage (void $*$ *addr*, bool *isFixed*) `[inline]`

Allocates a shared page Calls GetSharedRegion with addr, isFixed, PAGE_SIZE.

See also:

GetSharedRegion

Returns:

Address of the page

6.8.4.22 void$*$ GetSharedRegion (void $*$ *addr*, bool *isFixed*, size_t *size*)

Allocates a shared region.

Parameters:

isFixed If set, map at a fixed address

addr

size Size of the region

Returns:

Address of the mapped region or MAP_FAILED if error encountered

6.8.4.23 bool GetSharingBit (void ∗ *addr*) `[inline]`

Gets the sharing bit corresponding to page having address addr for currest process.

6.8.4.24 void InitAddrSpace ()

Initializes shared region and sets segfault handler.

Returns:

None

6.8.4.25 void InitEnv ()

Initializes the parameters for the library.

6.8.4.26 void InitSem (char ∗ *SEMKEY*, sem_t ∗∗ *mutex*)

Initializes a `POSIX` semaphore after getting it In linux, semkey has to be an existing filename beginning with / but not having more than 14 chars and more slashes.

Parameters:

SEMKEY Name of the semaphore

mutex The semaphore address

6.8.4.27 bool IsCloseToMmapLimit (int *newRequest* = `0`) `[inline]`

Checks if number of mmap calls are close to system limit.

Returns:

true if close to limit

Deprecated

By setting `vm.max_map_count` to a large value (e.g. 512K), we do not need this check

6.8.4.28 bool IsOtherSharing (void ∗ *addr*) `[inline]`

Checks any other process shares the page having address addr by checking sharingProcessesInfo bits.

6.8.4.29 void MakeReadOnlyWrapper (void ∗ *addr*, size_t *len*) `[inline]`

makes a region readonly

Parameters:

addr	Start address of the region
len	Size of the region

6.8.4.30 void MakeReadWriteWrapper (void ∗ *addr*, size_t *len*) `[inline]`

makes a region writeable

Parameters:

addr	Start address of the region
len	Size of the region

6.8.4.31 void MergeByALLOC_FREQUENCY ()

Merges pages based on frequency of mallocs. When the number of outstanding malloc/free becomes more than x where x is the merging frequency, merge operation is triggered.

6.8.4.32 void MergeByBUFFERED ()

merges pages when the buffer of dirty pages becomes full Experimental. NOT EXTENSIVELY TESTED

6.8.4.33 void MergeByTHRESHOLD ()

Merges pages based on threshold When the amount of used memory changes by more than 1000 pages i.e. 4MB, merge operation is triggered.

6.8.4.34 int MergeManyPages (uintptr_t *start_addr*, size_t *size*, const void ∗ *data*)

Merges many pages This is the main routine that identifies the type of merge and invokes appropriate functions. It is a state machine based implementation to reduce fragmentation on the number of mmaps.

Parameters:

start Address of the start of the region

size Size of the region

data Call stack of the region

Returns:

0 if successful, -1 otherwise

6.8.4.35 void MergeNode (const void ∗ *key*, const void ∗ *value*, const void ∗ *data*)

a node corresponding to <key, value> pair is checked so that identical data pages can be merged

Parameters:

key Address of start address of the region to check for merging

value Size of the region

data Address of the creator of the region

Deprecated

`MergeNode2` is more efficiently implemented

See also:

MergeNode2

6.8.4.36 void MergeNode2 (const void ∗ *key*, const void ∗ *value*, const void ∗ *data*, void ∗ *isDirty*)

A node corresponding to <key, value> pair is checked so that identical data pages can be merged. Merges many pages at once.

Parameters:

key Address of start address of the region to check for merging

value Size of the region

data Call stack when the region was created

isDirty Flag indicating if the region is dirty

6.8.4.37 int MergePages (void * *p*, uintptr_t *creator_addr*)

Tries to merge pages of current process.

Parameters:

p Address of the page to be compared

creator_addr Address of the creator of page p

Deprecated

This routine is not tested thouroughly, and `MergeByTHRESHOLD` is more apt.

Returns:

0 if not merged. 1 if merged

6.8.4.38 int MPI_Finalize ()

Replaces the `MPI_Finalize` to call `PMPI_Finalize()`.

6.8.4.39 int MPI_Init (int * *argc*, char *** *argv*)

Replaces `MPI_Init` of mpi library and initializes shared memory, metadata etc. param argc argc from main(int argc, char **argv) param argv argv from main(int argc, char **argv).

Returns:

Result of PMPI_Init

6.8.4.40 return munmap (__VA_ARGS__)

6.8.4.41 int MyComparator (const void * *key1*, const void * *key2*) `[inline]`

AVL data structure compartor.

Returns:

0 if identical, otherwise the difference as 1 or -1

6.8.4.42 void PrintMergeStat () `[inline]`

Flushes merge stat buffer to file.

6.8.4.43 int RemapRegion (void ∗ *start*, size_t *size*)

Maps pages from private region to shared space Some other process shared this memory region, so just remapping required here.

Parameters:

start Address of the start of the region
size Size of the region

Returns:

-1 if failure, 0 if successful

6.8.4.44 int RemapToZero (void ∗ *start*, size_t *size*)

Remaps the pages to the zero page.

Parameters:

start Address of the start of the region
size Size of the region

Returns:

0 if successful, -1 otherwise

6.8.4.45 bool ResetAndReturnBit (char ∗, char ∗) `[inline]`

Resets and returns old value of bit in array corresponding to page address.

6.8.4.46 bool SetAndReturnBit (char ∗, char ∗) `[inline]`

Sets and returns old value of bit in array corresponding to page address.

6.8.4.47 void SetBit (char ∗ *array*, char ∗ *page_addr*) `[inline]`

@ Sets bit corresponding to an address from bitvector array

Parameters:

array Bit vector
page_addr Address of the page

Returns:

None

6.8.4.48 void SetMultiBits (char ∗ *array*, char ∗ *page_addr*, size_t *size*)

@ Sets multiple bits corresponding to a region, more than 1 page from bitvector array

Parameters:

array	Bit vector
page_addr	Start address of the region
size	Size of the region

Returns:

None

6.8.4.49 void SetSharingBit (void ∗ *addr*) `[inline]`

Sets the sharing bit corresponding to page having address addr for currest process.

6.8.4.50 void∗ SH_MMAP (...) `[inline]`

6.8.4.51 int SH_UNMAP (...) `[inline]`

6.8.4.52 void SigBusHandler (int *signo*, siginfo_t ∗ *si*, void ∗ *sc*)

SIGBUS signal handler.

See also:

SigSegvHandler

6.8.4.53 void SigIntHandler (int *signo*, siginfo_t ∗ *si*, void ∗ *sc*)

SIGINT signal handler.

See also:

SigSegvHandler

6.8.4.54 void SignalSem (sem_t ∗ *mutex*) `[inline]`

operation **V:** used for semaphore handling

Parameters:

mutex	Address of the semaphore ∗

6.8.4.55 void SigSegvHandler (int *signo*, siginfo_t ∗ *si*, void ∗ *sc*)

SIGSEGV signal handler. Handles write faults for readonly marked shared pages. If the page was never touched, the permission bit is changed and returned. Otherwise, a copy of the page is created in private region.

Parameters:

signo The signal number

si `siginfo_t` provides the information regarding fault address and type

sc Not used in this routine

6.8.4.56 void StoreMemUsageStat ()

Collects current memory usage Not locking for aliveProcs as it is updated in the beginning of startup.

Returns:

None

6.8.4.57 uintptr_t TranslateMmapAddr (uintptr_t) `[inline]`

Translate mmapped address to file offset.

Parameters:

addr Address of the page to be translated

Returns:

0 if out of address range, translated address otherwise

6.8.4.58 void UnsetBit (char ∗ *array*, void ∗ *page_addr*) `[inline]`

@ Unsets bit corresponding to an address from bitvector array

Parameters:

array Bit vector

page_addr Address of the page

Returns:

None

6.8.4.59 void UnsetSharingBit (void ∗ *addr*) `[inline]`

Unsets the sharing bit corresponding to page having address addr for currest process.

6.8.4.60 void UpdateMergeHist (void ∗ *pageAddr*, uint32_t *currTime*)

Updates merge profile. If the split is before a threshold of time, it is considered a failure.

Parameters:

pageAddr Address of the page having segfault.

currTime Current Time. lastMergeTime is compared with it to see if the last merge was successful.

6.8.4.61 void UpdateMergeStat (const long int *tpm*, const long int *tptm*, const long int *tzm*, const long int *tsm*, const long int *tum*, const long int *tmm*, const int *mtm*) `[inline]`

Stores memory usage in a buffer and flushes to file when the buffer is full.

Parameters:

← ***tpm*** Total memory as private pages

← ***tptm*** Total memory used by internal allocator

← ***tzm*** Total zero memory in current process

← ***tsm*** Total shared memory usage

← ***tum*** Memory footprint if merging is disabled

← ***tmm*** Memory footprint with merging enabled

← ***mtm*** Time used for merging in microsecond

6.8.4.62 void WaitSem (sem_t ∗ *mutex*) `[inline]`

operation **P:** used for semaphore handling

Parameters:

mutex Address of the semaphore

7 SBLLmalloc Page Documentation

7.1 Todo List

File SharedHeap.cpp
- Modify backtrace capability:

1. Allocate an array for each of the AVL node instead of just creator (Done)
2. Store 20 entries from the callstack (Done)
3. At every merge point: find what pages are merged and what is the source i.e. the creator (Done)
4. Dump out the addresses. (Done)
5. After exit, run `addr2line` to find out the stack traces (Done)
6. Report top contributors as a graph.

- Recover from mmap failure

  1. We really cannot recover, we can just keep our finger crossed that no other library calls mmap.

- Support non-MPI apps by changing `MPI_Init` to a routine that runs at program startup

  1. Use `__attribute__` ((constructor)) `TheStartRoutineForNonMPIapp();`

7.2 Deprecated List

Member IsCloseToMmapLimit By setting `vm.max_map_count` to a large value (e.g. 512K), we do not need this check

Member MergeNode `MergeNode2` is more efficiently implemented

Member MergePages This routine is not tested thouroughly, and `MergeByTHRESHOLD` is more apt.

Index